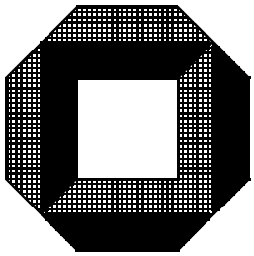

Einführung in C

W. Alex und G. Bernör

1995

Universität Karlsruhe

E-Mail: wulf.alex@.ciw.uni-karlsruhe.de

gerhard.bernoer@.ciw.uni-karlsruhe.de

WWW: http://www.ciw.uni-karlsruhe.de/mvm_www/lehre/22997.html

WWW: http://www.ciw.uni-karlsruhe.de/mvm_www/lehre/22998.html

Telefon: 0721/608-2404

Fax: 0721/693965

Ausgabedatum: 6. November 1995.

Dies ist ein Skriptum (Arbeitsfassung). Es ist unvollständig und enthält Fehler.

Geschützte Namen wie UNIX werden ohne Kennzeichnung verwendet.

Geschrieben mit dem Editor `vi(1)` auf einer Workstation Hewlett-Packard 9000/712 unter HP-UX (UNIX System V), formatiert mit LaTeX auf einem Pentium-90-PC unter LINUX, ausgegeben auf einem Hewlett-Packard Laserjet IIISi unter Verwendung von Postscript.

Alle Programmbeispiele sind im Internet mittels Anonymous-FTP von `ftp.ciw.uni-karlsruhe.de`, Verzeichnis `/c/pub/skriptum/...` abrufbar, ebenso die angeführte Elektronische Literatur im Verzeichnis `/c/pub/docs/...`. Die Programmbeispiele sind 100 % logisch abbaubar, können jedoch bei übermäßigem Genuß vorübergehende Störungen der Befindlichkeit verursachen.

Das Skriptum ist die Grundlage für das Buch *UNIX, C und Internet*, erschienen im Springer-Verlag unter der ISBN 3-540-57881-1.

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

Unser Buch wendet sich an Leser mit wenigen Vorkenntnissen in der Elektronischen Datenverarbeitung (EDV); es soll – wie FRITZ REUTERs *Urgeschicht von Meckelnborg* – ok för Schaulkinner tau bruken sin. Für die wissenschaftliche Welt 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, der Programmiersprache C und dem internationalen Computernetz Internet, die so weit reicht, daß der Leser selbständig weiterarbeiten kann. Ausgelernt hat man nie.

Der Text besteht aus acht Teilen. Nach anfänglichen Schritten zur Eingewöhnung in den Umgang mit dem Computer beschreibt der zweite Teil kurz die Hardware, der dritte die Betriebssystemfamilie UNIX, der vierte die Programmiersprache C, der fünfte das Internet mit Schwerpunkt Netzdienste, der sechste einige Anwendungen und der siebte Rechtsfragen im Zusammenhang mit der EDV. Ein Anhang enthält Fakten, die man immer wieder braucht. Für die zweite Auflage wurden viele Kleinigkeiten verbessert und die objektorientierten Zweige von C (Objective C, C++) berücksichtigt. Bei der Stoffauswahl haben wir uns von unserer Arbeit als Benutzer und Verwalter international vernetzter UNIX-Systeme sowie als Programmierer vorzugsweise in C und FORTRAN leiten lassen.

Hinsichtlich vieler Einzelheiten wird auf die Referenz-Handbücher und Manpages zu den Rechenanlagen und Programmiersprachen verwiesen. Wir wollen nicht den Text durch Dinge aufblähen, die man besser dort nachschlägt. Den Umfang haben wir auf knapp 700 Seiten beschränkt, um den Leser nicht durch zu hohe Anforderungen an seinen Geldbeutel und an seine Zeit abzuschrecken. *Alles über UNIX, C und das Internet* 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 örtliche 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.

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, aber nicht mit uns.

Das Internet ist das größte internationale Computernetz, eigentlich ein Zusammenschluß vieler regionaler Netze. Vor allem Universitäten und Behörden sind eingebunden, zum Teil auch Schulen, Organisationen und Industrie. Es ist nicht nur eine Daten-Autobahn, sondern eine ganze Landschaft. 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 bei, die Programme und ergänzende Texte stehen im Netz zur Verfügung. UNIX, C und das Internet könnten unabhängig voneinander betrachtet werden, in der Praxis jedoch sind sie miteinander verflochten, sie bilden ein Dreieck.

An einigen Stellen gehen wir außer auf das Wie auch auf das Warum ein. 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, einer Programmiersprache oder der Netzdienste nicht ohne praktische Übungen erlernen – das ist wie beim Klavierspielen oder Kuchenbacken. Die Beispiele und Übungen wurden auf einer Hewlett-Packard 9000/712 unter HP-UX (UNIX V) 9.0 und einem PC der Marke Weingartener Katzenberg Auslese unter Microsoft DOS 6.2, Microsoft Quick-C 2.5 und GNU gcc 2.6.3 sowie unter LINUX 1.2.8 entwickelt. Als Shell wurde die Korn-Shell (11/16/88) bevorzugt. Auf anderen Anlagen können sich geringe Abweichungen ergeben.

Dem Text liegen eigene Erfahrungen aus 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. Darüber hinaus haben wir nach Kräften das Internet angezapft und viele dort umlaufende Guides, Primers, Tutorials und Sammlungen von Frequently Asked Questions verwendet. Dem Springer-Verlag danken wir dafür, daß er uns geholfen hat, aus einem lockeren Skriptum ein ernsthaftes Buch zu machen, ohne unsere dichterische Freiheit zu beschneiden.

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 immer noch für unfertig und bitten, die Mängel zu entschuldigen.

Weingarten (Baden), 1. November 1995 Wulf Alex

Inhaltsverzeichnis

Abbildungen

Programme

Rien n'est simple.
Sempé

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),
- erzeugt oder verändert (Änderung 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 Anderen – 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 Anderer 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 oder löschen. Tätigkeiten, die mit Listen, Karteien, Rechenschemata zu tun haben oder die mit geringen Abweichungen 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.

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 wiederum geht auf lateinisch *computare* zurück, was berechnen, veranschlagen, erwägen, überlegen bedeutet. Die Franzosen sprechen vom *ordinateur*, dessen lateinischer

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

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 Begriff der Zahl 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[2] 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 im Altertum 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, mehr nicht. Es hat auch mechanische Rechenautomaten gegeben, die für spezielle Aufgaben wie die Berechnung von Geschoßbahnen oder Gezeiten eingerichtet waren. Das Verhalten von mechanischen Automaten ist durch ihre Mechanik unveränderlich vorgegeben.

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

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

gesetzt. 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* 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. Hingegen ist 1 bit (kleingeschrieben) die Maßeinheit für die Entscheidung zwischen 0 und 1 im Sinne der Informationstheorie von CLAUDE ELWOOD SHANNON. Kombinationen von acht 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 $2^8 = 256$ unterschiedliche Zeichen darstellen. Das reicht für unsere europäischen Buchstaben, Ziffern und Satzzeichen. Ebenso wird mit einem Byte eine Farbe aus 256 unterschiedlichen Farben ausgewählt. 1024 Byte ergeben 1 Kilobyte, 1024 Kilobyte sind 1 Megabyte, 1024 Megabyte sind 1 Gigabyte, 1024 Gigabyte machen 1 Terabyte.

Der Computer verarbeitet Informationen in Einheiten eines **Maschinenwortes**, das je nach der Breite der Datenregister des Prozessors ein bis 16 Bytes umfaßt. Der durchschnittliche Benutzer[3] kommt mit dieser Einheit selten in Berührung; für den Assembler-Programmierer sind die Datentypen am einfachsten, die sich gerade in einem Maschinenwort darstellen lassen.

[3]Unter Benutzer, Programmierer, System-Manager usw. verstehen wir auch ihre weiblichen Erscheinungsformen, ohne dieses besonders hervorzuheben.

1.2 Woraus besteht ein Computer?

Der Benutzer sieht von einem Computer vor allem den **Bildschirm**[4] (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.

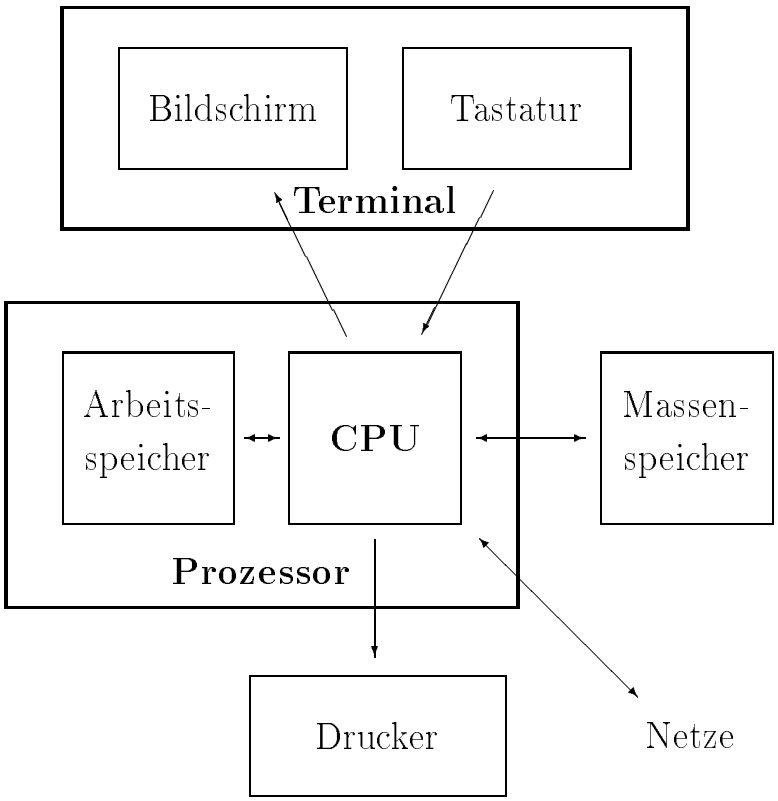

Abb. 1.1: Aufbau eines Computers

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.

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

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. 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[5]. Zum Arbeiten mit einem Computer muß man drei Dinge lernen:

- den Umgang mit der **Hardware**[6],
- 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. 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** 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:

[5] Auf PCs kann man durch Ändern einzelner Bits in den Registern von Grafikkarten den Bildschirm zerstören. Aus Versehen gerät man jedoch nie in diese Lage.

[6] Wir wissen, daß wir ein deutsch-englisches Kauderwelsch gebrauchen, aber wir haben schon so viele schlechte Übersetzungen der amerikanischen Fachwörter gelesen, daß wir der Deutlichkeit halber teilweise die amerikanischen Wörter vorziehen. Oft sind auch die deutschen Wörter mit unerwünschten Assoziationen befrachtet. 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?

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

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, DOS-7 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 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[7].

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

- 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

[7]Zum Vergleich: es gibt etwa 6000 lebende natürliche Sprachen. Die Bibel ist in etwa 2000 Sprachen übersetzt.

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) 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.

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 I *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[8]. 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.

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. 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

[8] 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.

zum Dialog mit den Urhebern. Wir haben versuchsweise einen Anonymous-FTP-Server `ftp.ciw.uni-karlsruhe.de` eingerichtet, auf dem ergänzende Informationen verfügbar sind, 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 steht vorn im Buch 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**[9], 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 voran.

Weiß auch Ihr Wizard nicht weiter, können Sie sich an die Öffentlichkeit wenden, das heißt an die schätzungsweise zehn Millionen Usenet-Teilnehmer. Den 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*[10] 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

[9]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 Anfang der 60er Jahre von TED NELSON in den USA geprägt.

[10]siehe Anhang F *Abkürzungen*

paar Tasten mehr. Oft gehört auch eine Maus dazu. 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[11].

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[12]. 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 **??** *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 dürfen 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 beginnt. Auf einem PC geben Sie beispielweise `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[13].

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

[11] Laden Sie ihn oder sie gelegentlich zu Kaffee und Kuchen oder einem Viertele Wein ein.

[12] 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

[13] Manche Systeme unterscheiden zwischen Return-Taste und Enter-Taste, rien n'est simple. Auf Tastaturen für den kirchlichen Gebrauch trägt die Taste die Bezeichnung Amen.

keine Bedeutung, die Sitzung läuft weiter!

Zum Eingewöhnen führen wir eine kleine Sitzung durch. 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 /bin
exit
```

Die Grundform eines **UNIX-Kommandos** ist (ähnlich wie bei MS-DOS):

```
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` (wie long) 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.

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

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

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[14]. Sie gliedern sich in folgende **Sektionen**:

- 1 Kommandos und Anwendungsprogramme
- 1M Kommandos zur Systemverwaltung (maintenance)
- 2 Systemaufrufe

[14]Real programmers don't read manuals, behauptet das Netz.

- 3C Subroutinen der Standard-C-Bibliothek
- 3M Mathematische Bibliothek
- 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 Erwähnen eines Kommandos wird die Sektion des Handbuchs in Klammern angegeben, da das gleiche 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 europäischer Absonderlichkeiten)

Bei vielen Kommandos finden sich nur Name, Synopsis und Description. Der Sinn 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 sein kann. Versuchen Sie folgende Eingabe:

```
man ls
man 2 time
man man
```

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*[15]. an, das die Einzelheiten der Kommandos bringt, auf die wir verzichten. Außerdem ist die Nutshell preiswert und nicht zu dick.

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 in die Tastatur hacken zu dürfen. Sehr junge Informatiker äußern 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 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 drei bis zehn 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

[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.

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 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 Hexenwerk, 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 **Verantwortung** 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 Telefon-

nummer 0721/19429 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] 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? Welche Welt ist wirklich? War *Der längste Tag* nur ein Bildschirmspektakel? 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. 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 – 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, zum Umblättern geeignetes Buch zurücklegen.

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

Brevity is the soul of wit.
Shakespeare, Hamlet

2 Programmieren in C

Dieses Kapitel erklärt die Kunst des Programmierens anhand der Sprache C, die sich besonders gut mit UNIX-Systemen verträgt. Grundkenntnisse im Programmieren in einer anderen Sprache (BASIC, FORTRAN, PASCAL, COBOL) sind hilfreich.

2.1 Sprachenfamilien

Hat man eine Aufgabe, ein Problem zu lösen, so kann man drei Abschnitte unterscheiden:

- Aufgabenstellung,
- Lösungsweg,
- Ergebnis.

Das Ergebnis ist nicht bekannt, sonst wäre die Aufgabe bereits gelöst. Die Aufgabenstellung und erforderlichenfalls 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 **algorithmische** oder **prozedurale** Programmiersprachen im weiteren Sinn bezeichnet, weil die Programme aus Prozeduren bestehen, die Anweisungen an den Computer enthalten (lateinisch *procedere* = vorangehen). Diese Familie wird unterteilt in die imperativen oder prozeduralen Sprachen im engeren Sinne einerseits und die objektorientierten Sprachen andererseits (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 gehen diesen Weg (lateinisch *declarare* = erklären, beschreiben). Die deklarativen Sprachen unterteilt man in die **funktionalen** und die **logischen** oder **prädikativen** Sprachen.

Wir haben also folgende Einteilung (wobei die tatsächlich benutzten Sprachen Mischlinge sind und die Einordnung ihrem am stärksten ausgeprägten Charakterzug folgt):

- Prozedurale Sprachen im weiteren Sinn
  - imperative, algorithmische, operative oder im engeren Sinn prozedurale Sprachen (BASIC, FORTRAN, COBOL, C, PASCAL)

  - objektorientierte Sprachen (SMALLTALK, C++)

- Deklarative Sprachen

  - funktionale oder applikative Sprachen (LISP, SCHEME)
  - logische oder prädikative Sprachen (PROLOG)

Diese Sprachentypen werden auch **Paradigmen** (Beispiel, Muster) genannt.

Die **objektorientierten** Programmiersprachen wie C++ teilen ein Programm in abgeschlossene Objekte auf, die miteinander über Nachrichten oder Botschaften verkehren. Das Innere der Objekte bleibt unzugänglich. Diese Sichtweise entspringt dem Bedürfnis, die einzelnen Programmteile voneinander unabhängig und damit das ganze Programm besser beherrschbar zu gestalten. SMALLTALK ist eine von Grund auf neue Sprache, C++ eine Weiterentwicklung von C. In C++ lassen sich daher auch Programme im gewohnten imperativen Stil schreiben.

Programme in funktionalen Programmiersprachen wie LISP oder SCHEME bestehen aus Definitionen von Funktionen, äußerlich ähnlich einem Gleichungssystem, die auf Listen von Werten angewendet werden. Die Sprache C ist trotz der Verwendung des Funktionsbegriffes keine funktionale Programmiersprache, da ihr Konzept nicht anders als in FORTRAN oder PASCAL auf der sequentiellen Ausführung von Anweisungen beruht.

Programmen in logischen Sprachen wie PROLOG werden Fakten und Regeln zum Folgern mitgegeben, es beantwortet dann die Anfrage, ob eine Behauptung mit den Fakten und Regeln verträglich (wahr) ist oder nicht. Viele Denksportaufgaben legen eine solche Sprache nahe. Die Umgewöhnung von einem Paradigma auf ein anderes geht über das Erlernen einer neuen Sprache hinaus und beeinflußt die Denkweise, die Sicht auf ein Problem.

Es gibt ein zweite, von der ersten unabhängige Einteilung, die zugleich die historische Entwicklung spiegelt:

- maschinenorientierte Sprachen (Maschinensprache, Assembler)
- problemorientierte Sprachen (höhere Sprachen)

In der Frühzeit gab es nur die völlig auf die Hardware ausgerichtete und unbequeme Maschinensprache. Assembler sind ein erster Schritt in Richtung auf die Probleme und die Programmierer zu. Höhere Sprachen wie FORTRAN sind von der Hardware schon ziemlich losgelöst und in diesem Fall an mathematische Probleme angepaßt. Es gibt aber für spezielle Aufgaben wie Stringverarbeitung, Datenbankabfragen oder Grafik Sprachen, die in ihrer Anpassung noch weiter gehen. Auch die zur Formatierung des vorliegenden Textes benutzte Sammlung von LaTeX-Makros stellt eine problemangepaßte Sprache dar. Der Preis für die Erleichterungen ist ein Verlust an Allgemeinheit. Denken Sie an die Notensprache der Musik: an ihre Aufgabe bestens angepaßt, aber für andere Gebiete wie etwa die Mathematik ungeeignet.

2.2 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 Z22 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 2.1 : Ausschnitt aus einem Programm für die ZUSE Z22

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 mathematisch 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 2.18.3 *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*. Daß Leutnant Grace M. Hopper (eine Frau) sowohl den ersten Bug erlegt wie auch COBOL

[1]Recht betrachtet, will man auch keine Gleichungen, sondern Aufgaben wie die Dimensionierung eines Maschinenteils oder das Zusammenstellen eines Sachregisters lösen.

erfunden habe, ist eine Legende um ein Körnchen Wahrheit herum. COBOL ist 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 2.2 : 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*. JOHN KEMENY und THOMAS 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, spielte 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 nicht-britischen 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 2.18.3 *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 eine Sprache **C++** entwickelt, die ebenfalls eine Übermenge bildet. Der Denkansatz (Paradigma) 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. 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, siehe das Buch von RUDI KLATTE et al.

Auf die übrigen 992 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 Codierung – sondern in der Formulierung und Strukturierung der Aufgabe.

[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.

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.

2.3 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 **Compilertreiber** aufgerufen, so daß der Benutzer nichts von den beiden Schritten bemerkt. Man 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 Pro-

gramm ist **binär-kompatibel** zu zwei Systemen, 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.

Einen Weg zurück vom ausführbaren Programm zum Quellcode gibt es nicht. Das Äußerste ist, mit einem **Disassembler** aus dem ausführbaren Code Assemblercode zu erzeugen, ohne Kommentar und typografische Struktur. Nur bei kurzen, einfachen Programmen ist dieser Assemblercode verständlich.

2.4 Wie entsteht eine Programmiersprache?

Von den Nullen und Einsen im Befehlsregister des Prozessors bis zu einer Anweisung einer höheren Programmiersprache ist ein weiter Weg.

2.5 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 seltenen 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. Das Schlimmste ist, wenn ein Programm trotz eines Fehlers ein scheinbar richtiges Ergebnis ausgibt. 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. Nach umfangreichen Änderungen – möglichst Verbesserungen – erscheint eine neue **Version** des Programmes. Ein Programm, von dem nicht 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 besondere 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.

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.

C läßt dem Programmierer viel Freiheit, mehr als PASCAL. Damit nun nicht jeder schreibt, wie ihm der Schnabel gewachsen ist, hat die Programmierergemeinschaft Regeln und Gebräuche entwickelt. Ein Verstoß dagegen beeindruckt den Compiler nicht, aber das Programm ist mühsam zu lesen. Der Beautifier `cb(1)` automatisiert die Einhaltung einiger dieser Regeln, weitergehende finden sich in:

- Nelson Ford, Programmer's Guide, siehe Anhang,
- B. W. Kernighan, P. J. Plauger, Software Tools, siehe Anhang,
- Rob Pike, Notes on Programming in C, `/pub/../pikestyle.ps` auf `ftp.ciw.uni-karlsruhe.de`
- Firmen-Richtlinien wie Nixdorf Computer C-Programmierrichtlinien (Hausstandard), 1985
- K. Henning, Portables Programmieren in C – Programmierrichtlinien, verfaßt 1993 vom Hochschuldidaktischen Zentrum und vom Fachgebiet Kybernetische Verfahren und Didaktik der Ingenieurwissenschaften der RWTH Aachen im Auftrag von sechs Chemiefirmen. Der Verbreitung dieser Richtlinien

stehen leider ein Hinweis auf das Urheberrecht sowie ein ausdrückliches Kopierverbot entgegen.

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 Menüs, 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.

2.6 Programmiertechnik

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 zu dieser Programmiertechnik. 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 fünf Stufen vor (waterfall approach):

- Aufgabenstellung (Formulierung)
- 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 Buchmanuskripte.

Der steigende Bedarf an Software und ihre wachsende Komplexität verlangen die Entwicklung von Programmierverfahren, mit denen durchschnittliche Programmierer zuverlässige Programme entwickeln. Auf geniale *Real Programmers* allein kann sich keine Firma verlassen. Die Entwicklung dieser Programmiertechnik (Software Engineering) ist noch nicht abgeschlossen.

2.7 Aufgabenanalyse und Entwurf

2.7.1 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?

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. Anpassungsfähigkeit ist für Software und Lebewesen wichtiger als Höchstleistungen.

2.7.2 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)

[7] Real programmers know better than the users what they need.

[8] Real programmers disdain structured programming.

- 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.

2.7.3 Zusammensetzen aus Teilaufgaben

Der umgekehrte Weg – **Bottom-up-Entwurf** – liegt nicht so nahe. Es gibt wiederkehrende **Grund-Operationen** wie Suchen, Sortieren, Fragen, Ausgeben, 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 Anspruch auf ein gut und verständlich funktionierendes Programm. Die Zeiten, als der Computer als Entschuldigung für alle möglichen Unzulänglichkeiten herhalten mußte, sind vorbei.

2.8 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 nur ihren Namen auf den Bildschirm. Von einem menugesteuerten Vokabeltrainer beispielsweise schreibt man zunächst 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.

2.9 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 halbwegs überschaubare Grafik zu beschreiben.

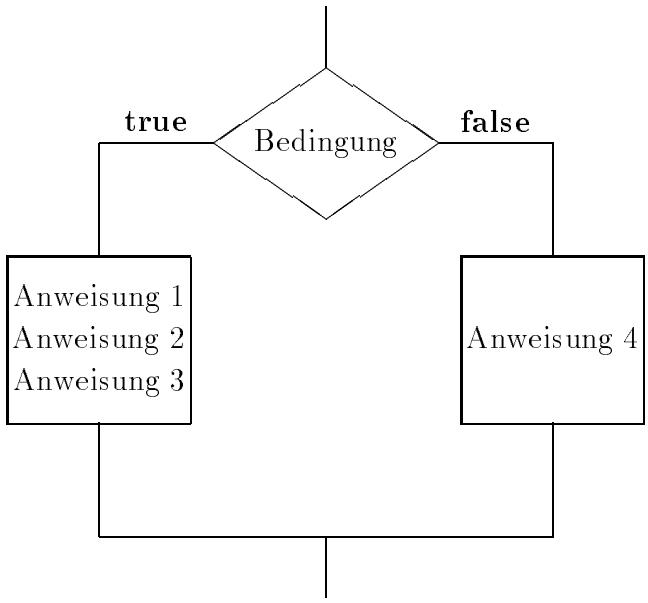

Abb. 2.1: Flußdiagramm einer if-else-Verzweigung

Flußdiagramme (flow chart), auch Blockdiagramme genannt, sollen die Abläufe innerhalb eines Programmes durch Sinnbilder nach DIN 66 001 und Text

[9] Real programmers don't draw flowcharts.

darstellen, unabhängig von einer Programmiersprache. Obwohl das Flußdiagramm vor dem Programmcode erstellt werden sollte, halten sich viele Programmierer nicht an diese Reihenfolge. Zum Teil ersetzt eine gute typografische Gestaltung der Programmquelle auch ein Flußdiagramm, während das Umgekehrte nicht gilt. Ein Flußdiagramm ist nicht mit einem Syntaxdiagramm zu verwechseln, lesen Sie die beiden entsprechenden Abbildungen, die die if-else-Verzweigung darstellen, einmal laut vor.

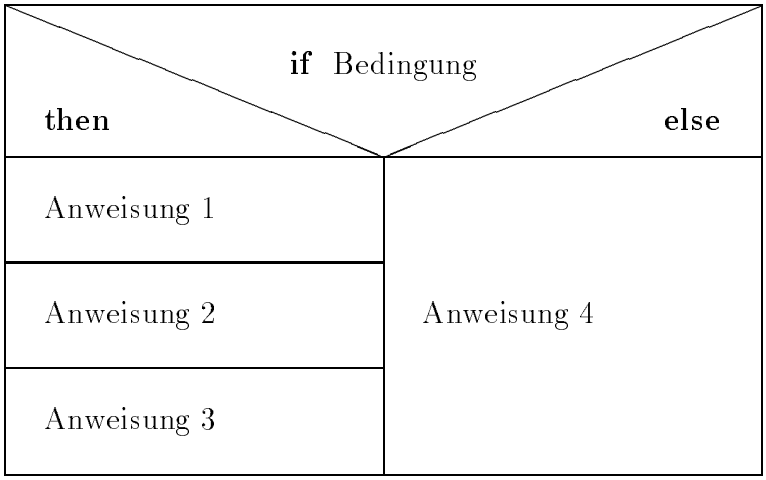

Abb. 2.2: Nassi-Shneiderman-Diagramm einer if-else-Verzweigung

Nassi-Shneiderman-Diagramme oder Struktogramme nach ISAAC NASSI und BEN SHNEIDERMAN sind ein weiterer Versuch, den Programmablauf grafisch darzustellen. Sie sind näher an eine Programmiersprache angelehnt, so daß es leicht fällt, nach dem Diagramm eine Quelle zu schreiben. Das läßt sich teilweise sogar mit CASE-Werkzeugen in beide Richtungen automatisieren.

2.10 Memo Programmiertechnik

- Nichts.

2.11 Übung Programmiertechnik

2.12 Bausteine eines Quelltextes

Alle Zeichen oder Zeichengruppen eines Programmes im Quellcode sind entweder

- Kommentar (comment),
- Namen (identifier),
- Schlüsselwörter (Wortsymbole) (keyword),
- Operatoren (operator),

- Konstanten (Literale) (constant, literal),
- Trennzeichen (separator) 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 Spaces, Semikolons und Linefeeds. Bedeutungslose Zeichen sind überzählige Spaces, Tabs oder Linefeeds.

Die Syntax der einzelnen Bausteine – d. h. 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. 2.3 stellen wir die Syntax zweier 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 (true – false),
- 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.

Ein weiterer Weg zur Beschreibung der Syntax einer Programmiersprache ist die Backus-Naur-Form, die von JOHN BACKUS, einem der Väter von FORTRAN, und PETER NAUR, einem der Väter von ALGOL, als Metasprache zu ALGOL 60 entwickelt worden ist. Weiteres siehe bei D. GRIES.

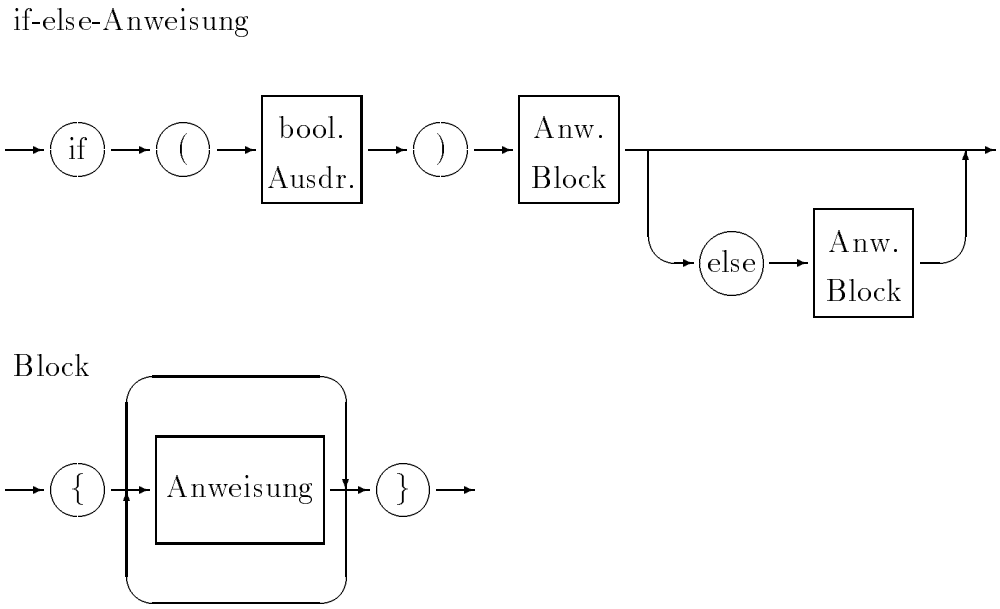

Abb. 2.3: Syntax-Diagramm der if-else-Anweisung und des Blockes

2.13 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 Zeile");    /* Kommentar */
/* Kommentar */     puts("Dritte Zeile");
```

```
/*
puts("Vierte Zeile");    /* Kommentar geschachtelt!!! */
*/

puts("Ende");
return 0;
}
```

Programm 2.3 : 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 (Sankt-Nimmerleins-Tag) verschoben wird.

2.14 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**[11] 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 D.1 *C-Lexikon, Schlüsselwörter*.

[10] Real programmers don't comment their code.

[11] 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.

2.15 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. Diese Objekte und ihre Untereinheiten nennen wir **Operanden** (operand). Mit ihnen werden Operationen durchgeführt. Das Wort *Objekt* vermeiden wir, um keine Assoziationen an objektorientiertes Programmieren zu wecken. Das kommt später. Ein Operand

- hat einen **Namen** (identifier),
- gehört einem **Typ** (type) an,
- hat einen konstanten oder variablen **Wert** (value),
- belegt zur Laufzeit **Speicherplatz** im Computer,
- hat einen **Geltungsbereich** (scope) und
- eine **Lebensdauer** (lifetime).

Die Speicheradresse wird auch **Zeiger** oder **Pointer** genannt. Wir bevorzugen das englische Wort *Pointer*, weil das deutsche Wort *Zeiger* drei Bedeutungen hat (Pointer, Index, Cursor). Auf den Operanden wird über den Namen oder den Pointer zugegriffen. Der Geltungsbereich ist ein Block, eine Funktion oder das ganze Programm. Ähnliches gilt für die Lebensdauer. 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 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.

2.15.1 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 (magic numbers) 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.

2.15.2 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, d. h. ein Operand, der einmal als ganzzahlig deklariert worden ist, bleibt dies während des ganzen Programmes. Einige Programmiersprachen erlauben auch variable Typen, die erst zur Laufzeit bestimmt werden oder sich während dieser ändern. Typfreie Sprachen kennen nur das Byte oder das Maschinenwort als Datentyp. Die Typisierung[12] erleichtert die Arbeit und erhöht die Sicherheit. Stellen Sie sich vor, Sie müßten bei Gleitkommazahlen Exponent und Mantisse jedesmal selbst aus den Bytes herausdröseln.

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.

2.15.2.1 Einfache Typen

In jeder Programmiersprache gibt es Grundtypen, aus denen alle höheren Typen zusammengesetzt werden. In C sind dies ganze Zahlen, Gleitkommazahlen und Zeichen.

[12] Real programmers don't worry about types.

Ganze Zahlen In C gibt es ganze Zahlen mit oder ohne Vorzeichen sowie in halber, 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 0 vor dem Dezimalkomma oder -punkt 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 **??** *Arithmetikprozessoren*.

Alphanumerischer Typ Eine Größe, deren Wertevorrat die Zeichen der ASCII-Tabelle oder einer anderen 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. Man mache sich die Gemeinsamkeiten an einem kleinen Programm klar:

```
/* Programm zum Demonstrieren von character und integer */

#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 2.4 : 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 links 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 will man ein Byte wahlweise als Ganzzahl oder als Zeichen auffassen, aber das gehört zu den berüchtigten Tricks in C.

Das Ausgabegerät empfängt 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) 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.

Leerer Typ Der leere Datentyp **void** wird 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. Man hätte den Typ auch `byte` nennen können.

2.15.2.2 Zusammengesetzte Typen

Arrays Die meisten Programmiersprachen kennen **Arrays**, unglücklicherweise auch als **Felder**[13] bezeichnet; das sind geordnete Mengen von Größen desselben Typs. Jedem Element ist ein fortlaufender Index (Hausnummer) zugeordnet, der in C stets mit 0 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, Unions oder Pointer sein, jedoch keine Funktionen. Files sind formal Strukturen, ein Array von Files ist also erlaubt.

Der Compiler muß die Größe eines Arrays (Anzahl und Typ der Elemente) 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 2.21.12 *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. Eine Matrix wird zeilenweise gespeichert. Vorsicht beim Übertragen von oder nach FORTRAN: dort läuft die Indizierung anders als in C, eine Matrix wird spaltenweise gespeichert. PASCAL verhält sich wie C.

Der **Name** eines Arrays ist der Pointer auf sein erstes Element. Er ist eine Pointerkonstante und kann daher nicht auf der linken Seite einer Zuweisung vorkommen. Weiteres dazu im Abschnitt 2.15.2.5 *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.

[13] Felder in Datensätzen sind etwas völlig anderes.

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 2.21.10.3 *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, bestehend aus alphanumerischen und numerischen Komponenten. Mit den numerischen wird gerechnet, auf die alphanumerischen werden Stringfunktionen angewendet. Telefonnummern oder Postleitzahlen sind alphanumerische Größen, da Rechenoperationen mit ihnen sinnlos sind.

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. In C 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.

2.15.2.3 Union

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, in PASCAL der variante Record. Eine Union belegt so viele Bytes wie der längste in ihr untergebrachte Datentyp.

2.15.2.4 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`. Auch Farben bieten sich für einen Aufzählungstyp an.

2.15.2.5 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. Genaugenommen gehören die Adressen zur Hardware und sind für den Programmierer in der Regel bedeutungslos, während die Pointer eine Einrichtung der Programmiersprache sind und zur Laufzeit mit den Adressen verknüpft werden. Adressen sind hexadezimale Zahlen, Pointer haben Namen. 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[14]. 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 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.

[14]Der in ANSI-C eingeführte Pointer auf den Typ `void` ist ein Pointer, der zunächst auf keinen bestimmten Typ zeigt.

Ein Pointer, der auf die Adresse `NULL` verweist, wird **Nullpointer** genannt und zeigt auf kein gültiges Datenobjekt. Sein Auftreten kennzeichnet eine Ausnahme oder einen Fehler. Der Wert `NULL` ist der einzige, der direkt einem Pointer zugewiesen werden kann; jede Zuweisung einer Ganzzahl ist ein Fehler, da Pointer keine Ganzzahlen sind. Ansonsten dürfen nur Werte, die sich aus einer Pointeroperation oder einer entsprechenden Funktion (deren Ergebnis ein Pointer ist) einem Pointer zugewiesen werden.

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		Speicher	Variable = Wert
a	$\longrightarrow$	4	$*a = a[0] = 4$
$a+1$	$\longrightarrow$	7	$*(a+1) = a[1] = 7$
$a+2$	$\longrightarrow$	1	$*(a+2) = a[2] = 1$
$a+3$	$\longrightarrow$	2	$*(a+3) = a[3] = 2$

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 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 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 $\otimes$. 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	$*(s+3) = s[3] = X$
$s+4$	$\longrightarrow$	$\otimes$	$*(s+4) = s[4] = \otimes$
$s+5$	$\longrightarrow$	??	$*(s+5) = s[5] =$??

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 Spaces 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. Gespeichert sind 12 Ganzzahlen, die Pointer 1. Ordnung sind nicht gespeichert. 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 `?? 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 Spaces aufgefüllt auf gleiche Länge:

```
char *ptag[] = {"Sonntag, ", "Montag,  ", ...};
```

Hier wird ein Array von 7 Pointern gespeichert. Dazu kommen natürlich noch die Strings, die wegen des Aussehens auf dem Bildschirm gleich lang sind, aber vom Programm her ungleich lang sein dürften.

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	$*(ptag[0]+6) = ptag[0][6] = g$
	$\longrightarrow$	$ptag[0]+7$	$\longrightarrow$	,	$*(ptag[0]+7) = ptag[0][7] =,$
	$\longrightarrow$	$ptag[0]+8$	$\longrightarrow$		$*(ptag[0]+8) = ptag[0][8] =$
	$\longrightarrow$	$ptag[0]+9$	$\longrightarrow$	$\otimes$	$*(ptag[0]+9) = ptag[0][9] = \otimes$
$ptag+1$	$\longrightarrow$	$ptag[1]$	$\longrightarrow$	M	$*ptag[1] = ptag[1][0] = M$
	$\longrightarrow$	$ptag[1]+1$	$\longrightarrow$	o	$*(ptag[1]+1) = ptag[1][1] = o$
	$\longrightarrow$	$ptag[1]+2$	$\longrightarrow$	n	$*(ptag[1]+2) = ptag[1][2] = n$

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.

2.15.2.6 Arrays und Pointer

2.15.2.7 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 Prüfung 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 Typnamen `BOOLEAN` schaffen, der zwar im Grunde nichts anderes ist als der Typ `int`, aber die Verwendung deutlicher erkennen läßt. Zu Beginn der Deklarationen oder vor `main()` schreiben wir:

```
typedef   int   BOOLEAN;
```

(die Großschreibung ist nicht zwingend) und können anschließend eine Variable `janein` als `BOOLEAN` deklarieren

```
BOOLEAN    janein;
```

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 arithmetischen 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.

2.15.3 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 Defaultklasse für **lokale Variable** und braucht nicht eigens angegeben zu werden. Variablen dieser Klasse leben nur innerhalb des Blockes, in dem sie deklariert wurden, und sterben beim Verlassen des Blockes.

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 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.

2.15.4 Geltungsbereich

Eine Variable gilt nur innerhalb des Bereiches, zu dessen Beginn sie deklariert worden ist. Ihr **Geltungsbereich** (scope) ist dieser Bereich. Außerhalb des zugehörigen Bereiches ist die Variable unbekannt oder unsichtbar. Der Name der Variablen ist in diesem Zusammenhang bedeutungslos. Ein Bereich 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 Bereiches nochmals deklariert, so hat für diesen Bereich die lokale Deklaration Vorrang vor der äußeren Deklaration. Der Geltungsbereich der äußeren Variablen hat eine Lücke.

Das Konzept des Geltungsbereiches läßt sich über ein Programm hinaus erweitern. Die Umgebungs-Variablen der Sitzungsshell gelten für alle Prozesse einer Sitzung und können von diesen abgefragt oder verändert werden. Darüber hinaus sind auch Variable denkbar, die in einem Verzeichnis, einem Filesystem oder in einer Netz-Domain gelten. Je größer der Geltungsbereich ist, desto sorgfältiger muß man mit der Schreibberechtigung umgehen.

2.15.5 Lebensdauer

Beim Eintritt in einen Bereich wird für die in diesem Bereich deklarierten Variablen Speicher zugewiesen (allokiert). Beim Verlassen des Bereiches wird der Speicher freigegeben, von den Variablen bleibt keine Spur zurück. Ihre **Lebensdauer** ist die aktive Zeitspanne des Bereiches. Beim nächsten Aufruf des Bereiches 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.

2.15.6 Deklaration und Definition von Operanden

2.15.7 Memo Operanden

- Nichts.

2.15.8 Übung Operanden

2.16 Operationen

2.16.1 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 geschieht. 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.

2.16.2 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 Spaces 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 eine Konstante, 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

$$x = x + 1 \tag{2.1}$$

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 von x. 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

$$x + 2 = 5 \tag{2.2}$$

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.

2.16.3 Arithmetische Operationen

Die **arithmetischen Operationen** sind:

- Vorzeichenumkehr `-` (unärer Operator)
- Addition `+`
- Subtraktion `-` (binärer Operator)
- Multiplikation `*`
- Division `/`

- Modulus `%` (Divisionsrest, 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 ausführliche Schreibweise bleibt weiterhin erlaubt und wird nur seltener verwendet.

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.

2.16.4 Logische Operationen

Die **logischen Operationen** sind:

- bitweise Negation (not) ~ (Tilde)
- ausdrucksweise Negation (not) !
- bitweises Und (and) &
- ausdrucksweises Und (and) &&
- bitweises exklusives Oder (xor) ^ (Circumflex, caret)
- bitweises inklusives Oder (or) |
- ausdrucksweises inklusives Oder (or) ||

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 0 als logisch falsch (false), jeder Wert ungleich 0 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 0, bei dem auch alle Bits auf 0 stehen. Sowie ein beliebiges Bit auf 1 stünde, hätte die Variable einen Wert ungleich 0 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 2.5 : 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-0 oder 0). Bitweise logische Operationen können jedes beliebige Ergebnis im Bereich der ganzen Zahlen haben.

2.16.5 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 zusammengesetzt 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 2.18.7 *Rekursiver Aufruf einer Funktion*.

2.16.6 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. Zur Veranschaulichung der Bitoperationen ein kleines Programm:

```
/* 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 2.6 : C-Programm mit Bitoperationen

2.16.7 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.

2.16.8 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)`.

2.16.9 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 skalare Typbezeichnung ohne Speicherklasse und geht dem umzuwandelnden skalaren 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. Bei der Typumwandlung können Bits oder Dezimalstellen verlorengehen, wenn der neue Typ kürzer ist als der alte. Die Typumwandlung gilt nur im Zusammenhang mit dem cast-Operator und hat für die weiteren Anwendungen der Variablen keine Bedeutung. Manche Compiler beanstanden einen cast-Operator auf der linken Seite einer Zuweisung.

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.

2.16.10 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
*  /  %
+  -
<<  >>
<  <=  >  >=
==  !=
&
^
|
&&
||
?  :
=  +=  -=  *=  /=  %=  <<=  >>=  &=  ^=  |=
,
```

Zuerst werden also die runden Klammern ausgewertet; wie in der Mathematik haben sie Vorrang vor allen anderen Operatoren.

2.16.11 Memo Operationen

- Nichts.

2.16.12 Übung Operationen

2.17 Anweisungen

2.17.1 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). Ein Beispiel:

```
while ((c = getchar()) != 125);
```

Die Schleife liest Zeichen ein und verwirft sie, bis sie auf ein Zeichen Nr. 125 (rechte geschweifte Klammer) trifft. Das wird auch noch entsorgt, dann geht es nach der Schleife weiter.

2.17.2 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);
```

Ähnlich wie die Return-Taste in der Kommandozeile bewirkt hier erst das Semikolon, daß etwas geschieht.

2.17.3 Kontrollstrukturen

Kontrollstrukturen steuern in Abhängigkeit von dem Wert eines booleschen 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.

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 typgleichen 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 nach `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).

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 der Schleifenrumpf betreten. Zuletzt wird der Ausdruck `inkrementierung` ausgewertet und zur Bedingung zurückgesprungen. Die `for`-Schleife in C hat also eine andere Bedeutung als die `for`-Schleife der Shell oder der Programmiersprache PASCAL. Jeder der drei Ausdrücke darf fehlen:

```
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 Wert gleich oder ungleich 0 ergeben. Eine `continue`-Anweisung innerhalb des Rumpfs führt wieder zur Initialisierung. Zwei Beispiele:

```
/* Beispiel fuer for-Schleife, 04.03.1993 */

#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 2.7 : 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, die Ausführung des Programms geht nach der Schleife weiter. Nun eine leicht verrückte `for`-Schleife:

```
/* Testen der for-Schleife, 04.03.1993 */

#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 2.8 : C-Programm mit zusammengesetzter 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 vor die Schleife. 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 1993 in einem Vortrag bemerkte, daß die klassischen Schleifen mit ihrer harten Ja-nein-Entscheidung das wirkliche 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[15]). Die zugehörige Variable ist der **Schleifenzähler**. Auf seine 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 ein `goto` das Programm verbesern, meist ist es vom Übel, weil es erstens sehr gefährlich, zweitens auch nicht nötig ist[16].

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

[15] 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?

[16] Real programmers aren't afraid to use `goto`'s.

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 2.9 : C-Programm mit goto, grauenvoll

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 2.10 : C-Programm, verbessert

Am `goto` hatte sich um 1970 herum 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.

2.17.4 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 2.11 : C-Programm mit return-Anweisungen

Im Hauptprogramm `main()` haben `return(n);` und `exit(n);` dieselbe Wirkung. In anderen Funktionen führt `return` zur Rückkehr in die nächsthöhere Einheit, `exit` zum Abbruch des gesamten Programmes. In der Syntax unterscheiden sich beide Aufrufe: `return` ist ein Schlüsselwort von C, `exit()` ein Systemaufruf von UNIX, also eine Funktion. Weiterhin sind `exit` und `return` auch eingebaute Shell-Kommandos – siehe `sh(1)` oder `ksh(1)` – die aber nicht in C-Programmen vorkommen können.

2.17.5 Memo Anweisungen

- Nichts.

2.17.6 Übung Anweisungen

2.18 Funktionen

2.18.1 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 */
.
.
}

float multipl(float x, float y)   /* F´definition */
{
return (x * y);
}
```

Programm 2.12 : 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.

2.18.2 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 2.20 *Präprozessor*.

2.18.3 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 kommt der gleiche Satz von Variablen – gegebenenfalls unter anderem Namen – mit jeweils aktuellen Werten vor; sie werden als **aktuelle Parameter**

oder Argumente bezeichnet. Die Schnittstelle von Programm und Funktion muß zusammenpassen wie Stecker und Kupplung einer elektrischen Verbindung, d. h. 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 aufrufenden Programmes übergeben. Daraus folgt, daß die Funktion die aktuellen Parameter des aufrufenden Programmes nicht verändern kann.

Bei der **Adressübergabe** (call by reference) werden der Funktion die Speicheradressen der aktuellen Parameter des aufrufenden Programmes übergeben. Die Funktion kann daher die Werte der Parameter mit Wirkung für das aufrufende Programm 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 aufrufenden Programm ersetzt. Call by name ist selten und soll in ALGOL 60 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.

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 Shellscript 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 (mixed language programming). 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 2.13 : 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 2.14 : 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 (in FORTRAN 90 ist es möglich). 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 2.15 : 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 2.16 : 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 2.17 : 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 2.18 : 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 2.19 : 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 2.20 : 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 2.21 : 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 2.22 : 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 2.23 : FORTRAN-Programm, das Parameter by value und by reference an PASCAL-Funktionen übergibt

Die FORTRAN-Compiler-Anweisung `$ALIAS` veranlaßt den Compiler, der jeweiligen Funktion die Parameter entgegen seiner Gewohnheit by value zu überge-

ben. 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 2.24 : 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 2.25 : 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 2.26 : PASCAL-Programm, das Parameter by value und by reference an PASCAL-Funktionen übergibt

Sollten Sie die Beispiele nachvollzogen haben, müßte Ihr 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. Hinter dem Filepointer bzw. der Unit steckt der Bildschirm. 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 (eine Ausgabe in ein File wäre möglich):

```
{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 2.27 : 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 2.28 : 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 Pro-

gramme 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 2.29 : 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 2.30 : C-Programm, das Parameter von einem Shellscript übernimmt

Ferner können Shellscripts **Shellfunktionen** aufrufen, siehe das Shellscript **??** *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.

2.18.4 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 Spaces 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 folgendermaßen 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 2.31 : 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).

2.18.5 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 2.32 : C-Funktion mit wechselnder Anzahl von Argumenten

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 Argumentenliste. 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 Argumentenliste 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.

2.18.6 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 2.33 : 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.

2.18.7 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 nicht geheuer. Es gibt aber Probleme, die ursprünglich rekursiv sind und sich durch eine Rekursion elegant programmieren lassen. Eine **Zirkeldefinition** ist eine Definition eines Begriffes, die diesen selbst in der Definition enthält, damit es nicht sofort auffällt, gegebenenfalls um einige Ecken herum. Ein **Zirkelschluß** ist eine Folgerung, die Teile der zu beweisenden Aussage bereits zur Voraussetzung hat. Bei einer Rekursion hingegen

- wiederholt sich die Ausgangslage nie,

- wird eine Abbruchbedingung nach endlich vielen Schritten erfüllt, d. h. die Rekursionstiefe ist begrenzt.

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 2.34 : 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 */

#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));
```

```
    return 0;
}

/* funktion fak() */

int fak(int n)
{
    return(n <= 1 ? 1 : n * fak(n - 1));
}
```

Programm 2.35 : C-Programm zur rekursiven Berechnung der Fakultät

Weitere rekursiv lösbare Aufgaben sind die Türme von Hanoi und Quicksort. Rekursive Probleme lassen sich auch iterativ lösen. Das kann sogar schneller gehen, aber die Eleganz bleibt auf der Strecke.

Da in C auch das Hauptprogramm `main()` eine Funktion ist, die auf gleicher Stufe mit allen anderen Funktionen steht, kann es sich selbst aufrufen:

```
/* Experimentelles Programm mit Selbstaufruf von main() */

#include <stdio.h>

int main()
{
puts("Selbstaufruf von main()");
main();
return(13);
}
```

Programm 2.36 : 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.

2.18.8 Assemblerroutinen

Auf die Assemblerprogrammierung wurde in Abschnitt 2.2 *Programmiersprachen* bereits eingegangen. Da das Schreiben von Programmen in Assembler mühsam ist und die Programme nicht portierbar sind, läßt man 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.

2.18.9 Memo Funktionen

- Nichts.

2.18.10 Übung Funktionen

2.19 Funktions-Bibliotheken

2.19.1 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, muß aber für dieselbe Maschine compiliert sein.
- Die getrennt compilierte Funktion steht zusammen mit weiteren in einer Bibliothek 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 ?? *Bibliotheken, Archive* im Rahmen der Programmer's Workbench erläutert. Im folgenden geht es um die Verwendung von Bibliotheken.

2.19.2 Standardbibliothek

2.19.2.1 Übersicht

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. Die Systemaufrufe (Sektion 2) gehören dagegen nicht zur Standardbibliothek (Sektion 3), sondern zum Betriebssystem. Und Shell-Kommandos sind eine Sache der Shell (Sektion 1).

Die mit dem C-Compiler eines UNIX-Systems mitgelieferte Standardbibliothek wird im Referenz-Handbuch unter `intro(3)` vorgestellt und umfaßt mehrere Teile:

- die Standard-C-Bibliothek, meist gekoppelt mit der Standard-Input-Output-Bibliothek, den Netzfunktionen und den Systemaufrufen (weil sie zusammen gebraucht werden),
- die mathematische Bibliothek,
- gegebenenfalls eine grafische Bibliothek,
- gegebenenfalls eine Bibliothek mit Funktionen zum Messen und Regeln,
- gegebenenfalls Datenbankfunktionen und weitere Spezialitäten.

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`).

2.19.2.2 Standard-C-Bibliothek

Die Standard-C-Bibliothek `/lib/libc.a` wird 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 2.37 : C-Programm zur Stringverarbeitung

Internet-Funktionen Eine Übersicht über diese Funktionen findet sich in `intro(3N)`. Beispiele sind Funktionen zur Verarbeitung von Netzadressen, Protokolleinträgen, Remote Procedure Calls, zum Mounten entfernter File-Systeme, zur Verwaltung von Benutzern und Passwörtern im Netz. Geht über den Rahmen dieses Textes hinaus. Falls Sie sich ein eigenes Programm `telnet` oder `ftp` schreiben wollten, müßten Sie hier tiefer einsteigen.

2.19.2.3 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. Ferner muß im Programmkopf die Zeile

```
#include <math.h>
```

stehen. Dann verfügt man über Logarithmus, Wurzel, Potenz, trigonometrische und hyperbolische Funktionen. Weiteres siehe `math(5)`.

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 2.38 : 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.

2.19.2.4 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. Leider nicht standardisiert, sonst gäbe es nicht Starbase, GKS, PHIGS, Uniras ...

2.19.2.5 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.

2.19.3 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.

2.19.4 Weitere Bibliotheken

2.19.4.1 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.

2.19.5 Eigene Bibliotheken

Wir haben bereits in Abschnitt ?? *Bibliotheken, Archive* gelernt, eine eigene Programmbibliothek mittels des UNIX-Kommandos `ar(1)` herzustellen. Zunächst macht es Arbeit, seine Programmierergebnisse in eine Bibliothek einzuordnen, aber wenn man einmal einen Grundstock hat, zahlt es sich aus.

2.19.6 Speichermodelle (MS-DOS)

Unter UNIX gibt es keine Speichermodelle, infolgedessen auch nur eine Standardbibliothek. Unter MS-DOS hingegen ist die Speichersegmentierung zu beachten, d. h. 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, 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.

2.19.7 Memo Bibliotheken

- Nichts.

2.19.8 Übung Bibliotheken

2.20 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.

2.20.1 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. Bei Strings sind die Gänsefüßchen mit in die Definition zu nehmen, man darf 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 Prográmmchen:

```
/* 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
```

2.20.2 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 2.39 : Include-File /usr/include/stdio.h, gekürzt

2.20.3 Bedingte Compilation (#ifdef)

2.20.4 Memo Präprozessor

- Nichts.

2.20.5 Übung Präprozessor

2.21 C-Programme

2.21.1 Name

Obwohl es aus den bisherigen Beispielen klar geworden sein müßte, weisen wir nochmals darauf hin: Jedes selbständige C-Programm heißt im Quellcode `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.

2.21.2 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. Dem Compiler übergibt man im Minimum ein File, das eine Funktion enthält. Mehrere Files hinwiederum können ein vollständiges, nach dem Compilieren lauffähiges **Programm** bilden. Erinnern Sie sich an das Werkzeug `make(1)`?

Das Minimalprogramm in C und C++ besteht aus einer Funktion – nämlich `main()` – deren Rumpf leer ist, in einem File:

```
main()
{
}
```

Programm 2.40 : 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 2.41 : C-Programm, einfachst

Dieses Programm wird vom `lint(1)` gutgeheißen. Die Deklaration von `main()`

als `int` könnte entfallen, da sie unabänderlich 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 2.42 : 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 2.43 : 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 2.44 : 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 2.45 : 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!

2.21.3 Fehlersuche

Einige Gesichtspunkte sind bereits im Abschnitt ?? *Debugger* behandelt worden. 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.

Die Programmeinheiten (Funktionen) lasse man nicht zu umfangreich werden. Ein bis zwei Seiten Quelltext überschaut man noch, wird es mehr, sollte man die Funktion unterteilen. Weiterhin gebe man im Entwicklungsstadium an kritischen Stellen die Werte mittels `printf()` oder `fprintf()` aus. Diese Zeilen kommentiert man später aus oder klammert sie gleich in `#ifdef-` und `#endif`-Anweisungen ein. Bewährt hat sich auch, die eigenen Programme einem anderen zu erklären, da wundert man sich manchmal über den eigenen Code. Ein Programm, das man nach ein bis zwei Wochen Pause selbst nicht mehr versteht, war von vornherein nicht gelungen.

Und wenn dann der Computerfreak zu nächtlicher Stunde den Bugs hinterherjagt, schließt sich ein weiter Bogen zurück in die Kreidezeit, denn die ersten Säugetiere – Zeitgenossen der Saurier – waren auch nachtjagende Insektenfresser.

2.21.4 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 manchmal. 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 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.

2.21.5 2 mal 2 ist nicht 4

2.21.6 Der elliptische Kreis

2.21.7 Die schöne Julia

2.21.8 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(4)` 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 */
/* Compileraufruf cc -o xclear xclear.c -lcurses */
/* falls terminfo-Fkt. verwendet werden sollen, noch
   -DTERMINFO anhaengen */

#include <curses.h>        /* enthaelt stdio.h */

#ifdef TERMINFO
#include <term.h>          /* nur fuer terminfo */
#endif

int main(argc, argv)

int argc;
char *argv[];

{
int i;
```

```
if (argc > 1) {     /* Files leeren, nicht loeschen */

    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 {

#ifdef TERMINFO    /* Bildschirm leeren, terminfo-Routinen */

    setupterm(0, 1, 0);
    putp(clear_screen);
    resetterm();

#else              /* Bildschirm leeren, curses-Routinen */

    initscr();
    refresh();
    endwin();

#endif

}
return 0;
}
```

Programm 2.46 : 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 2.47 : 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(3)`-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`, d. h. 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.

2.21.9 Wie blind ist der Zufall?

2.21.10 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 nichts angeht.
- 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.

2.21.10.1 Erzeugen von Pointern: pdemo.c

2.21.10.2 Der Nullpointer: nullp.c

2.21.10.3 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 man 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 2.48 : C-Programm mit Pointer auf void

Die Funktion `xread()` braucht als erstes Argument einen Pointer (aus demselben 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.

2.21.10.4 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 den Gewohnheiten entgegenkommt. Bei der Optimierung wurden alle Indizes durch Pointer ersetzt, wie im Abschnitt 2.21.4 *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 -DUNIX */

/* 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. */

#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>

/* globale Variable */

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 */

/* Funktionsprototypen */

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("\nZ. zu gross! Maximal %lu\n", MAX);
        exit(1);
    }
        if (ende < MIN) {
        printf("\nZ. zu klein; genommen wird %lu\n\n\n", MIN);
        ende = MIN;
    }
    if (g = ende % 10) {
    printf("\nZ. muss durch 10 teilbar sein: %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)+i) );

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 2.49 : 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! Am Algorithmus und seiner Verwirklichung etwas zu optimieren, bringt für die Gesamtdauer praktisch nichts. Die Ausgabe-Funktion `printf(3)` ließe sich durch eine selbstgeschriebene, schnellere Funktion ersetzenr, unter Abstrichen an der Allgemeinheit.

2.21.10.5 Strings und Strukturen

2.21.10.6 Pointer auf Funktionen

2.21.11 Ein Dämon

Beim Schreiben eines Programms, das als Dämon laufen soll, müssen die besonderen Lebensumstände eines Dämons berücksichtigt werden:

2.21.12 Dynamische Speicherverwaltung (malloc)

Wir haben gelernt, daß die Größe eines Arrays oder einer Struktur bereits zur Übersetzungszeit bekannt sein, d. h. 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), 01.06.94  */
```

```
#define MAX 40
#define DELTA 2

#include <stdio.h>
#include <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.
   In stdlib.h wird size_t als unsigned int definiert.          */

la = (long *)calloc((size_t)MAX, (size_t)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((void *)la, (size_t)(DELTA * sizeof(long)));

/* Array anschauen */

la[MAX + DELTA] = x;

printf("erweitert: %ld    %ld\n", la[10], la[MAX + DELTA]);

/* Speicher freigeben mit free() */

free((void *)la);

return 0;
}
```

Programm 2.50 : 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 2.51 : C-Programm zum Sortieren eines Textes nach den Regeln des Duden

Die Variable `flag`, die auch anders heißen kann, ist ein **Flag**, d. h. eine Variable, die in Abhängigkeit von bestimmten Bedingungen einen Wert 0 oder nicht-0 annimmt und ihrerseits wieder in anderen Bedingungen auftritt. Ein gängiger, einwandfreier Programmiertrick.

2.21.13 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 2.52 : C-Programm für X-Window-System

2.22 Obfuscated C

2.23 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 2.18.3 *Parameterübergabe* bekommen. Unter folgenden Voraussetzungen geht es 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.

2.24 Einbinden von PASCAL-Modulen

2.25 Dokumentation

indexProgramm!Dokumentation Die **Dokumentation** dient dazu, ein Programm im Quellcode einem menschlichen Leser verständlicher zu machen. Längere undokumentierte Programme sind nicht nachzuvollziehen. Eine Dokumentation[17] 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

[17] Real programmers write programs, not documentation.

- externe Unterprogramme, soweit verwendet

2. Anwendung
   - Aufruf
   - Konstante, Variable
   - Eingabe (von Tastatur, aus Files)
   - Ausgabe (zum Bildschirm, Drucker, in Files)
   - Einschränkungen
   - Fehlermeldungen
   - Beispiel
   - 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.

2.26 Portieren von Programmen

2.26.1 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 portieren, 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 Portieren; es gibt auch Programme für diese Tätigkeit[18]. Die wirklichen Aufgaben liegen zwischen diesen beiden Grenzfällen.

Schon beim ersten Schreiben eines Programmes erleichtert man ein künftiges Portieren, wenn man einige Regeln beherzigt. Man vermeide:

- Annahmen über Eigenheiten des Filesystems (z. B. Länge der Namen),
- Annahmen über die Reihenfolge der Auswertung von Ausdrücken, Funktionsargumenten oder Nebeneffekten (z. B. bei `printf(3)`),
- Annahmen über die Anordnung der Daten im Arbeitsspeicher,
- Annahmen über die Anzahl der signifikanten Zeichen von Namen,
- Annahmen über die automatische Initialisierung von Variablen,
- die Dereferenzierung von Nullpointern (Null ist keine Adresse),
- 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 (Alignment),
- die Annahme, daß Groß- und Kleinbuchstaben unterschieden werden,
- die Annahme, daß der Typ `char` vorzeichenbehaftet oder vorzeichenlos ist (EOF = -1?),
- Bitoperationen mit vorzeichenbehafteten Ganzzahlen,
- die Verwendung von Bitfeldern mit anderen Typen als `unsigned`,
- Annahmen über das Vorzeichen des Divisionsrestes bei der ganzzahligen Division,
- 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,

[18] Im GNU-Projekt finden sich ein Program `f2c` (lies: f to c) zum Übertragen von FORTRAN nach C und ein Programm `p2c` zum Portieren von PASCAL nach C.

- 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.

2.26.2 Beispiel Lineares Gleichungssystem: lin.c

2.26.3 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 geht 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 2.53 : ALGOL-Programm ggT nach Euklid

Die Einlese- und Übersetzungszeit auf einer Z22 betrug 50 s, die Rechen- und Druckzeit 39 s. 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
   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 2.54 : 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 2.34 *C-Programm ggt nach Euklid, rekursiv* sieht man, wie unterschiedlich selbst ein so einfacher Algorithmus programmiert werden kann. Dazu kommen andere Algorithmen zur Lösung dersel-

ben Aufgabe, beispielsweise das Ermitteln aller Teiler der beiden Zahlen und das Herausfischen des ggT.

2.26.4 Übertragen von PASCAL nach C

2.26.5 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     ------------------------------------------------------
      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 2.55 : 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
   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 2.56 : C-Programm Quadratische Gleichung mit reellen Koeffizienten und komplexen Lösungen, 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: Schließlich wollen wir das Programm als Funktion (Subroutine) schreiben, die von einem übergeordneten Programm aufgerufen wird:

2.26.6 Übertragen von BASIC nach C

2.26.7 Übertragen von LISP nach C

2.27 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 2.5 *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.

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.

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).

Menü, curses Schreiben Sie das Shellscript **??** *Menü* 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.

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.

Systemaufrufe, Signale Schreiben sie ein Programm, das alle möglichen Auskünfte über sich und das System, auf dem ausgeführt wird, gibt.

Rekursion Schreiben sie das Shellscript **??** *Hanoi* in C um. Lesen Sie über das Sortierverfahren Quicksort nach und vollziehen Sie ein entsprechendes Programmbeispiel nach.

Bibliothek Bruchrechnung Erfinden sie einen Datentyp *Bruch* und schreiben Sie eine Bibliothek für die Grundrechenarten. Zeigen Sie den Gebrauch.

Übertragen Übertragen Sie das folgende einfache BASIC-Programm, das die Primzahlen mit dem Sieb des ERATHOSTENES berechnet, nach C:

```
10 REM Sieb des Erathostenes (Primzahlen)
20 REM B. 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 2.57 : BASIC-Programm Sieb des Erathostenes

2.28 Objektiv betrachtet: C++ und Objective C

2.28.1 Objekte, warum und wie?

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 RCS 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 vielleicht 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[19].

2.28.1.1 Objective C und NeXTstep

2.28.1.2 C++

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. Im GNU-Projekt gibt es einen C++-Compiler, der auf dem GNU-C-Compiler aufbaut. Auch für MS-DOS verfügbar, unter `djgpp` suchen.

[19] In unserem Institut wurde viele Jahre ein Buchhaltungsprogramm in der etwas exotischen Sprache DE/RPG verwendet, das auch fast nur aus einer Beschreibung der Daten bestand.

Hier als kleines Beispiel das Hello-World-Programm in C++:

2.28.2 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,
- Behandlung bestimmter Fehler (Indexgrenzen),
- weitere Funktionen zur numerischen Analysis.

Näheres ist in dem Buch von Rudi Klatte nachzulesen.

2.28.3 Memo C++/Objective C

- Nichts.

2.28.4 Übung C++/Objective C

2.29 Exkurs über Algorithmen

Der Begriff **Algorithmus** – benannt nach einem usbekischen Mathematiker des 9. Jahrhunderts – kommt im vorliegenden Text selten vor, taucht aber in fast allen Programmierbüchern auf. Ein beträchtlicher Teil der Informatik befaßt sich damit. 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. Ein Existenzbeweis ist in der Mathematik schon ein Erfolg, in der Technik brauchen wir einen Lösungsweg, einen Algorithmus.

Das klingt alltäglich. Das Rezept zum Backen einer Prinzregententorte[20] oder die Beschreibung des Aufstiegs auf die Hochwilde in den Ötztaler Alpen[21], dem Hausberg des Hochwildehauses der Sektion Karlsruhe des Deutschen Alpenvereins, sind demnach Algorithmen. Einige Anforderungen an Algorithmen sind:

[20] Dr. Oetker Backen macht Freude, Ceres-Verlag, Bielefeld. Die Ausführung dieses Algorithmus läßt sich teilweise parallelisieren.

[21] H. Klier, Alpenvereinsführer Ötztaler Alpen, Bergverlag Rudolf Rother, München. Der Algorithmus muß sequentiell abgeschwitzt werden.

- **Korrektheit**. Das klingt selbstverständlich, ist aber meist schwierig zu beweisen. Und Korrektheit in einem Einzelfall besagt gar nichts. Umgekehrt beweist bereits ein Fehler die Inkorrektheit.
- **Eindeutigkeit**. Das stellt Anforderungen an die Darstellungsweise, die Sprache; denken Sie an eine technische Zeichnung oder an Klaviernoten. Verschiedene Ausführungswege sind zulässig, bei gleichen Eingaben muß das gleiche Ergebnis herauskommen.
- **Endlichkeit**. Die Beschreibung des Algorithmus muß eine endliche Länge haben, sonst könnte man ihn endlichen Wesen nicht mitteilen. Er muß ferner eine endliche Ausführungszeit haben, man möchte seine Früchte ja noch zu Lebzeiten ernten. Er darf zur Ausführung nur eine endliche Menge von Betriebsmitteln belegen.
- **Allgemeinheit**. $3 \times 4 = 12$ ist kein Algorithmus, wohl aber die Vorschrift, wie man die Multiplikation auf die Addition zurückführt.

Man kann die Anforderungen herabschrauben und kommt dabei zu reizvollen Fragestellungen, aber für den Anfang gilt obiges. Eine vierte, technisch wie theoretisch bedeutsame Forderung ist die nach einem guten, zweckmäßigen Algorithmus oder gar die nach dem besten. Denken Sie an die vielen Sortierverfahren (es gibt kein *bestes* für alle Fälle).

Es gibt – sogar ziemlich leicht verständliche – Aufgaben, die nicht mittels eines Algorithmus zu lösen sind. Falls Sie Bedarf an solchen Nüssen haben, suchen Sie unter dem Stichwort *Entscheidbarkeit* in Werken zur Theoretischen Informatik.

2.30 Exkurs über Zahlen

2.31 Exkurs über Aussagenlogik

In Abschnitt 2.16.4 *Logische Operationen* haben wir logische Operationen kennengelernt. Da sie vielleicht nicht so geläufig sind wie arithmetische Operationen, hier eine kurze Übersicht.

... aber die Daten fehlen, um den ganzen
Nonsens richtig zu überblicken –
Benn, Drei alte Männer

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	^
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

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	;	^	^	^
95	137	^	_	_	_
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		y
153	231	r		Ö
154	232	a		Ü
155	233	o		
156	234	æ		£
157	235	–		Yen
158	236	Æ		Pt
159	237			*f*
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	a
167	247	x	Ï	o
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	f	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		f	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	,
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	Ø
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	a	$\bullet$
250	372		o	$\cdot$
251	373	Û	$\ll$	$\surd$
252	374	Ü	⊔	n
253	375	Ù	$\gg$	2
254	376	Ú	±	⊔
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 ^
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 oder das Cent-Zeichen konnten mit LaTeX nicht dargestellt werden.

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	^	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	«	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	º	Masculine ordinal indicator

187	273	≫	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	ï	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	ÿ	Latin small letter y with diaresis

C 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 Programmes
`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.

D C-Lexikon

D.1 Schlüsselwörter

Laut ANSI verwendet C folgende Schlüsselwörter (Wortsymbole, keywords):

- Deklaratoren
  - `auto`, Default-Speicherklasse
  - `char`, Zeichentyp
  - `const`, Typattribut (ANSI-C)
  - `double`, Typ Gleitkommazahl doppelter Genauigkeit
  - `enum`, Aufzählungstyp
  - `extern`, Speicherklasse
  - `float`, Typ Gleitkommazahl einfacher Genauigkeit
  - `int`, Typ Ganzzahl einfacher Länge
  - `long`, Typ Ganzzahl doppelter Länge
  - `register`, Speicherklasse Registervariable
  - `short`, Typ Ganzzahl halber Länge
  - `signed`, Typzusatz zu Ganzzahl oder Zeichen
  - `static`, Speicherklasse
  - `struct`, Strukturtyp
  - `typedef`, Definition eines benutzereigenen Typs
  - `union`, Typ Union
  - `unsigned`, Typzusatz zu Ganzzahl oder Zeichen
  - `void`, leerer Typ
  - `volatile`, Typattribut (ANSI-C)
- Schleifen und Bedingungen (Kontrollstrukturen)
  - `break`, Verlassen einer Schleife
  - `case`, Fall einer Auswahl (`switch`)
  - `continue`, Rücksprung vor eine Schleife
  - `default`, Default-Fall einer Auswahl (`switch`)
  - `do`, Beginn einer `do`-Schleife
  - `else`, Alternative einer Verzweigung

  - `for`, Beginn einer `for`-Schleife
  - `goto`, unbedingter Sprung
  - `if`, Bedingung oder Beginn einer Verzweigung
  - `switch`, Beginn einer Auswahl
  - `while`, Beginn einer `while`-Schleife
- Sonstige
  - `return`, Rücksprung in die aufrufende Einheit
  - `sizeof`, Bytebedarf eines Typs oder einer Variablen

Darüberhinaus verwenden einige Compiler – vor allem aus der PC-Welt – weitere Schlüsselwörter. Die folgende Aufzählung ist nicht vollständig:

- `asm`, Assembler-Aufruf innerhalb einer C-Quelle
- `cdecl`, Aufruf einer Funktion nach C-Konventionen
- `entry`, (war in K&R-C für künftigen Gebrauch vorgesehen)
- `far`, Typzusatz unter MS-DOS
- `fortran`, Aufruf einer Funktion nach FORTRAN-Konventionen
- `huge`, Typzusatz unter MS-DOS
- `near`, Typzusatz unter MS-DOS
- `pascal`, Aufruf einer Funktion nach PASCAL-Konventionen

D.2 Standardfunktionen

Folgende Standardfunktionen oder -makros sind gebräuchlich:

- Pufferbehandlung
  - `memchr`, sucht Zeichen im Puffer
  - `memcmp`, vergleicht Zeichen mit Pufferinhalt
  - `memcpy`, kopiert Zeichen in Puffern
  - `memset`, setzt Puffer auf bestimmtes Zeichen
- Zeichenbehandlung
  - `isalnum`, prüft Zeichen, ob alphanumerisch
  - `isalpha`, prüft Zeichen, ob Buchstabe
  - `iscntrl`, prüft Zeichen, ob Kontrollzeichen
  - `isdigit`, prüft Zeichen, ob Ziffer
  - `isgraph`, prüft Zeichen, ob sichtbar
  - `islower`, prüft Zeichen, ob Kleinbuchstabe

  - `isprint`, prüft Zeichen, ob druckbar
  - `ispunct`, prüft Zeichen, ob Satzzeichen
  - `isspace`, prüft Zeichen, ob Whitespace
  - `isupper`, prüft Zeichen. ob Großbuchstabe
  - `isxdigit`, prüft Zeichen, ob hexadezimale Ziffer
  - `tolower`, wandelt Großbuchstaben in Kleinbuchstaben um
  - `toupper`, wandelt Kleinbuchstaben in Großbuchstaben um

- Datenumwandlung
  - `atof`, wandelt String in `double`-Wert um
  - `atoi`, wandelt String in `int`-Wert um
  - `atol`, wandelt String in `long`-Wert um
  - `strtod`, wandelt String in `double`-Wert um
  - `strtol`, wandelt String in `long`-Wert um
  - `strtoul`, wandelt String in `unsigned long`-Wert um

- Filebehandlung
  - `remove`, löscht File
  - `rename`, ändert Namen eines Files

- Ein- und Ausgabe
  - `clearerr`, löscht Fehlermeldung eines Filepointers
  - `fclose`, schließt Filepointer
  - `fflush`, leert Puffer eines Filepointers
  - `fgetc`, liest Zeichen von Filepointer
  - `fgetpos`, ermittelt Stand des Lesezeigers
  - `fgets`, liest String von Filepointer
  - `fopen`, öffnet Filepointer
  - `fprintf`, schreibt formatiert nach Filepointer
  - `fputc`, schreibt Zeichen nach Filepointer
  - `fputs`, schreibt String nach Filepointer
  - `fread`, liest Bytes von Filepointer
  - `freopen`, ersetzt geöffneten Filepointer
  - `fscanf`, liest formatiert von Filepointer
  - `fseek`, setzt Lesezeiger auf bestimmte Stelle
  - `fsetpos`, setzt Lesezeiger auf bestimmte Stelle
  - `ftell`, ermittelt Stellung des Lesezeigers

  - `fwrite`, schreibt Bytes nach Filepointer
  - `getc`, liest Zeichen von Filepointer
  - `getchar`, liest Zeichen von `stdin`
  - `gets`, liest String von Filepointer
  - `printf`, schreibt formatiert nach `stdout`
  - `putc`, schreibt Zeichen nach Filepointer
  - `putchar`, schreibt Zeichen nach `stdout`
  - `puts`, schreibt String nach Filepointer
  - `rewind`, setzt Lesezeiger auf Fileanfang
  - `scanf`, liest formatiert von `stdin`
  - `setbuf`, ordnet einem Filepointer einen Puffer zu
  - `setvbuf`, ordnet einem Filepointer einen Puffer zu
  - `sprintf`, schreibt formatiert in einen String
  - `sscanf`, liest formatiert aus einem String
  - `tempnam`, erzeugt einen temporären Filenamen
  - `tmpfile`, erzeugt ein temporäres File
  - `ungetc`, schreibt letztes gelesenes Zeichen zurück
  - `vfprintf`, schreibt formatiert aus einer Argumentenliste
  - `vprintf`, schreibt formatiert aus einer Argumentenliste
  - `vsprintf`, schreibt formatiert aus einer Argumentenlsite

- Mathematik
  - `acos`, arcus cosinus
  - `asin`, arcus sinus
  - `atan`, arcus tangens
  - `atan2`, arcus tangens, erweiterter Bereich
  - `ceil`, kleinste Ganzzahl
  - `cos`, cosinus
  - `cosh`, cosinus hyperbolicus
  - `exp`, Exponentialfunktion
  - `fabs`, Absolutwert, Betrag
  - `floor`, größte Ganzzahl
  - `fmod`, Divisionsrest
  - `frexp`, teilt Gleitkommazahl auf
  - `ldexp`, teilt Gleitkommazahl auf
  - `log`, Logarithmus naturalis

  - `log10`, dekadischer Logarithmus
  - `modf`, teilt Gleitkommazahl auf
  - `pow`, allgemeine Potenz
  - `sin`, sinus
  - `sinh`, sinus hyperbolicus
  - `sqrt`, positive Quadratwurzel
  - `tan`, tangens
  - `tanh`, tangens hyperbolicus

- Speicherzuweisung
  - `calloc`, allokiert Speicher für Array
  - `free`, gibt allokierten Speicher frei
  - `malloc`, allokiert Speicher
  - `realloc`, ändert Größe des allokierten Speichers

- Prozesssteuerung
  - `abort`, erzeugt SIGABRT-Signal
  - `atexit`, Funktionsaufruf bei Programmende
  - `exit`, Programmende
  - `raise`, sendet Signal
  - `signal`, legt Antwort auf Siganl fest
  - `system`, übergibt Argument an Kommandointerpreter

- Suchen und Sortieren
  - `bsearch`, binäre Suche
  - `qsort`, Quicksort

- Stringbehandlung
  - `strcat`, verkettet Strings
  - `strchr`, sucht Zeichen in String
  - `strcmp`, vergleicht Strings
  - `strcpy`, kopiert String
  - `strcspn`, sucht Teilstring
  - `strerror`, verweist auf Fehlermeldung
  - `strlen`, ermittelt Stringlänge
  - `strncat`, verkettet n Zeichen von Strings
  - `strncmp`, vergleicht n Zeichen von Strings

- `strncpy`, kopiert n Zeichen eines Strings
- `strpbrk`, sucht Zeichen in String
- `strrchr`, sucht Zeichen in String
- `strspn`, ermittelt Länge eines Teilstrings
- `strstr`, sucht Zeichen in String

Dies sind alle Funktionen des ANSI-Vorschlags. Die meisten Compiler bieten darüberhinaus eine Vielzahl weiterer Funktionen, die das Programmieren erleichtern, aber die Portabilität verschlechtern.

D.3 Include-Files

Die Standard-Include-Files enthalten in lesbarer Form Definitionen von Konstanten und Typen, Deklarationen von Funktionen und Makrodefinitionen. Sie werden von Systemaufrufen und Bibliotheksfunktionen benötigt. Bei der Beschreibung jeder Funktion im Referenz-Handbuch ist angegeben, welche Include-Files jeweils eingebunden werden müssen. Gebräuchliche Include-Files sind:

- `ctype.h`, Definition von Zeichenklassen (`conv(3)`)
- `curses.h`, Bildschirmsteuerung (`curses(3)`)
- `errno.h`, Fehlermeldungen des Systems (`errno(2)`)
- `fcntl.h`, Steuerung des Filezugriffs (`fcntl(2), open(2)`)
- `malloc.h`, Speicherallokierung (`malloc(3)`)
- `math.h`, mathematische Funktionen (`log(3), sqrt(3), floor(3)`)
- `memory.h`, Speicherfunktionen (`memory(3)`)
- `search.h`, Suchfunktionen (`bsearch(3)`)
- `signal.h`, Signalbehandlung (`signal(2)`)
- `stdio.h`, Ein- und Ausgabe (`printf(3), scanf(3), fopen(3)`)
- `string.h`, Stringbehandlung (`string(3)`)
- `time.h`, Zeitfunktionen (`ctime(3)`)
- `varargs.h`, Argumentenliste variabler Länge (`vprintf(3)`)
- `sys\ioctl.h`, Ein- und Ausgabe (`ioctl(2)`)
- `sys/stat.h`, Zugriffsrechte (`chmod(2), mkdir(2), stat(2)`)
- `sys/types.h` , verschiedene Deklarationen (`chmod(2), getut(3)`)

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 bei Compilern unter MS-DOS für PCs.

D.4 Präprozessor-Anweisungen

Der erste Schritt beim Compilieren ist die Bearbeitung des Quelltextes durch den Präprozessor. Dieser entfernt den Kommentar und führt Ersetzungen und Einfügungen gemäß der folgenden Anweisungen (directives) aus:

- #define buchstäbliche Ersetzung einer symbolischen Konstanten oder eines Makros. Ist kein Ersatz angegeben, wird nur der Name als definiert angesehen (für #ifdef). Häufig.
- #undefine löscht die Definition eines Namens.
- #error führt zu einer Fehlermeldung des Präprozessors.
- #include zieht das angegebene File herein. Häufig.
- #if, #else, #elif, #endif falls Bedingung zutrifft, werden die nachfolgenden Präprozessor-Anweisungen ausgeführt.
- #ifdef, #ifndef falls der angegebene Name definiert bzw. nicht definiert ist, werden die nachfolgenden Präprozessor-Anweisungen ausgeführt.
- #line führt bei Fehlermeldungen zu einem Sprung auf die angegebenen Zeilennummer.
- #pragma veranlaßt den Präprozessor zu einer systemabhängigen Handlung.

E 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. Folgende Kennungen (a = ASCII-Text, b = binäres File, ? = wechselnd oder unbekannt) sind verbreitet:

$$$	?	Temporäres File
a	b	UNIX-Archiv, mit `ar(1)` erzeugt
adf	?	Adapter Description File (IBM)
adi	b	AutoCAD DXF
adn	?	Add In Utility (Lotus)
afm	?	Adobe Font Metrics (Adobe)
ani	b	Atari ST Graphics Format
ans	b	ANSI-Grafik
app	?	Application (RBase, NeXT)
arc	?	Archiv, mit `arc` oder `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
avi	b	Microsoft RIFF
avs	b	Intel DVI
bak	a	Backup-File
bas	b	BASIC-File
bat	a	Batchfile unter MS-DOS, entsprechend Shellscript
bbl	a	BIBTeX Literaturverzeichnis
bfx	b	Bitfax-File
bgi	b	Borland Graphic Interface
bib	?	Bibliographie
bif	b	Binary Image File
bin	b	binäres File
bit	b	TeX Ausgabe des Druckertreibers
bk	a	Backup-File (WordPerfect)
bld	b	BASIC Bload Graphics
bmp	b	Microsoft Bitmap Graphics (Windows, OS/2)
bnd	b	Microsoft RIFF
bpx	b	Lumena Paint
bsc	?	Boyan Script-File
bsv	b	BASIC Bsave Graphics
btm	a	Batchfile unter 4DOS
bw	b	SGI Image File Format
c	a	C-Quelle

C	a	C++-Quelle
cal	b	CAL Raster
cap	a	Capture-File (Telix)
cat	a	Catalogue, Verzeichnis
cbl	a	COBOL-Quelle
cc	a	C++-Quelle
cco	b	Btx-Grafik
cdf	?	Comma delimited format
cdr	b	Corel Draw Vektorgrafik
cdx	?	Compund index (FoxPro)
cfg	?	Konfigurations-File
cgm	b	Computer Graphics Metafile (Harvard Graphics, Lotus)
chk	?	von chkdsk erzeugtes File (MS-DOS)
cht	b	Harvard Graphics
clp	?	Clipboard (Windows), Clip Art, GRASP
cmd	a	Command, Skript, Batchfile unter OS/2
cmi	b	Intel DVI
cnc	a	CNC-Programme
cnf	?	Konfigurations-File
cob	a	COBOL-Quelle
cod	?	Codeliste
com	b	Kommando-File (MS-DOS), ausführbares Programm
cpi	?	Code Page Information (MS-DOS)
cpp	a	C++-Quelle
cpt	?	Compact Pro (Kompressor Macintosh)
crd	?	Cardfile
crf	?	Cross Reference File
csv	a	Comma Separated Values
ctx	b	Signaturfile (PGP)
cut	b	Dr. Halo Bitmap
cvf	b	Compressed Volume File
cxx	a	C++-Quelle
dat	a	Daten
db	b	Datenbank (Paradox)
dbf	?	Datenbank (dBase)
dcf	?	Driver Configuration file
dct	?	Dictionary
dcx	b	Fax-File
dd	?	Disk Doubler
def	?	Definitionen, Defaultwerte
dem	b	Demonstration
des	?	Description
dhp	b	Dr. Halo PIC
dib	b	Microsoft Windows Bitmap, OS/2 Bitmap
dic	?	Dictionary
dif	?	Lotus Data Interchange Format
dir	a	Directory, Verzeichnis
dll	b	Dynamically Linked Library (OS/2, Windows)
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)
ega	b	EGA-Grafik
el	a	Emacs Lisp
elc	b	Emacs Lisp compiled
eml	a	Electronic Mail
enc	b	encoded
eps	b	Encapsulated Postscript
err	a	Error, Fehlerprotokoll
etx	?	Setext File
exe	b	executable, ausführbares Programm (MS-DOS)
f	a	FORTRAN-Quelle
fax	b	Fax-File
fif	?	Fractal Image Format
fli	b	EmTeX Fontlibrary
fnt	b	Fontfile
for	a	FORTRAN-Quelle
fxs	b	Fax-File (Winfax)
gem	?	GEM Metadateien
gfb	b	Gifblast compressed GIF image
gif	b	Graphics Interchange File
glo	a	TeX Glossar
gz	b	mit GNU `gzip(1)` komprimiert
h	a	Header-File , include-File
hex	b	Hexdump
hdf	?	Hierarchical Data Format
hpp	a	Header-File C++
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/Bitmap
inf	a	Information
ini	?	Konfigurationsfile (Initialisierung)
jfi	b	JPEG File Interchange Format
jif	b	JPEG Interchange Format
jpeg	b	JPEG File
l	a	`lex(1)`-Quelle
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)
lst	a	Liste
lzh	b	Archiv, mit `lha` komprimiert
m4	a	`m4(1)`-Präprozessor-File
mac	b	Macintosh Paint
map	b	Map-Files (Quick C)
mak	a	Make-File
mdf	?	Menu Definition File
mid	b	Midi Sound Ffile
mif	b	Maker Interchange Format (FrameMaker)
mod	b	MODULA-Quelle, Amiga-Modul (Sound)
msp	b	Microsoft Windows Paint
nff	?	Neutral File Format
o	b	Objekt-Code
obj	b	Objekt-Code
old	?	Kopie eines Files vor Überschreiben
ovl	b	Overlay-File
ovr	b	Overlay-File
p	a	PASCAL-Quelle
pak	b	Archiv, mit `pkpak` komprimiert
pcd	b	Kodak Photo Compact Disc
pct	b	Macintosh PICT
pcx	b	Paintbrush, Rastergrafik
pic	b	Lotus Picture File
pif	b	Program Information File
pit	b	PackIt file (Kompressor Macintosh)
pix	b	Inset PIX, Lumena Paint
pov	b	Persistance of Vision
prn	a	Druckerfile
prt	b	Parallel Ray Trace
ps	b	Postscript-File
psf	b	Permanent Swap File
qfx	b	Fax-File (Quicklink)
qtm	b	QuickTime
r	a	Rational-FORTRAN-Quelle
ras	b	SUN/CALS Raster Grafik
rdi	b	Microsoft RIFF
rff	b	Dore Raster File Format
rla	b	Wavefront Run Length Encoded Version A
rlb	b	Wavefront Run Length Encoded Version B
rtf	?	Rich Text Format
s	a	Assembler-Quelle
sdf	?	Space Delimited File
sea	?	Self Extracting Archive (Macintosh)
sgi	b	SGI Image File Format
sh	a	Bourne-Shell-Script
shar	a	Bourne-Shell-Archiv
shr	a	Shell-Archiv
sit	b	Macintosh StuffIt Archive

src	a	Program Source Code
stf	?	Structured File
sys	b	Treiber unter MS-DOS
tar	b	`tar(1)`-Archiv
tdf	?	Trace/Typeface Definition File
tex	a	TeX- oder LaTeX-Quelltext
tfm	b	TeX-Font-Metrics
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
tz	b	`tar.Z`, tar-Archiv, `compress(1)`-komprimiert
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	Microsoft Windows Sound File (RIFF)
web	a	WEB-Quelle
wfx	b	Fax-File (Winfax)
wmf	b	Windows Metafile Format
wpg	a	WordPerfect Graphics Metafile
wri	a	Windows Textfile
wps	a	MS Works Textfile
x	b	SuperDisk self-extracting archive
xwd	b	X Window Dump
y	a	`yacc(1)`-Quelle
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.

F 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 3000 aus dem Leben gegriffenen Abkürzungen aus den Bereichen Informatik und Telekommunikation. Zwischen Groß- und Kleinschreibung wird nicht immer unterschieden. Einige Abkürzungen sind geschützte Namen; diese sind *nicht* gekennzeichnet. Die Liste beschreibt nur den Gebrauch, sie legt nicht eine bestimmte Definition fest. Eine andere Sammlung findet sich im Netz unter `http://www.chemie.fu-berlin.de/cgi-bin/acronym`. Auch in der Newsgruppe `de.etc.lists` findet man gelegentlich eine Liste.

7E1	(Zeichenformat: 7 Bits/Zeichen, Parity even, 1 Stopbit)
8N1	(Zeichenformat: 8 Bits/Zeichen, no parity bit, 1 Stopbit)
AA	Auto Answer
AAAI	American Association of Artificial Intelligence
AAE	Allgemeine Anschalt-Erlaubnis
AAL	ATM Adaptation Layer
AAMOF	As A Matter Of Fact (Slang)
AAP	Applications Access Point, Association of American Publishers
AAPI	Audio Applications Programming Interface
AARP	AppleTalk Address Resolution Protocol
AASP	ASCII Asynchronous Support Package
AAT	Average Access Time
ABB	ASEA Brown Boveri (Schweiz)
ABCS	Advanced Business Communications via Satellite
ABEND	Abnormal End
ABI	Application Binary Interface
ABIC	Access.bus Industry Group
ABK	Arbeitskreis Bildkommunikation
ABM	Asynchronous Balanced Mode
ABN	Australian Bibliographic Network
ABR	Available Bit Rate
ABT	Abort
AC	Accumulator, Alternating Current, Access Control
ACA	Application Control Architecture (DEC)
ACAS	Application Control Architecture Services
ACATS	Advisory Committee on Advanced Television Systems
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
ACFC	Address and Control Field Compression
ACIAS	Automated Calibration Internal Analysis System
ACID	Atomicity, Consistency, Isolation, Durability
ACIS	American Committee for Interoperable Systems
ACK	Acknowledge/Acknowledgement
ACL	Access Control List, Application Connectivity Link
ACM	Association for Computing Machinery, USA
ACMS	Application Control Management System
ACONET	Akademisches Computer-Netz, Österrreich
ACP	Automatic Channel Programming, Auxiliary Control Process
ACR	Attenuation to Crosstalk Ratio
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
ACT	Advanced Communication for Training, Applied Computerized Telephony (HP)
ACTS	Advanced Communications Technologies and Services
ACTV	Advanced Compatible Television
ACU	Automatic Calling Unit
A/D	Analog/Digital
ADA	Automatic Data Acquisition
AdaIC	Ada Information Clearinghouse
ADaM	Automatischer Datenmeßplatz
ADB	Apple Desktop Bus
ADC	Adaptive Data Compression, Add with Carry, Analogue to Digital Converter, Application Development Classes
ADCCP	Advanced Data Communication Control Procedure
ADD	Adapter Device Driver
ADE	Application Development Environment
ADF	Address Distribution Facility, Automatic Document Feed
ADI	Agence de l'Informatique, Autodesk Device Interface
ADL	Address Data Latch, Adventure Definition Language
ADLAT	Adaptive Lattice Filter
ADM	Application Development Modul, Asynchronous Disconnected Mode, Add/Drop-Multiplexer
ADMD	Administrative/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
ADSL	Asymmetrical Digital Subscriber Line/Loop
ADSP	AppleTalk Data Stream Protocol
ADSR	Attack, Decay, Sustain, Release
ADT	Abstrakter Datentyp, Application Data Type, Abrechnungsdatenträger

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
AEI	Additional Equipment Interface
AEIOUS	Advanced Enhanced Interface for Open Universal Systems
AEM	Automatic Emulation Management
AEP	AppleTalk Echo Protocol
AES	Application Environment Services
AET	Advanced Educational Technology, Application Entity Title
AF	Auxiliary Carry Flag
AFAIK	As Far As I Know (Slang)
AFC	Automatic Font Change, Automatic Frequency Control
AFD	Automatic File Distribution
AFII	Association for Font Information Interchange
AFJ	April Fool's Joke (Slang)
AFK	Away From Keyboard (Slang)
AFNOR	Association Française de Normalisation
AFP	Advanced Function Printing, AppleTalk Filing Protocol
AFS	Alex/Andrew/Advanced File System, Advanced Freephone Service
AFSK	Audio Frequency Shift Keyer/Keying (Modulation)
aFTP	Anonymous FTP (Internet)
AFUU	Association Française des Utilisateurs d'UNIX
AFUUSO	Association Française des Utilisateurs d'UNIX et des Systemes Ouverts
AGA	Advanced Graphics Adapter
AGC	Automatic Gain Control
AGCS	Advanced Graphic Chip Set
AGDX	Arbeitsgemeinschaft DX e. V., Erlangen
AGE	Automatische Gebührenerfassung
AGFMB	Arbeitsgemeinschaft Freier Mailboxen
AGFPL	Aladdin Ghostscript Free Punlic License
AGP	Advanced Graphics Package (HP)
AGUI	Advanced Graphical User Interface
AGV	Automatic Guided Vehicle
AHDL	Altera Hardware Description Language
AI	Artificial Intelligence, Analogue Input, Adobe Illustrator, Active Interface
AIA	Application Integrated Architecture (DEC)
AIDS	Automatic Installation and Diagnostic Service
AII	Active Input Interface
AIIM	Association for Information and Image Management
AIM	Automatic Interface Management, Advanced Informatics in Medicine; Apple, Intel, Motorola
AIMB	As I Mentioned Before (Slang)

AIPS	Astronomical Image Processing System
AIS	Alarm Indication Signal
AISB	Association of Imaging Service Bureaus
AISP	Association of Information Systems Professionals
AIW	APPN Implementor's Workshop
AIX	Advanced Interactive Executive (IBM)
AL	Application Layer, Assembly Language, Arbitrated Loop
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
ALWR	Arbeitskreis der Leiter Wissenschaftlicher Rechenzentren
AM	Amplitudenmodulation
AMANDDA	Automated Messaging and Directory Assistance
AMD	Active Matrix Display, Advanced Micro Devices, Inc., USA
AMDS	Amplitude Modulation Data System
AMH	Automated Materials Handling
AMLCD	Active Matrix Liquid Crystal Display
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
ANDF	Architecture Neutral Distribution Format
ANI	Automatic Number Identification (Telefon)
ANL	Automatic Noise Limiter
ANR	Automatic Network Routing
ANS	Access Network system
ANSA	Advances Network Systems Architecture
ANSI	American National Standards Institute
ANTC	Advanced Networking Test Center
ANW	International Academic Networking Workshop
AO	Analogue Output
AOCE	Apple Open Collaboration Environment
AOE	Application Operating Environment (AT&T)
AOI	Active Output Interface
AOL	America Online, Inc.
AOR	Atlantic Ocean Region
AOS	Add Or Subtract
AOW	Asia and Oceania Workshop (OSI)
AP	Application/Adjunct Processor, Automatic Pagination
APA	All Points Addressable, Adaptive Packet Assembly, Arithmetic Processing Accelerator, Amateur Press Association
APAR	Authorized Program Analysis Report (IBM)

APAREN	Address Parity Enable
APaRT	Automated Packet Recognition/Translation
APC	Application Protocol, Association for Progressive Communications, Arbeitsplatzcomputer, Airborne Public Correspondence, Automated Production Protocol
APCUG	Association of PC Users Groups
APDA	Apple Programmers and Developers Association
APEX	Advanced Packet Exchange
APG	Automated Password Generator
API	Application Program/Programming Interface
APIA	Application Program Interface Association
APIC	Advanced Programmable Interrupt Controller
APL	A Programming Language, Application Programm Library
APM	Advanced Power Management
APOL	Alternate Person On Line (Slang)
APP	Application Portability Profile
APPC	Advanced Program to Program Communication (IBM)
APPI	Advanced Peer-to-Peer Internetworking
APPN	Advanced Peer-to-Peer Network/Networking (IBM)
APS	Advanced Production System, Automatic Position Select, Asynchronous Protocol Specification, Auto Pilot System
APSE	Ada Project Support Environment
APT	Address Pass Through, Automatically Programmed Tools, Application Program title
ARA	AppleTalk Remote Access
ARAP	Apple Remote Access Protocol
ARCA	Advanced RISC Computing Architecture
ARCNET	Attached Resource Computer/Computing Network
ARDF	Amateur Radio Direction Finding
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
ARMS	Architecture for Reliable Managed Storage
ARNS	AppleTalk Remote Network Server
ARP	Address Resolution Protocol (Internet)
ARPA	Advanced Research Projects Agency, USA; Auto Radar Plotting Aid
ARPL	Adjust Requested Privilege Level
ARQ	Automatic Request, Automatic Repeat on Request
ARRL	American Radio Relay League
ARS	Adaptive Rate System, Address Retrieval System
ARTIC	A Real-Time Interface Coprocessor (IBM)
ARU	Audio Response Unit
AS	Autonomous System, Authentication Server
AS3AP	ANSI SQL Standard Scalable and Portable
ASA	American Standards Association
ASAM	Arbeitskreis zur Standardisierung von Automatisierungs- und

	Meßsystemen
ASAP	Automatic Switching And Processing, As Soon As Possible (Slang)
ASC	Accredited Standards Committee (ANSI), Authorized Service Center
ASCII	American Standard Code for Information Interchange
ASEA	Allmänna Svenska Electricitets-Aktiebolag, Schweden
ASI	Adapter Support Interface (IBM)
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, Automated Storage Management
ASN	Access Stack Node
ASN.1	Abstract Syntax Notation One (ISO, CCITT)
ASO	Active sideband Optimum
ASP	AppleTalk Session Protocol, Association of Shareware Professionals, Advanced Signal Processor
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
ATAPI	AT Attachment Packet Interface
ATC	Address Translation Cache
ATDM	Asynchronous Time Division Multiplexing
ATDMA	Asynchronous Time Division Multiple Access
ATE	Automated/Automatic Testing Equipment
ATF	Automatic Track Finding
ATFSM	Ask The Friendly System Manager (Slang)
ATIS	A Tools Integration Standard
ATK	Andrew Toolkit
ATM	Asynchronous Transfer Mode, Adobe Typeface Manager
ATMARP	ATM Address Resolution Protocol
ATM-CC	ATM Crossconnect
ATP	AppleTalk Transaction Protocol
ATR	Automatic Terminal Recognition
ATS	Apple Terminal Services, Administrative Terminal System
ATST	At The Same Time (Slang)
AT&T	American Telephone and Telegraph
ATT	At This Time (Slang)
ATTC	Advanced Television Test Center, USA
ATV	Advanced/Amateur Television
AU	Administrative/Arithmetic/Access Unit
AUDIT	Automated/Automatic Data Input Terminal

AUE	Andrew User Environment
AUGE	Apple User Group Europe
AUI	Attachment Unit Interface (Ethernet)
AUP	Acceptable/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), Audiovisuelle Integration
AVIA	Architecture for Virtual Interfaces for Applications
AVK	Audio-Video Kernel
AVL	Adelson-Velskij, Landis (-Baum)
AVON	Amtliches Verzeichnis der Ortsnetze
AVSS	Audio-Video Subsystem Software
AWADO	Automatischer Wechselschalter in der Anschlußdose
AWD	Automatische Wähleinrichtung für Datenverbindungen
AWG	American Wire Gauge
AWL	Anweisungsliste (DIN 19 239)
AWS	Apple Workgroup Server
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, Block Availability Map
BAPT	Bundesamt Post und Telekommunikation, Mainz
BARRnet	Bay Area Regional Research Network
BARTS	Bell Atlantic Regional Timesharing
BAS	Bild-Austast-Synchron(-Signal), Btx Anwendungsschnittstelle
BASIC	Beginner's All-Purpose Symbolic Instruction Code
BASM	Built-In Assembler
BATC	Block Address Translation Cache
BAZ	Belegauswertung von Zahlungsvorgängen
BB	beschränkt balanciert (Baum)
BBC	Britisch Broadcasting Company
BBN	Bolt, Beranek and Newman, Cambridge, Mass. (Internet)
BBNS	Broadband Network Services
BBS	Bulletin Board System
BC	Begin Chain
BCC	Block Check Character
BCD	Binary Coded Decimal
BCE	Bell Canada Enterprises
BCH	Bose-Chaudhuri-Hocquenghem (-Code)

BCI	Broadcast Interference
:w 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
BHT	Branch History Table
BI	Binary Input
B-ICI	Broadband Inter Carrier Interface
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
BIP	Bit-Interleaved Parity
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
BLADE	Basic Linear Algebra for Distributed Environments
BLAS	Basic Linear Algebra Subprograms/Subroutines
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, Bayonet Nut Coupling, Baby N Connector
BNF	Backus-Naur Form, Big Name Fan (Slang)
BNFSCD	But Now For Something Completely Different (Slang)
BNPF	Begin Negative Positive Finish
BNS	Broadband Network Services
BNU	Basic Networking Utilities (uucp)
BO	Binary Output
BOA	Basic Object Adapter
BOART	Bus Oriented Asynchronous Receiver Transmitter
BOC	Bell Operating Company
BOF	Board of Fellows, Birds Of a Feather (Slang)
BOFS	Birds of a Feather Session
BOM	Begin Of Message
BONT	Broadband Optical Network Termination
BOOM	Binocular Omni-Oriented Monitor
BOOTP	Bootstrap Protokoll (Internet)
BOP	Bit Oriented Protocol
BOS	Basic Operating System
BOT	Begin Of Tape/Table
BP	Bandpass, Base Pointer
BPB	BIOS Parameter Block
bpi	bits per inch
BPM	Bundespostministerium
BPR	Business Process Reengineering
bps	bits/bytes per second
BPSK	Binary Phase Shift Keying
BPU	Branch Prediction Unit
BQS	Berkeley Quality Software
BR	Bad Register
BRAM	Broadcast Recognizing Access Method
BRE	Basic Regular Expression

B-Rep	Boundary Representation
BRI	Basic/Basis Rate Interface, Brain Response Interface
BRIM	Bridge Router Interface Module
BS	Backspace, Base Station, Betriebssystem, Bitonic Sort (Graph)
BSA	Business Software Alliance
BSAM	Basic Sequential Access Method
BSC	Base Station Controller, 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, Board of Standards Review (ANSI)
BSS	Broadband Switching System
BSY	Busy
BSYNC	Binary Synchronous Communications (Protocol)
BT	Bit Test, British Telecommunications
BTAC	Branch Target Address Cache
BTAM	Basic Telecommunication Access Method (IBM)
BTB	Branch Target Buffer
BTC	Bit Test and Complement, Biting The Carpet (Slang)
BTI	Britisch Telecom International
BTIC	But Then, I'am Crazy (Slang)
BTLB	Block TLB
BTMA	Busy Tone Multiple Access
BTP	Batch Transfer Program
BTR	Bit Test and Reset
BTS	Bit Test and Set, Base Transceiver System
BTU	Basic Transmission Unit
BTW	By The Way (Slang)
Btx	Bildschirmtext
BUAF	Big Ugly ASCII Font
BUAG	Big Ugly ASCII Graphic
BUF	Buffer
BUS	Broadcast and Unknown Server
BUUG	Belgian UNIX Users Group
BVIT	Bundesverband Informationstechnologien, Bonn
BWM	Block Write Mode
BWQ	Buzz Word Quotient
BWSN	Baden-Württembergisches Schulnetz
BZR	Befehlzählregister
BZT	Bundesamt für Zulassungen in der Telekommunikation
BZV	Belegorientierter Zahlungsverkehr, Befristeter Zählervergleich

CA	Common Applications, Collision Avoidance, Computer Animation, Computer Associates, Certification Authority,

	Conditional Access
CAA	Computer Aided Assembling/Anthropometrics
CAAD	Computer Aided Architectural Design
CAC	Computer Aided Consulting/Computing/Creativity/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, Computer Aided Education
CAF	Computer Aided Fishing, Computer and Academic Freedom
CAG	Column Adress Generator
CAH	Computer Aided Holidays/Homework/Hausaufgaben
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
CALIES	Computer Aided Locomotion by Implanted Electrical Stimulation
CALL	Computer Aided Language 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, Computing for Analysis Project
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
CARE	Computer Aided Reverse Engineering
CARLOS	Cancer Registry Lower Saxony
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
CAST	Computer Aided Software Testing
CAT	Computer Aided Teaching/Testing/Telecommunication, 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, Cable TV
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
CBDS	Connectionless Broadband Data Service
CBEMA	Computer Business Equipment Manufacturers' Association
CBIP	Compressed Binaries IBM PC, Current Book In Progress
CBL	Computer Based Learning
CBR	Constant/Continuous Bit Rate
CBT	Computer Based Training
CBW	Convert Byte to Word
CBX	Computer-controlled Branch Exchange
CC	Cluster Controller, Congestion Control, Crossconnect
CCC	Ceramic Chip Carrier, 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
CC-NUMA	Cache Coherent NUMA
CCP	Console Command Processor/Program
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, Colour Display,

	Change Direction, Committee Document (ISO),
CDA	Compound Document Architecture (DEC), Communications Decency Act
CDATA	Character Data
CDC	Control Data Corporation
CDDI	Copper Distributed Data Interface
CDE	Common Desktop Environment, Cooperative Development Environment, Complex Data Entry
CDF	Common Data File
CDI	Compact Disc Interactive
CDIF	Case Data Interchange Format
CDL	Computer Design Language
CDM	Compressed Data Mode, Code Division Multiplex
CDMA	Code Division Multiple Access
CDOS	Concurrent Disk Operating System
CDPD	Cellular Digital Packet Data
CDR	Call Detail Record/Recording
CD-R	Compact Disk Recordable
CDRA	Character Data Representation Architecture
CDRAM	Cached DRAM
CD-RDx	Compact Disk Read Only Memory Data Exchange Standard
CDRL	Contract Data Requirements List
CD-ROM	Compact Disk Read Only Memory
CD-ROMXA	CD-ROM Extended Architecture
CDS	Cell Directory Service (OSF), Current Directory Structure
CDTV	Compact Disk Television, Commodore Dynamic Total Vision
CD-V	Compact Disk Video
CD-WO	Compact Disk Write Once
CE	Clear Entry, Cache Enable, Collision Elimination
CeBIT	Centrum für Büro, Information und Telekommunikation
CEBus	Consumer Electronics Bus
CECC	Electronic Components Committee
CEDEX	Courrier d'entreprise distribution exceptionelle
CEFIC	EDIFACT-Subset Chemical Industry
CEG	Continuous Edge Graphics
CEI	Conducted Electromagnetic Interference
CELP	Code/Coded Excited Linear Prediction
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	Conseil Europeen pour la Recherche Nucleaire
CERT	Computer Emergency Response Team (DFN)
CES	Consumer Electronics Show, Circuit Emulation Service
CFDP	Coherent File Distribution Protocol (Internet)
CFM	Configuration Management
CFP	Computer, Freedom and Privacy
CFR	Controlled Ferro Resonant (-USV)

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, Common Gateway Interface
CGM	Computer Graphics Metafile
CGPP	Computer Grahics: Principles and Practise (by Fowley)
CGW	Customer Gateway
CHAP	Cryptographic/Challenge-Handshake Authentication Protocol
CHARM	Character Mode
CHCK	Channel Check
CHILL	CCITT High Level Language
CHOP	Channel Operator
CHPC	Center for High Performance Computing
CHRP	Common Hardware Reference Platform
CHS	Cylinders, Heads, Sectors
CI	Command Interpreter, Computer Interconnect
CIAC	Computer Incident Advisory Capability
CIAM	Computerized Integrated and Automatic Manufacturing System
CIB	Computer Integrated Bureaucracy
CIC	Coordination and Information Center (CSNet)
CICS	Customer Information Control System (IBM)
CID	Configuration, Installation, Distribution; Customer Identifier
CIDR	Classless Internet Domain Routing
CIE	Commission Internationale de l'Eclairage (Farbmodell)
CIEDS	Computer Integrated Electron Design Series
CIF	Computer Integrated Facilities, Common Intermediate Format
CIL	Computerintegrierte Logistik
CIM	Computer Integrated Manufacturing, CompuServe Information Manager
CIOCS	Communication Input/Output Control system
CIOS	Communication Institute for Online Scholarship
CIP	Comp.-Investitionsprogr. des Bundes und der Länder, Computer Aided Intuition Guided Programming, Command Interface Port
CIR	Committed Information Rate
CIRC	Circular Reference, Cross Interleaved Reed Solomon Code
CIRSC	Cross Interleave Solomon Code
CIS	CASE Integration Services, Computer Information System, CompuServe Information Service, Contact Image Sensor
CISC	Complex Instruction Set Computer/Code
CISE	Computer and Information Science and Engineering Directorate
CISPR	Comite International Special des Perturbations Radioelectriques
CIT	Company Integrated Telephony, Computer Integrated Telephone, California Institute of Technology
CIX	Compulink Information Exchange, Commercial Internet Exchange Ass. Association, Central Internet Exchange
CKD	Count Key Data
CKSM	Checksum

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
CLOE	Common LISP Operating Environment
CLOO	Reverse Object Oriented Logical Channel
CLOS	Common LISP Object System
CLP	Cell Loss Payload/Priority, Constraint Logic Programming
CLS	Connectionless Server
CLTP	Connectionless Transport Protocol (OSI)
CLTS	Clear Task Switch Flag, Connectionless Transport Service
CLUT	Color Look-up Table
CLV	Constant Linear Velocity
CM	Corrective Maintenance, Communications Manager (IBM), Connection Machine, Configuration Management
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
CME	Center for Media Education
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/Connection 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)
CNAME	Canonical Name
CNASI	Centre for Networked Access to Scholarly Information (Australien)
CNC	Computerized Numerical Control, Complete Network Centr

CNE	Certified NetWare Engineer/Expert (Novell)
CNET	Centre National d'Etudes des Telecommunications
CNG	Calling (tone)
CNI	Centre National d'Informatique, Certified NetWare Instructor (Novell), Coalition for Networked Information, Communications Network International
CNIDR	Clearinghouse for Networked Information Discovery and Retrieval
CNM	Customer Network Management
CNMA	Communications Network for Manufacturing Application
CNN	Cable News Network
CNRI	Corporation for National Research Initiatives
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 Microfilm, Common Object Model
COMAL	Common Algorithmic Language
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/Systems Environment
COSINE	Cooperation for OSI Networking in Europe
CoSN	Consortium for School Networking
COSS	Common Object Services Specification
CoSy	Conferencing System
COTS	Connection Oriented Transport Service, Coin Operated Telephone System
CP	Complete Partitioning, Choke Packet, Continuous Path, Control Point
CPA	Certified Public Accountant, Code Parasite Autopropageable
CPE	Customer Provided/Premises Equipment
CPFSK	Continuous Phase Frequency Shift Keying
CPG	COSINE Policy Group, Clock Pulse Generator
cpi	characters per inch
CPI	Common Programming Interface (IBM), Clocks Per Instruction
CPI-C	Common Programming Interface for Communications (IBM)
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
CPSR	Computer Professionals for Social Responsibilty
CPST	Control Point Spanning Tree
CPT	COSINE Project Management Team, Command Pass Thru
CPU	Central Processing Unit
CQL	Channel Queue Limit
CR	Carriage Return
C/R	Command Response
CRAM	Cyberspatial Reality Advancement Movement
CRC	Cyclic Redundancy Check; Class, Responsibility, Collaboration
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
CRL	Certificate Revocation List
CR/LF	Carriage Return/Line Feed
CROM	Control Read Only Memory
CRS	Cell Relay Service
CRT	Cathode Ray Tube
CRTC	Canadien Radio-Television and Telecommunications Commission
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)
CSDC	Code Segment Descriptor Cache
CSE	Customer Suppoirt Engineer
CSF	Compagnie Sans Fil
CSG	Constructive Solid Geometry
CSI	Common Synchronous Interface
CSID	Calling Station Identifier
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
CSN	Card Select Number
CSNet	Computer and Science Network, USA
CSO	Central Services Organization
CSP	Cross System Product, CompuCom Speed Protocol, Commercial Subroutine Package
CSRG	Computer Systems Research Group (Berkeley)
CSS	Client-Server-System, Centralized Structure Store
CSTA	Computer Supported Telephony Applications

CSTS	Computer Supported Telephony Standard
CSU	Channel Service Unit
CSV	Comma Separated Variables
CT	Clock Time, Cordless Telephone
CTAN	Comprehensive TeX Archive Network
CTB	Communication Toolbox (Apple)
CTCP	Client To Client Protocol
CTCPEC	Canadian Trusted Computer Product Evaluation Criteria
CTI	Computer Telephony Integration
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 Time-Sharing System
CTY	Console Teletype
CU	Control Unit
CUA	Common User Access (IBM)
CUCN	Communication Controller Node
CUG	Closed User Group
CUI	Character/Common User Interface (IBM)
CUL	See You Later (Slang)
CUNY	City University of New York
CUSI	Configurable Unified Search Index
CUSSNET	Computer Use in Social Sciences Network
CUT	Control Unit Terminal
CUU	Computerunterstützter Unterricht
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 magenta black (Farbmodell)

D/A	Digital/Analog
D2T2	Dye Diffusion Thermal Transfer (Drucker)
DA	Destination Address
AA	Data Access Arrangement, Decimal Adjust for Addition, Digest Access Authentication
DAB	Digital Audio Broadcasting, Daten-Autobahn
DAC	Digital-Analog-Converter, Distributed Academic Computing, Dual Attachment Concentrator, Design Automation Conference, Data Acquisition and Control

DACS	Digital Access and Cross-connect System
DACT	Data Compression Technology
DAF	Destination Address Field
DAG	Direktrufnetzabschlußgerät
DAGM	Deutsche Arbeitsgemeinschaft für Mustererkennung
DAKfCBNF	Deutscher Arbeitskreis für CB- und Notfunk
DAL	Disk Access Logout, Data Access Language (Apple)
DAM	Direct Access Method, Data Acquisition and Monitoring
DAN	Domestic/Desk Area Network
DANTE	Delivery of Advanced Network Technologies in Europe
DaNzKo	Datennetzkoordinator
DAP	Data Access Protocol (DEC), Division Advisory Panel (NSF), Document Application Profile
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
DASAT	Datenübertragung via Satellit
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
DAVIC	Digital Audio Video Council
DAVID	Digital Audio Video Interactive Decoder
DAWG	Directed Acyclic Word Graph
dB	Dezibel (Maßeinheit)
DB	Data Base/Buffer
DBA	Database Access/Agent
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, Data Center Administration, Defense Communications Agency, USA

DCAF Distributed Console Access Facility (IBM)
DCB Data Control Block, Disk Coprocessor Board (Novell)
DCC Digital Compact Cassette, Data Country Code, Display Combination Code, Data Communication Channel
DCCS Distributed Capability Computing System
DCD Data Carrier Detect
DCE Data Circuit/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
DCI Display Control Interface
DCL Digital Command Language (DEC), Device Clear, Declaration
DCM Data Communication Multiplexer, Distributed Computing Model
DCNA Data Communication Network Architecture
DCS Desktop Colour Separation, Diagnostic Communication System, Data Collection/Control System, Digital Cellular Telecommunication System
DCT Discrete Cosine Transformation
DD Double Density, Digital Display
DDA Distributed Database Architecture, Deutsche Daten-Autobahn
DDB Device Dependent Bitmap
DDBMS Distributed Data Base Management System
DDC Display Data Channel
DDCMP Digital Data Communication Message Protocol (DEC)
DDCS Distributed Database Connection Services (IBM)
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 Manager/Management
DDN Defense Data Network, USA
DDNS Distributed Domain Naming Service
DDP Distributed Data Processing
DDS Digital Data Service/Storage, Disk Definition Structure, Direct Digital Service, Dataphone Digital Service (AT&T), Direkte digitale (Frequenz-)Synthese
DDTA Digital Distributed Transaction Architecture
DDV Datendirekt(ruf)verbindung, Dialog Data Validation
DDVT Dynamic Dispatch Virtual Tables
DDX-P Digital Data Exchange, Packet-switching Network
DE Discard Eligible
DEARN Deutsches EARN
DEBI DMA Extended Bus Interface
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
DELUG	Deutsche LINUX User Group
DEN	Document Enabled Networking
DE-NIC	Deutsches Network Information Center
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
DFV	Datenfernverarbeitung
DG	Datagram
DGIS	Direct Graphics Interface Standard
DGL	Device-independent Graphics Library (HP)
DGT	Direction Generale des Telecommunications
DHCP	Dynamic Host Configuration Protocol (Internet)
DHSD	Duplex High Speed Data
DI	Data In, Destination Index, Decoder Identification
DIA	Document Interchange Architecture (IBM), Display Industry Association
DIB	Device Independent Bitmap
DID	Direct Inward Dialing, Digital Image Design
DIF	Data Interchange Format
DIGI	Deutsche Interessengemeinschaft Internet e. V.
DII	Dynamic Invocation Interface
DIIP	Direct Interrupt Identification Port
DIL	Dual in line
DIMDI	Deutsches Inst. für Medizinische Information und Ddokumentation

DIMM	Dual Inline Memory Module
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)
Disco	Digital Secure Communication (Bosch/Ascom)
DI-SCSI	Differential SCSI
DISP	Displacement
DIV	Digitale Vermittlungsstelle
DI-W-SCSI	Differential Wide SCSI (16 Bit)
DIX	DEC, Intel, Xerox
DKE	Deutscher Kleinempfänger (Goebbelsharfe), Deutsche Elektrotechnische Kommission im DIN und VDE
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)
DLCI	Data Link Connection Identifier
DLCX	Data Link Control Exchange
DLE	Data Link Escape
DLL	Data Link Layer, Dynamic Link Library
DLM	Dynamic Link Module, Distributed LAN Monitoring
DLP	Digital Light Processing
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, Document Management Alliance
DMAC	DMA Controller
DMC	Distributed Management Chain
DMD	Device Manager Driver, Directory Management Domain, Digital Mirror Device
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

DMUX	Demultiplexer
DMY	Day Month Year (Datumsangabe)
DNA	Digital Network Architecture (DEC), Document Enabled Network
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 Server/Service/Scheme/System (Internet)
DO	Data Out, Distributed Object
DOAG	Deutsche Oracle-Anwender-Gruppe
DoD	Department of Defence, USA
DOD	Digital Optical Disc
DOE	Distributed Objects Everywhere
DOL	Direct Object Linking
DOMF	Distributed Object Management Facility (Sun)
DOS	Disk Operating System, Distributed Operating System
DOSEM	DOS Emulator/Emulation
DOSS	Dedicated Office Systems and Services
DOT	Documents to Think with (GMD)
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 Signaling/Standard
DPO	Data Phase Optimization
DPS	Display Postscript, Digital Print System
DPSK	Digital Phase Shift Keying
DPT	Drive Parameter Table
DPU	Data Parallel Unit
DQCB	Distributed Queue Control Bus
DQDB	Distributed Queue Double/Dual Bus
DQL	DataEase Query Language
DQPSK	Differential Quadrature Phase Shift Keying
DR	Data Received, Digital Research

DRAM	Dynamic Random Access Memory
DRAW	Direct Read After Write
DRCS	Dynamically Redefinable Character Set
DRDA	Distributed Relational Database Architecture (IBM), Distributed Remote Database Access
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	Double Sided, Drive Select, Distributed Services (OSF), Data Segment/Send
DS-3	Digital Signal Level 3
DSA	Data/Digital Storage Architecture, Directory System Agent (X.500), Data Service Adapter, Digital Signature Algorithm
DSAB	Distributed Systems Architecture Board
DSAF	Destination Subarea Address Field
DSAS	Data Sharing Architecture System
DSB	Datenschutzbeauftragter
DSC	Document Structuring Conventions (Postscript)
DSCS	Defense Satellite Communications System
DSDC	Data Segment Descriptor Cache
DSDD	Double Sided Double Density (Floppy)
DSE	Data Switching Exchange, Data Storage Equipment
DSEA	Display Station Emulation Adapter
DSECT	Data/Dummy Control Section
DSH	Desparately Seeking Help (Slang)
DSHD	Double Sided High Density (Floppy)
DSI	Digital Sspeech Interpolation
D-SIMM	Dual RAS SIMM
DSIS	Distributed Support Information Standard
DSL	Dialogue Scripting Language (OSF), Dynamic Simulation Language
DSLB	Digital Subscriber Line Board
DSMIT	Distributed SMIT (IBM)
DSN	Deep Space Network
DSOM	Distributed System Object Model
DSP	Digital Signal Processing/Processor
DSPU	Downstream Physical Unit
DSQD	Double Sided Quad Density (Floppy)
DSR	Data Set Ready, Device Status Register/Report, Digitales Satelliten-Radio
DSS	Decision Support System, Digital Satellite System, Digital Signature Standard, Digital Subscriber Signalling
DSSI	Digital Storage Systems Interconnect
DSSS	Direct Sequence Spread Spectrum
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	Dial/Dual Tone Multi-Frequency
DTP	Desktop Publishing
DTPS	Dialer Token Pairs
DTR	Data Terminal Ready, Data Transfer Rate
DTS	Distributed Time Server, Direct To SOM
DTTB	Digital Terrestrial TV Broadcasting
DTX	Discontinuous Transmission
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
DVC	Digital Video Cassette
DVCR	Digital Video Cassette Recording
DVD	Deutsche Vereinigung für Datenschutz
D-VHS	Data VHS
DVI	Digital Video Interactive/Interface, Device Independent, Deutsches Video Institut
DVL	Digital Video Link
DVST	Datenvermittlungsstelle
DWIM	Do What I Mean (Slang)
DWM	Diskless Workstation Managent
DWT	Discrete Wavelet Transformation
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 Address/Attribute
EAC	European Activities Committee
EAN	European Article Number, Extragalactic Area Network, European Advanced Networking (Protocol)
EANCOM	EDIFACT-Subset Konsumwirtschaft
EANTC	European Advanced Networking Test Center

EARN	European Academic Research Network
EAROM	Electrically Alterable Read Only Memory
EARS	Electronic Access to Reference Services
EASA	European Academic Software Award
EATA	Enhanced AT Bus Attachment
EAZ	Endgeräteauswahlziffer (ISDN)
EBC	EISA Bus Controller
EBCDIC	Extended Binary Coded Decimal Interchange Code (IBM)
EBI	Equivalent/Extended Background Input/Investigation
EBNF	Erweiterte (Extended) Backus-Naur-Form
EBR	Excessive Burst Rate
EBT	Electronic Benefits Transfer
EBU	European Broadcasting Union
EBV	Elektronische Bildverarbeitung
EC	End Chain, Exchange Carrier, European Community
ECAS	Exchange Card Architecture Specification
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
ECO	Electron Coupled Oscillator
ECODU	European Control Data Users Group
ECOOP	European Conference on Object Oriented Programming
ECP	Enhanced Capabilities Port (Microsoft)
ECRC	European Computer Industry Research Centre
ECS	European Common Standard
ECSI	European Custom Systems Integration
ECU	EISA Configuration Utility
ED	Erase Display
EDA	Embedded Document Architecture, Electronic Design Automation, Explorative Datenanalyse
EDAC	Error Detection And Correction
EDC	Error Detection Code, Enhanced Data Correction
EDDC	Extended Distance Data Cable
EDFA	?
EDI	Electronic Data/Document Interchange
EDIBDB	EDIFACT-Subset Baumärkte
EDIFACT	Electronic Data Interchange for Administration, Commerce and Transport (UN, DIN)
EDIFICE	EDIFACT-Subset Elektroindustrie
EDIFURN	EDIFACT-Subset Möbelindustrie
EDIOFFICE	EDIFACT-Subset Bürobedarf

EDITEX	EDIFACT-Subset Textilindustrie
EDL	Encapsulator Description Language
EDLC	Ethernet Data Link Control
EDM	Engineering Data Management
EDO	Enhanced/Extended Data Output
EDP	Electronic Data Processing
EDR	External Developer Relaease
EDRAM	Enhanced DRAM
EDS	Elektronisches Datenvermittlungssystem
EDSI	Enhanced Small Device Interface
EDSRA	Earth Data System Reference Application
EDSS	European Digital Subscriber System
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
EES	Escrowed Encryption Standard
EFA	Extended File Attribute
EFAC	Extended File Access Control
EFF	Electronic Frontier Foundation
EFI	Electromechanical Frequency Interference
EFISC	Enhanced Fast Instruction Set Computer
EFL	Emitter Follower Logic
EFM	Eight-to-Fourteen Modulation
EFT	Euro File Transfer, Electronic Funds Transfer
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)
EGPE	Extended Generalized Programming Environment
EGW	Edge Gateway
EHKP	Einheitliches höheres Kommunikationsprotokoll
EIA	Electronic Industries Association, USA
EIAJ	Electronics Industry Association of Japan
EIB	European Installation Bus
EIBIEM	Enhanced Interactive Business Integrating Environment Manager
EIDE	Enhanced IDE
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, Enterprise/Executive Information System
EISA	Extended Industry Standard Architecture
EISS	Europäisches Institut für Systemsicherheit
EITI	European Interconnect Technology Initiative
EITO	European Information Technology Observatory
EIUF	European ISDN User Forum
EKT	Envelope-Kanal-Teiler
EL	Erase Line, Electroluminiscent (Display), Elevation
ELAN	Educational/Elementary Language, Emulated LAN
ELANC	Enhanced LAN Controller
ELF	Extremely Low Frequency, Executable and Linking Format
ELFEXT	Equal Level Far End Crosstalk
ELOD	Erasable Laser Optical Disk
ELS	Entry Level System
EM	End of Medium, Electronic Mail, Expanded Memory
EMA	Enterprise Management Architecture
EMAC	European Market Awareness and Education Committee
EMACS	Editing Macros (GNU)
EMAIL	Electronic Mail
EMB	Extended Memory block, Enhanced Master Burst
EMC	Electromagnetic Compatibility, Extended Math Coprocessor
EME	Earth - Moon - Earth (Connection)
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, Electronic Mail System, Electronic Message Service
EMUG	European MAP Users Group
EMV	Elektromagnetische Verträglichkeit
EMVG	Gesetz über die elektromagnetische Verträglichkeit von Geräten
EMX	Enterprise Mail Exchange
EN	End Node, European Norm
ENA	Electronic Networking Association
ENCMM	Ethernet Network Control + Management Modeul
ENIAC	Electronic Numerical Integrator And Calculator
ENITH	European Network for Information Technology in Human Services
ENL	European Network Laboratories
ENP	Ethernet Node Processor
ENQ	Enquiry
ENS	Enterprise Network System/Services
ENSS	Exterior Nodal Switching Subsystem (Internet)
EOA	End Of Address
EOB	End Of Block
EOC	End of Character/Conversion, Embedded Operations Channel
EOF	End Of File/Flame, Enterprise Objects Framework
EOI	End Or Identify

EOJ	End Of Job
EOL	End Of Line/List, Europa Online
EOM	End Of Message/Meckerei
EON	Enhanced Information Concerning Other Networks
EOR	Exclusive Or (auch: XOR)
EOS	End Of String, Educational Online Sources
EOT	End of Tape/Transmission/Table/Text
EOU	End Of User
EOUG	European Oracle Users Group
EPA	Environmental Protection Agency, USA
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, Electronic Publishing System
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, European Radiocommunication Committee
ERCIM	European Research Consortium for Informatics and Applied Mathematics
ERIC	Educational Resources Information Center, USA
ERLL	Enhanced Run Length Limited
ERM	Entity Relationship Model
ERMES	European Radio Message System, European Messaging System
ERN	Explicit Route Number
EROM	Erasable Read Only Memory
ERP	Effective Radiated Power
ERR	Error
ES	Extra Segment, 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
ESDC	Extra Segment Descriptor Cache
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
ESIG	European SMDS Interest Group

ESM	Ethernet Switching Module
ESMTP	Extended SMTP (Internet)
ESOC	European Space Operation Centre
ESP	Enhanced Serial Port, Encapsulating Security Payload
ESPRIT	European Strategic Programme for Research in Information Technology
ESS	Environmental Stress Screening, Electronic Switching System
ESU	Electrostatic Unit
ESV	Elektronikschrottverordnung
ETANN	Electrically Trainable Analog Neural Network (Intel)
ETB	End of Transmission Block
ETDD	Electrical Time Division Demultiplexer/Demultiplexing
ETDM	Electrical Time Division Multiplexer/Multiplexing
ETE	Entry to Exit
ETI	Extended Terminal Interface
ETOL	Evil Twin On Line (Slang)
ETOX	EPROM Tunnel Oxide
ETR	Early Token Release
ETS	European Telecommunication Standard
ETSA	European Telecommunication Services Association
ETSI	European Telecommunications Standards Institute
ETX	End of Text
EU	Execution Unit
EUC	Enhanced/Extended UNIX Code
EUnet	European UNIX Network
EurOpen	Dachverband europäischer UNIX-Anwendervereinigungen
EUUG	Extragalactic/European Unix User Group
EV	Etagenverteiler, European Videotelephony
EVA	Enhanced Video Adapter, Elektronischer Verkehrslotse für Autofahrer, Elektronische Fahrplan- und Verkehrsauskunft, Empfangs- und Verteilanlage
EVAS	EDV-gestütztes Vereins-Administrations-System
EVE	Exklusiver Verzeichnis-Eintrag (Datex-J)
EVGA	Extended Video Graphics Array/Adapter
EVN	Einzelverbindungsnachweis
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 Subset of APT
EXCH	Exchange
EXFCB	Extended File Control Block
EXP	Exponent
EXT	External
EXTRN	External Reference
EXUG	European X User Group
EZV	Elektronischer Zahlungsverkehr

F2F	Face to Face (Slang)
FA	Full Adder, Fermeldeamt, Fernamt
FAB	(Chip) Fabrication Plant
FABS	Fast Access Btree Structure
FAC	File Access Code
FACT	Federation Automatic Coding Technologies
FAF	File Attribute File
FAG	Fernmeldeanlagengesetz
FAL	File Access Listener
FAMOS	Floating Gate Avalanche MOS
FAN	Family Area Network
FAP	File Access Protocol
FAPI	Family Application Program Interface
FAPL	Format and Protocol Language
FAQ	Frequently Asked Questions; Fragen, Antworten, Quellen der Erleuchtung (Netnews)
FAQL	FAQ List
FARNET	Federation of American Research Networks
FASIC	Function- and Algorithm-Specific Integrated Circuit
FAST	Fast Access Stationary Tape Guide Transport
FAT	File Allocation Table
FAXT	Far End Crosstalk
Fax-TAM	Fax and Telephone Answering Machine
FBAS	Farb-Bild-Austast-Synchronsignal (RGB-Signal)
FBE	Free Buffer Enquiry (Arcnet)
FBRM	Fractional Bit Rate Modulation
FC	Frame Control, Federal Criteria, Fibre Channel
FCA	Fibre Channel Association
FCB	File Control Block
FCC	Federal Communications Commission, USA
FCCSET	Federal Coordination Committee on Science, Engineering and Technology
fci	flux changes per inch
FCP	Fibre Channel Protocol
FCS	Frame Check Sequence, First Customer Shipment
FCSI	Fiber Channel Systems Initiative
FCT	Fast CMOS Technology
FD	Full Duplex, Floppy Disk
FDC	Floppy Disk Controller
FDDI	Fiber Distributed Data Interface
FDDI TP-PMD	FDDI Twisted Pair Physical Layer Medium Dependent
FDDI/UTP	FDDI Unshielded Twisted Pair
FDE	Full Duplex Ethernet
FDES	Full Duplex EtherSwitch
FD-FDDI	Full Duplex FDDI
FDISK	Fixed Disk
FDM	Frequency Division Multiplexing
FDMA	Frequency Division Multiple/Multiplex Access

FDSE	Full Duplex Switched Ethernet
FDX	Full Duplex
FE	Finite Elemente, Format Effektor, Frame Error
FEA	Fast Ethernet Alliance
FEAL	Fast Data Encipherment Algorithm
FEBE	Far End Block Error
FEC	Forward Error Correction
FECN	Forward Explicit Congestion Notification
FEM	Finite-Elemente-Methode
FEP	Front End Processor
FERF	Far End Receive Failure
FET	Feldeffekt-Transistor
FEXT	Far End Crosstalk
FF	Form Feed, Flipflop
FFAPI	File Format API
FFDT	FDDI Full Duplex Technology (DEC)
F!FF	Forum Informatiker für Frieden und gesellschaftliche Verantwortung
FFS	Fast File System, For Free Server, Flexible Fertigungssysteme
FFT	Fast Fourier Transformation
FFTP	Fast Food Transfer Protocol (Mampfnet)
FG	Frame Ground, Feature Group
FGV	Fernmeldegebührenvorschriften
FIAWOL	Fandom Is A Way Of Life (Slang)
FID	Format Identifier/Identification
FIF	Fachinformation in Fachhochschulen, Fractal Image Format
FIFF	Forum InformatikerInnen für den Frieden e. V.
FIFO	First In, First Out (Speicher)
FILO	First In, Last Out (Speicher)
FIMS	Forms Integration Management Standard
FIP	File Processor Buffering
FIPS	Federal Information Processing Standard, USA
FIR	Finite Impulse Response
FIRP	Federal Internetworking Requirements Panel (NIST)
FIRST	Forschungsinst. für Rechnerarchitektur und Softwaretechnik, Berlin, Forum of Incident Response and Security Teams
FISH	First In, Still Here
FIT	Failures In Time
FITB	Fill In The Blank (Slang)
FITNR	Fixed In The Next Release (Slang)
FITS	Flexible Image Transport System
FIZ	Fachinformationszentrum
FKTG	Fernseh- und Kinotechnische Gesellschaft
FLC	Ferro-electric Liquid Crystal
FLOF	Full Level One Feature
FLOP	Floating Point Operation
FM	Frequenzmodulation, Facility/Fault/Function Management
FMC	Flexible Manufacturing Cell
FMH	Function Management Header
FMP	Functional Multiprocessor/processing

FMS	Fernmeldesystem, File/Forms Management System, Flexible Manufacturing System
FNC	Federal Networking Council, USA
FO	Fernmeldeordnung
FOAF	Friend Of A Friend (Slang)
FOC	Fiber Optic Cable
FOIRL	Fiber Optic Inter Repeater Link
FORTRAN	Formula Translator
FORTWIHR	Forschungsverbund für technisch-wissenschaftliches Hochleistungsrechnen
FOSSIL	Fido-Opus-Seadog Standard Interface Layer
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
FPLMTS	Future Public Land Mobile Telecommunications System
FPLS	Field Programmable Logic Sequencer
FPM	Fast Packet Multiplexing, Fast Page Mode
FPML	Field Programmable Macro Logic
FPODA	Fixed Assignment PODA
FPP	Fast Parallel Port, Floating Point Processor, Fixed Path Control
FPS	For Pay Server, Fast Packet Switching, Frames per Second
FPT	Forced Perfect Termination
FPU	Floating Point Unit
FQDN	Fully Qualified Domain Name (Internet)
FR	Frame Relay
FRAD	Frame RElay Assembler/Disassembler
FRAM	Ferroelectric Random Access Memory
FRE	Full Regular Expression
FRICC	Federal Research Internet Coordinating Committee
frpi	flux reversals per inch
FRMR	Frame Reject (frame)
FRS	Frame Relay Service
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
FSD	File System Driver
FSF	Free Software Foundation
FSK	Frequency Shift Keying (Modulation)
FSM	Finite State Machine
FSP	File Service Protocol (Internet)
FSS	Fixed Satellite Services
FTA	Final Type Approval
FTAM	File Transfer, Access and Management (OSI)

FTLS	Formal Top Level Specification
FTP	File Transfer Protocol (Internet), Foiled Twisted Pair
FTPI	Flux Transitions Per Inch
FTTB	Fiber To The Building
FTTC	Fiber To The Curb
FTTH	Fiber To The Home
FTTG	Fiber To The Galaxy
FTZ	Fernmeldetechnisches Zentralamt, Forschungs- und Technologiezentrum der DBP Telekom
FUBS	Fido Used Book Squad (Slang)
FUD	Flußdiagramm (DIN 66 001)
FuE	Forschung und Entwicklung
FUI	File Update Information
FUNET	Finnish University and Research Network
FUNI	Frame Relay User Network Interface
FUP	Funktionsploan (DIN 40 719)
FÜV	Fernmeldeanlagen-Überwachungsverordnung
FVT	Full Video Translation
FVV	Feste virtuelle Verbindung
FWEE	Fernwirkendeinrichtung
FWEG	Fernwirkendgerät
FWI	Frequently Wanted Information
FWIW	For What It's Worth (Slang)
FWLSt	Fernwirkleitstelle
FYA	For Your Amusement (Slang)
FYI	For Your Information (Internet)
FZI	Forschungszentrum Informatik, Karlsruhe

GaAs	Galliumarsenid
GAB	Group Audio Bridging
Gabico	Ganz billiger Computer
GAE	German Application Environment
GAIN	German Advanced Integrated Network (IBM)
GAL	Generic Array Logic
GAMM	Gesellschaft für Angewandte Mathematik und Mechanik
GAN	Global/Galactic Area Network
GAT	Geek of All Trades
GBB	Generic Blackboard System
GBG	Geschlossene Benutzergruppe (Btx, Datex-J)
GC	Garbage Collect
GCAD	Geographical Computer Aided Design System
GCI	General Circuit Interface
GCN	Global Challenge Network
GCR	Group Coded Recording
GCS	Geek of Computer Science
GDA	Global Data Area, Global Directory Agent
GDDM	Graphics Data Display Manager
GDI	Graphical Device Interface
GDMO	Guidelines for the Definition of Managed Objects (OSI)

GDN	Government Data Network (UK)
GDP	Generalized Drawing Primitive
GDS	Global Directory Service
GDT	Global Descriptor Table
GDTRC	Global Descriptor Table Register Cache
GE	Geek of Engineering
GECOS	General Electric Comprehensive Operating System
GEIS	General Electric Information Services
GEM	Graphics Environment Manager, Global Enterprise Management
GEN	Global European Network
GEO	Geostationary Earth Orbit
GFC	Generic Flow Control
GFSK	Gaussian Frequency Shift Keying
GGS	Gütegemeinschaft Software e. V., Köln
GI	Gesellschaft für Informatik
GIF	Graphics Interchange Format
GIGO	Garbage In, Garbage Out
GII	Global Information Infrastructure
GIPS	Giga-Instructions Per Second
GIS	Geographic Information Systems
GIX	Global Internet Exchange
GKS	Graphical Kernel System
GL	Graphics Language
GLASS	Globally Accessible Services
GLU	General Logic Unit
GM	General MIDI
GMD	Gesellschaft für Mathematik und Datenverarbeitung
GMDS	Global Managed Data Services
GME	Gesellschaft Mikroelektronik im VDE
GMHS	Global Message Handling System
GML	Generalized Markup Language
GMS	Global Messaging Service (Novell)
GMSK	Gaussian Minimum Shift Keying (Modulation)
GMT	Greenwich Mean Time
GNA	Globewide Network Academy
GND	Ground
GNI	Gesellschaft für Angewandte Neuroinformatik
GNN	Global Network Navigator
GNSS	Global Navigation Satellite System
GNU	GNU's Not UNIX
GO	Geek of Other
GoS	Grundsätze ordnungsgemäßer Speicherbuchführung
GOSIP	Government OSI Profile
GOT	Global Offset Table
gov	Government; Internet-Domain: Behörden in USA
GP	General Purpose, Gas Plasma
GPC	General Purpose Computer
GPF	General Protection Fault
GPI	Graphics Programming Interface

GPIB	General Purpose Interface Bus
GPL	General Public License (GNU)
GPP	Generic Packetized Protocol
GPS	Global Positioning System, Global Pizza Service
GPU	Graphic Processor Unit
GQL	Graphical Query Language
GRA	Group Random Access
GRE	Graphics Engine
GREAT	Graphical Environment And Desktop
GS	Group Separator, Geek of Science, Geprüfte Sicherheit (TÜV)
GSA	Global Security Architecture
GSL	Graphics Subroutine Library
GSM	Group Speciale Mobile, Global System for Mobile Communications
GSMA	Global Scheduling Multiple Access
GSNW	Gateway Services for Netware
GSX	Graphics System Extension
GTL	Gunning Transceiver Logic
GTO	Gate Turn Off
GU	Geek Undecided
GUD	Grand Unified Debugger
GUI	Graphical User Interface
GULP	Graph Unification Logic Programming
GUMM	Gurus of UNIX Meeting of Minds
GUN	Gemeinschaft der UNIX-Anwender Niederrhein
GUP	Generic Ultimate Protocol
GUUG	German/Galactic Unix User Group
GV	Gebäudeverteiler
GVBD	Götz-von-Berlichingen-Dämon (Internet)
GVPN	Global Virtual Private Network
GVV	Gewählte virtuelle Verbindung
GW	Gateway, Gleichwelle
GWL	Gateway Link
GZS	Gesellschaft für Zahlungssysteme
HA	Halbaddierer, Half Adder
HACMP	High Availability Cluster Multi-Processing (IBM)
HAs	Hauptanschluß
HAL	Hardware Abstraction Layer, House Programmed Array Logic, Hard Array Logic
HASP	Houston Automatic Spooling Priority
HBA	Host Bus Adaptor
HBS	Home Bus System
HCA	Heartbeat Collision Avoidance
HCI	Human Computer Interaction
HCT	Home Communication Terminal
HD	Hard disk, Heavy Duty, High Density
HDA	Head Disk Assembly
HDB	HoneyDanBer (P. Honeyman, D. A. Norwitz, B. E. Redman) (uucp)
HDB3	High Density Bipolar Code of Order 3

HDCD	High Definition Compatible Digital, High Density CD
HDF	Hierarchical Data Format, HDSL Dual Framer
HDI	Head to Disk Interference
HDLC	High Level Data Link Control (Protocol)
HDPLD	High Density Programmable Logic Device
HDR	Header, HDSL Dual Regenerator
HDSC	High Density Signal Carrier
HDSL	High Bit Rate/Speed Digital Subscriber Line/Link
HDTV	High Definition Television
HDW	Hardware
HDX	Half Duplex
HEC	Header Error Control/Check
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, High Functionality
HfD	Hauptanschluß für Direktwahl/Direktruf
HFrG	Gesetz über den Betrieb von Hochfrequenzgeräten
HFS	Hierarchical File System
HFT	High function Terminal
HGA	Hercules Graphics Adapter, High Gain Antenna
HHI	Heinrich-Hertz-Institut, Berlin
HHOK	Ha Ha Only Kidding (Slang)
HHOS	Ha Ha Only Serious (Slang)
HIFD	High Density Floppy Disk
HIFI	Hypertext Interface For Information
HIL	Human Interface Link (HP)
HIPER	High Performance European Radio
HIPPI	High Performance Parallel Interface
HL	Hop Level (Flow Control)
HLCO	High Low Close Open
HLL	High Level Language
HLLAPI	HLL Application Program Interface
HLQ	High Level Qualifier
HLR	Home Location Register
HLRZ	Hochleistungs-Rechenzentrum (GMD)
HLS	Hue, Lightness/Luminance, Saturation (Farbmodell)
HLT	Halt
HMA	Hub Management Architecture, High Memory Area
HMD	Head/Helmet Mounted Display
HMOS	High Density Metal Oxide Semiconductor, High Speed MOS
H-MUX	Hybrid Multiplexer
HOH	HDSL Overhead Bit Handling
HOPTE	High Order Path Terminating Equipment
HP	Hochpass, Hewlett-Packard
HPCC	High Performance Computing and Communications

HPCS	High Performance Communication Server
HPF	High Performance FORTRAN
HP-FL	Hewlett-Packard Fiber Optic Link
HPFS	High Performance File System (OS/2)
HPGL	Hewlett-Packard Graphics Language
HPIB	Hewlett-Packard Interface Bus
HPL	Home Product Link
HPM	Hyper Page Mode
HPOFS	High Perormance Optical File System
HPPA	Hewlett-Packard Precision Architecture
HPPI	High Performance Parallel Interface
HPSL	Hewlett-Packard Support Line
HPSN	High Performance Scalable Networking
HQ	High Quality
HRC	Hybrid Ring Control
HRG	High Resolution Graphics
HRIS	Human Resource Information system
HRMS	Human Resource Management System
HS	High Speed
HSB	Hue, Saturation, Brightness
HSC	High Speed Channel
HSD	High Speed Data
HSDT	High Speed Data Transport
HSI	Hayes Synchronous Interface; Hue, Saturation, Intensity
HSLAN	High Speed Local Area Network
HSL-FX	Hierarchical Specification Language - Function Extension
HSM	Hierarchical Storage Management
HSP	High Speed Printer
HSSI	High Speed Serial Interface
HST	High Speed Technology
HSV	Hue, Saturation, Value (Farbmodell)
HT	Horizontal Tabulator
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol (Internet)
HVC	Hue, Value, Chroma (Farbmodell)
HVP	Horizontal + Vertical Position
HW	Hardware
HWCP	Hardware Code Page
HYTEA	Hypertext Authoring Environment
I	Information (frame)
I4DL	Interface, Inheritance, Implementation, Installation Definition Language
I&A	Identification & Authentication
IAAS	Institut für Angewandte Analysis und Stochastik, Berlin
IAB	Internet Activities/Architecture Board, International Academy of Broadcasting
IAC	In Any Case (Slang), Interapplication Communication (Apple)
IAE	ISDN Anschalteinheit/Anschlußeinheit

IAFA	Internet Anonymous FTP Archive
IAG	Industry Advisory Group (Novell)
IAL	International Algebraic Language (ALGOL)
IAN	Integrated Access Node
IANA	Internet Assigned Numbers Authority
IAR	Institut für Automation und Robotik
IARU	International Amateur Radio Union
IAT	Import Address Table
IAUG	International AIX Users Group
IBA	Independent Broadcasting Authority
IBAC	In Band Adjacent Channel
IBC	Illinois Benedictine College (Gutenberg-Projekt), International Broadcasting Convention
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, Input/Integrated Circuit, Information Center, Interrupt Controller, Interexchange Carrier
ICA	Interface Connector Assembly, Intelligent Communication Adapter
ICALP	International Colloquium on Automata, Languages and Programming
ICAM	Integrated Computer Aided Manufacturing
ICAP	Integrated Computer Aided Production
ICAS	Intel Communicating Applications Specifications
ICCCM	Interclient Communications Conventions Manual
ICCHP	International Conference on Computers for Handicapped Persons
ICCS	Integrated Communications Cabling System (Siemens)
ICE	Inter Client Exchange (Protocol), In-Circuit Emulator/Emulation
ICI	Inter/Interexchange Carrier Interface
ICIP	ICI Protocol
ICL	Interface Clear
ICMP	Internet Control Message Protocol
ICO	Intermediate Circular Orbit
ICOCBW	I Could, Of Course, Be Wrong (Slang)
ICP	Input Control Procedure, Integrated Channel Processor
ICR	Intelligent Character Recognition
ICS	Intuitive Command Structure, IBM Cabling System
ICU	ISA Configuration Utility
ID	Identifier, Identification, Instruction Decode
IDA	Independent/Intelligent Disk/Drive Array
IDAPI	Integrated Database Application Programming Interface
IDC	International Data Corporation
IDE	Integrated Drive Electronics, Interactive Design and Engineering, Interface Design Enhancement
IDEA	International Data Encryption Algorithm
IDF	Intermediate Distribution Frame
IDK	Interessengemeinschaft Datenkommunikation
IDL	Interface Definition Language, Interactive Data Language

IDMS	Integrated Database Management System
IDN	Integriertes Datennetz, Integrated Digital Network
IDP	Internet Datagram Protocol, Integrated Data Processing
IDR	Intermediate Data Rate Service, Intergalactic Digital Research
IDT	Improved Definition Television, Interrupt Descriptor Table
IDTRC	Interrupt Descriptor Table Register Cache
IEC	International Electrotechnical Commission, Interexchange Carrier
IECC	Intercultural E-mail Classroom Connections
IECQ	IEC Quality Assessment System for El. Components
IEEE	Institute of Electrical and Electronics Engineers
IEEE/CS	IEEE Computer Society
IEF	Information Engineering Facility
IEN	Internet Experimental Note
IES	Information Exchange System
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IEX	Imaging Extensions for XWS
IFA	Internationale Funk-Ausstellung
IFC	Integrated Factory Control
IFF	Interchange file format (Amiga)
IFFT	Inverse Fast Fourier Transformation
IFG	Incoming Fax Gateway
IFIP	International Federation of Information Processing Societies
IFL	Internet For Learning
IFS	Internal Field Separator, Installable File System (OS/2)
IFU	Internationale Fernmeldeunion
IFV	Internationaler Fernmeldevertrag
IFVV	Internationale feste virtuelle Verbindungen
IfW	ISDN for Workgroups (Microsoft)
IfWfW	ISDN for Windows for Workgroups (Microsoft)
IGA	Integrated Graphics Adaptor/Array
IGBT	Insulated/Insulator Gate Bipolar Transistor
IGC	Institute for Global/Galactic Communications, Integrated Graphics Controller
IGES	Initial Graphics Exchange Specification
IGMP	Internet Group Message Protocol
IGN	IBM Global Net
IGP	Interior Gatway Protocol (Internet)
IGRP	Internet Gateway Routing Protocol (Cisco)
III	Institute for Information Industry, Taiwan
IIL	Integrated Injection Logic
IIPS	Irrevocably Interruptible Power Supply
IIR	Immediate Impulse Response
IIRC	If I Remenber Correctly (Slang)
IITF	Information Infrastructure Task Force, USA
IJCAI	International Joint Conference on Artificial Intelligence
IK-CAPE	Industriekooperation Computer Aided Process Engineering
IKOE	Institut für Kommunikationsökologie, Hamburg
ILLAB	Ich liege lachend am Boden (Slang, vgl. ROTFL)

ILMI	Interim Local Management Interface
IM	Information Modeling
IMA	Interactive Multimedia Association, Internationale Multimedia Akademie (Karlsruhe)
I-MAC	Isochronous MAC (FDDI)
IMACS	International Association for Mathematics and Computers in Simulation
IMAO	In My Arrogant Opinion (Slang)
IMCO	In My Considered Opinion (Slang)
IMDS	Image Data Stream (IBM)
IMEI	International Mobile Equipment Identifier/Identity
IMG	Image
IMHO	In My Humble Opinion (Slang)
IMNSCO	In My Not So Considered Opinion (Slang)
IMNSHO	In My Not So Humble Opinion (Slang)
IMO	In My Opinion (Slang)
IMOBO	In My Own Biased Opinion (Slang)
IMP	Inter/Interface Message Processor
IMPA	Intelligent Multi-Port Adaptor
IMS	Intermediate Maintenance Standards, Information Mangement System
IMSI	International Mobile Subscriber Identity
IN	Intelligentes Netz (Telekom), Individual Network e. V., Input
INC	Increment, In Chain
INET	International Networking Conference
INH	Intelligent Networking Hub
INKA	Individual Network Region Karlsruhe
INM	Integrated Network Management
INMARSAT	International Maritime Satellite Organization
INN	International Network News
INPO	In No Particular Order (Slang)
INRIA	Institut National de Recherche en Informatique et en Automatique
INS	Input String
INT	Interrupt, Internal, Integer
INTA	Interrupt Acknowledge
INTAP	Interoperability Technology Association for Information Processing, Japan
INWG	Internet Working Group
I/O	Input/Output
IOC	Input Output Controller
IOCS	Input/Output Control System
IOCTL	Input/Output Control
IOFEXT	Input Output Far End Crosstalk
IONL	Internal Organization of the Network Layer (OSI)
IOP	Input Output Processor
IOPL	Input/Output Priviledge Level
IOR	Indian Ocean Region
IOSGA	Input/Output Support Gate Array
IP	Internet Protocol (TCP/IP), Information Provider,

	Instruction Pointer
IPAE	IP Address Encapsulation
IPC	Interprocess Communication, Instructions per Clock
IPCP	Internet Protocol Control Protocol
IPDS	Intelligent Printer Data Stream
IPF	Information Presentation Facility
IPI	Intelligent Peripheral Interface
IPL	Initial Program Loader, Interrupt Priority Level
IPM	Intelligent Power Mode/Module/MOS, Interpersonal Message
IPng	Internet Protocol Next Generation
IPON	Intelligent Passive Optical Network
ips	inch per second
IPSE	Integrated Project Support Environment
IPTO	Information Processing Techniques Office
IPVR	Inst. für parallele und verteilte Höchstleistungsrechner, Stuttgart
IPX	Internetwork Packet Exchange (Novell)
IPXCP	Internetwork Packet Exchange Control Protocol
IQL	Interactive Query Language
IR	Information Retrieval, Infrarot
IRAF	Image Reduction and Analysis Facility
IRB	Instrumentation Request Broker
IRC	Internet Relay Chat
IRD	Integrated Receiver Decoder (TV)
IrDA	Infrared Data Association
IRD	Internet Resource Directory
IRDS	Information Resource Dictionary System
IRL	Inter Repeater Link, In Real Life (Slang)
IRLED	Infrared Light Emitting Diode
IRM	Information Resource Management
IRQ	Interrupt Request
IRS	Internal Revenue Service
IRTF	Internet Research Task Force
IS	Information Separator/System, In Service, Interrupt Status, Intermediate System, International Standard (ISO), Interactive Service
ISA	Industry Standard Architecture, Instruction Set Architecture, Interactive Services Association
ISAM	Index Sequential Access Method
ISC	Instruction Set Computer, Internet Service Center
ISD	Image Section Descriptor, Instructional System Design
ISDN	Integrated Services Digital Network
ISH	Information Superhighway
ISI	Internally Specified Index
IS-IS	Intermediate System to Intermediate System Protocol (OSI)
ISL	Interactive System Language, Intersatellite Link
ISM	Industrial, Scientific, Medical
ISN	Internet School/Shopping Networking
ISO	International Standardization Organization
ISOC	Internet Society

ISODE	ISO Development Environment (OSI)
ISP	International Standardized Profile, Internet Service Provider, Interrupt Stack Pointer, Interrupt Status Port
ISPBX	ISDN Private Branche Exchange
ISPF	Interactive Structured Programming Facility (IBM)
ISPI	Institut für Integrierte Publikationssysteme, Darmstadt
ispLSI	in-system-programmable LSI
ISPO	Information Society Project Office
ISR	Intermediate Session Routing, Information Storage and Retrieval, Interrupt Service Routine
ISSA	Information Systems Security Association
ISSCC	Intenational Solid State Circuits Conference
ISSP	Inter-Switch Signalling Protocol
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 Vendor
ISVR	Intel Smart Video Recorder
IT	Information Technology
ITA	Interface Test Adapter, Interim Type Approval
ITAC	ISDN Terminal Adapter Circuit
ITB	Intermediate Text Block, Information Technology Branch
ITE	Informationstechnische Einrichtung
ITG	Informationstechnische Gesellschaft im VDE
ITI	Interactive Terminal Interface
ITLB	Instruction TLB
ITO	Indium Tin Oxide
ITR	Internet Talk Radio
ITS	Incompatible Time-Sharing System, Institute for Telecommunication Sciences, USA
ITSEC	Information Technology Security Evaluation Criteria
ITSEM	Information Technology Security Evaluation Manual
ITSTC	Information Technology Steering Comitee
ITT	Invitation To Transmit (Arcnet)
ITU	International Telecommunications Union, Intelligent Thermal Update
ITUG	International Telecommunications Users' Group
ITV	Interactive Television
ITX	Intermediate Text Block
IU	Integer Unit
IUG	Informix User Group
IuK	Information und Kommunikation
IV	Initialisation Vector
IVDLAN	Integrated Voice and Data Local Area Network
IVHS	Intelligent Vehicle Highway System
IVIS	Interactive Video Information System
IVOD	Interactive Video On Demand
IVP	Installation Verification Image

IVR	Interactive Voice Response
IVS	IBM Verkabelungssystem, Interactive Visualization System
IVT	Interrupt Vector Table
IVV	Independent Verification and Validation
IWBNI	It Would Be Nice If (Slang)
IWF	Impulswahlverfahren (Telefon)
IWU	Interworking Unit
IXI	International X.25 Interconnect

JA	Jump Address
JAD	Joint Application Design
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
JES	Job Entry System
JESSI	Joint European Submicron Silicon Initiative
JFET	Junction Field Effect Transistor
JFIF	JPEG File Interchange Format
JFS	Journaled File System
JICST	Japan Information Center of Science and Technology
JOSS	Joint Object Services Submission
JPEG	Joint Photographic Experts Group
JPL	Jyacc Procedural Language, Jet Propulsion Laboratory
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

K-12	Kindergarten through 12th grade
KB	Kilobyte, Keyboard
kbps	Kilobit/byte per second
KCL	Kyoto Common Lisp
KCS	Kansas City Standard
KDC	Key Distribution Center
KDT	Key Definition Table
KES	Key Escrow System
KF	Family Key
KNF	Konjunktive Normalform
KI	Künstliche Intelligenz
KIBO	Knowledge In, Bullshit Out (Slang)
KIS	Knowbot Information Service
KISS	Keep It Simple/Small and Stupid/Simple
KIT	Kernel Software for Intelligent Terminals

KOALA	Konstanzer Ausleih- und Anfragesystem
KOP	Kontaktplan (DIN 19 239)
K&R	Kernighan + Ritchie
KRS	Knowledge Retrieval System
KS	Session Key
KSAM	Keyed Sequential Access Method
KTA	Korean Telecommunication Authority
KtK	Kommission für den Ausbau des technischen Kommunikationssystems
KU	Unit Key
KUUG	Karlsruher UNIX User Group
KV	Kabelverzweiger, Karnaugh Veitch (Diagramm)
KWIPS	Kilo Whetstones Per Second
KZU	Kennzeichenumsetzer

LADDR	Layered Device Driver Architecture
LADE	Language Definition Environment
LAI	LAN Automatic Inventory, Location Area Identity
LALL	Longest Allowed Lobe Length
LAM	Lobe Attachment Unit
LAN	Local Area Network
LANE	LAN Emulation
LAP	Link Access Protocol/Procedure
LAPB	Link Access Procedure for Balanced Mode
LAPD	Link Access Procedure for the D Channel
LAPM	Link Access Protocol for Modems
LAPS	LAN Adapter and Protocol Support (IBM)
LAR	Load Access Right
Laser	Light amplification by stimulated emission of radiation
LAST	Local Area Storage Transport (DEC)
LAT	Local Area Transport (DEC)
LATA	Local Access and Transport Area
LAU	Lobe Access Unit
LAVC	Local Area VAX Cluster
LAWN	Local Area Wireless Network
LBA	Logical Block Addressing
LBC	Local Bus Controller
LBM	Local Bus Master
LBT	Local Bus Target, Listen Before Talk
LBX	Low Bandwidth X
LCA	Losely Coupled Architecture, Logic Cell Array
LCD	Liquid Crystal Display
LCF	LAN Configuration Facility
LCHU	Low Cost Home User
LCI	Logical Channel Indentifier
LCM	LEAF Creation Method
LCP	Link Control Protocol (Internet)
LCS	Liquid Crystal Shutter
LCU	Last Cluster Used
LDA	Logical Device Address, Local Delivery Agent

LDLN	Long Distance Learning Network
LDM	Long Distance Modem
LDR	Light Dependent Resistor
LDS	Landesamt für Datenverarbeitung und Statistik
LDSG	Landesdatenschutzgesetz
LDT	Local Descriptor Table
LDTR	Load Descriptor Table Register
LDTRC	Load Descriptor Table Register Cache
LDTV	Limited Definition Television
LE	Link Encapsulation
LEAF	Law Enforcement Access Field
LEARP	LAN Emulation Address Resolution Protocol
LEC	LAN Emulation Client
LECS	LAN Emulation Client/Configuration Server/Service
LED	Light Emitting Diode
LEL	Link Embedded and Launch-to-edit
LEM	Language Extension Module
LEMP	Lightning Electromagnetic Pulse
LEN	Low Entry Networking
LEO	Low Earth Orbit/Orbiting
LER	Light Emitting Resistor
LES	LAN Emulation Server, Land-Erdfunkstelle
LF	Line Feed
LFI	Last File Indicator
LFK	Landesanstalt für Kommunikation
LFS	Local File System
LFSA	List of Frequently Seen Acronyms
LFU	Least Frequently Used
LGA	Low Gain Antenna
LGPL	Library General Public License (GNU)
LH	Link Header
LIC	Logical Link Control
LICS	Lotus International Character Set
LIEP	Large Internet Exchange Packet (Novell)
LIF	Low Insertion force
LIFO	Last In, First Out (Speicher)
LILO	Last In, Last Out (Speicher)
LIM	Lotus, Intel, Microsoft
LIMS	Library Information Management System
LIP	Large Internet Packet, Line Interconnection Point
LIPS	Logical Inferences per Second
LIPX	Large Internet Packet Exchange
LIS	Logical IP Subnetwork
LISP	List Processing (Language), Lots of Irritating Superfluous Parentheses
LITA	Library and Information Technology Association, USA
LKM	Loadable Kernel Modul
LKS	Loadable Kernel Server
LL	Local Loopback

LLA	Link Level Access
LLC	Logical Link Control
LLC2	Logical Link Control Class 2
LLP	Link Layer Protocol
LLTA	Lots and Lots of Thundering Applause (Slang)
LMBCS	Lotus Multibyte Character Set
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
LOC	Lines Of Code, Loop On-Line Control
LOCIS	Library Of Congress Information System, USA
LOF	Loss of Frame
LOL	Laughing Out Loud (Slang)
LON	Local Operating Network
LOP	Loss of Pointer, Logikplan (DIN 19 239/40 719)
LORE	Line Oriented Editor
LOS	Loss of Signal
LP	Line Printer, Logical Partition
LPC	Linear Predictive Coding, Local Procedure Call
LPD	Landespostdirektion, Line Printer Daemon (UNIX)
lpi	lines per inch
LPL	Lotus Programming Language
lpm	lines per minute
LPN	Logical Page Number
LPP	Licensed Program Product
lps	lines per second
LPS	Low Power Schottky (Semiconductor)
LPT	Line Printer
LQ	Letter Quality
LQM	Link Quality Monitoring
LQR	Link Quality Report
LRC	Longitudinal Redundancy Check, Local Register Cache
LRE	Limited Regular Expression
LRI	Least Recently Loaded
LRPC	Lightweight Remote Procedure Call
LRU	Least Recently Used, Lowest Replaceable Unit
LSA	Link State Algorithm, Line Sharing Adapter, lötfrei – schraubfrei – abisolierfrei
LSB	Least Significant Bit
LSC	Least Significant Character
LSD	Least Significant Digit
LSI	Large Scale Integration
LSID	Local Session Identifier
LSZH	Low Smoke Zero Halogen (Kabel)

LT	Link Trailer
LU	Logical Unit
LUF	Lowest Usable Frequency
LUG	Local User Group
LUN	Logical Unit Number
LUNI	LAN Emulation User Network Interface
LUT	Look Up Table
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
MA	Management Agent
MAC	Mandatory/Medium/Media Access Control, Message Authentication Check/Code
MACP	Motion Adaptive Color Plus
MACS	Mixed Aloha Carrier Sense, Modem Access Control System
MAE	Macintosh Application Environment
MAN	Metropolitan Area Network
MAP	Manufacturing Automation Protocol, Maintenance Analysis Procedure, Mobile Application Part, Manufacturing Package
MAPI	Mail/Message Application Programmer/Programming Interface
MAPICS	Manufacturing, Accounting and Production Information Control System (IBM)
MAP/TOP	MAP Technical Office Protocol
MAR	Memory Address Register
MARS	Message Archiving and Retrieval Service
MASM	Macro Assembler (Microsoft)
MAU	Multi Station Access Unit, Medium Attachment Unit (Ethernet)
MAVT	Mobile Audio Visual Terminal
MAZ	Magnetbildaufzeichnung
MB	Megabyte
MBONE	Multicast Backbone
Mbps	Megabits/bytes per second
MBR	Master Boot Record
MBS	Mobiles Breitband-System
MBX	Mailbox
MCA	Micro Channel Architecture (IBM), Multiprocessor Communication Adapter, Mission Critical Application, Motion Capture and Analysis
MCB	Memory Control Block
MCC	Memory Cache Controller, Microelectronics and Computer Technology Corporation
MCCOI	Multimedia Communications Community of Interest

MCDA	Micro Channel Developers Association
MCGA	Multi Colour Graphics Adaptor/Array
MCI	Media Control Interface
MCM	Multichipmodul
MCP	Master Control Programm, Macintosh Coprocessor Platform, Microsoft Certified Professional
MCSE	Microsoft Certified Systems Engineer
MCT	MOS Controlled Thyristor
MCU	Micro Control Unit, Multi Chip Unit
MD	Message Digest
MDA	Monochrome Display Adapter, Mediated Digest Authentication
MDBS	Mobile Data Base Station
MDI	Medium Dependant Interface, Multiple Document Interface
MDIS	Mobile Data Intermediate System
MDK	Multimedia Development Kit (Microsoft)
MDNS	Managed Data Network Service
MDR	Minimum Design Requirement
MDT	Mittlere Datentechnik
MDY	Month Day Year (Datumsangabe)
ME	Mutation Engine
MEB	Memory Expansion Board
MEO	Medium Earth Orbit
MES	Mobile End System
MESI	Modified Exclusive Shared Invalid (Cache)
MF	Multifunktion
MFC	Multifunktionale Chipkarte, Microsoft Foundation Classes
MFFS	Microsoft Flash File System
MFLOP	Million Floating Point Operations
MFM	Modified Frequency Modulation
MFP	Multifunction Peripheral
MFS	Maximum Frame Size
MFT	Multi-Programming with a Fixed number of Tasks
MFTL	My Favorite Toy Language
MFV	Mehrfrequenz-Wahlverfahren (Telefon)
MGA	Monochrome/Multimode Graphic Adapter
MH	Modified Huffman
MHEG	Multimedia and Hypermedia Expert Group
MHF	Message Handling Facility
MHS	Message/Mail Handling Service/System
MIB	Management Information Base
MIC	Media Interface Connector, Macro Interpretative Command, Multimedia Interactive Control, Message Integrity Check
MICE	Multimedia Integrated Conferencing for Europe
MICR	Magnetic Ink Character Recognition
MID	Message Identifier
MIDI	Musical Instrument Digital Interface
MIF	Management Information Format
MII	Microsoft, IBM, Intel
mil	military; Internet-Domain: Militär in den USA

MIL	(amerikanische Militärnorm), Machine Interface Layer
MILNET	Military Network, USA
MIM	Metall-Isolator-Metall (Bildschirmtechnik)
MIMD	Multiple Instruction Multiple Data
MIME	Multipurpose Internet Mail Extensions
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
MIX	Member Information Exchange
MJ	Modular Jack (Steckersystem)
M-JPEG	Motion-JPEG
ML	Machine Language
MLI	Multiple Link Interface
MLP	Message Link Protocol
MLS	Multiple Link Support, Multi Level Security
MLT	Multi-Level Transition
MM	Maschinenlesbares Merkmal, Mobility Management
MMAC	Multi Media Access Center
MMC	Matched Memory Cycle
MMCD	Multimedia Compact Disc
MMDF	Multichannel Memorandum Distribution Facility
MMF	Multi-Mode Fiber
MMI	Man Machine Interface
MMIC	Monolithic Microwave Integrated Circuit
MMIS	Materials Manager Information System
MMJ	Modified Modular Jack (Steckersystem)
MMM	Memory Mapper Modul
MMPM	Multi Media Presentation Manager
MMS	Manufacturing Message Specification
MMTCA	Multimedia Toolbox for Cooperative Applications
MMU	Memory Management Unit
MNIT	Mobile Networks Integration Technology
MNP	Microcom Networking Protocol, Multiple Networking Program
MO	Magneto-Optical (Disk), Mobile Originated (Message)
MOD	Magneto-Optical Disk
MODEM	Modulator-Demodulator
MOHLL	Machine Oriented High Level Language
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
MOV	Metal Oxide Varistor
MP	Multiprocessing, Multiple Processors, Multilink Protocol
MPA	Minimum Perceptable Action

MPC	Microsoft Certified Professional, Multimedia PC
MP/CS	Manufacturing Planning and Control System
MPE	Multiprogramming Executive
MPEG	Motion/Moving Pictures Expert Group
MPF	Mapping Field
MPI	Message Passing Interface, Multi-Protocol Interchange
MPP	Message Posting/Processing Program/Protocol, Massively Parallel Processing/Processor
MPR	Multi-Protocol Router, Mät- och Provningsrådet (Schweden), Multipart Repeater
MPT	Ministry of Post and Telecommunications, UK + Japan
MPTF	Multiple Protocol Transport Feature
MPTN	Multi-/Multiple Protocol Transport Network (IBM)
MPTS	Multiprotocol Transport Services (IBM)
MPU	Microprocessor Unit, MIDI Processing Unit
MPW	Macintosh Programmers Workbench
MQ	Message Queuing
MQI	Messaging and Queuing Interface
MR	Modem Ready, magneto-resistiv
MRCC	Maritime Rescue Coordination Center
MRCF	Microsoft Realtime Compression Format
MRCI	Microsoft Realtime Compression Interface
MRM	Motif Resource Manager
MRO	Multi-Region Operation
MRP	Material Requirements Planning, Manufacturing Resource Planning
MRPL	Main Ring Path Length
MRB	Method Request Broker
MRT	Mean Repair Time
MRU	Maximum Receive Unit
MS	Memory System, Message Store, Microsoft Corporation, Mobile Station
MSAIS	Multiplexer Section Alarm Indication Signal
MSAP	Minislotted Alternating Priorities
MSB	Most Significant Bit
MSC	Mobile Switching Centre
MSCDEX	Microsoft Compact Disk Extensions
MSCP	Mass Storage Control Protocol
MSD	Mass Storage Device, Most Significant Digit
MS-DOS	Microsoft Disk Operating System
MSDR	Multiplexed Streaming Data Request
MSDS	Microsoft Developer Support
MSG	Message
MSI	Medium Scale Integration
MSK	Minimum Shift Keying
MSL	Mirrored Server Link, Map Specification Library
MSN	Multiple Subscriber Number, Microsoft Network
MSOH	Multiplexer Section Overhead
MSP	Multiplexer Section Protection

MSR	Model Specific Register
MSS	MAN Switching System
MSTE	Multiplexer Section Terminating Equipment
MSW	Machine Status Word
MT	Mobile Terminated (Messaging)
MTA	Mail/Message Transfer Agent (OSI), Multiple Terminal Access
MTBB	Mean Time Between Breakdowns
MTBF	Mean Time Between Failure
MTBJ	Mean Time Between Jams
MTDA	Mean Time Data Availability
MTF	Microsoft Tape Format
MTP	Mail Transfer Protocol (Internet)
MTS	Message Transfer Services/System, Multichannel Television Sound, Multimedia Teleschool (Berlitz)
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
MTU	Maximum Transmission Unit
MUA	Mail User Agent
MUD	Multiple-User Dialogues/Dimension (Spiele), Multi-User Dungeon
MUF	Maximal Usable Frequency
MUFF	Maus- und Fenster-Firlefanz
MUL	Mobile User Link
MUMPS	Massachusetts General Hospital Utility Multi-Programming System
MUNDI	Multiplexed Network for Distributive and Interactive Services
MUSICAM	Masking Universal Subband Integrated Coding and Multiplex
MUX	Multiplexer
MVB	Multimedia Viewer Book
MVC	Multimedia Viewer Compiler, Model View Controller (Smalltalk), Multiple Virtual DOS Machines, Karlsruhe // MVDM
MVDM	Multiple Virtual DOS Machines
MVGA	Monochrome Video Graphics Array
MVS	Multiple Virtual Storage (IBM)
MVT	Multiprogramming with a Variable number of Tasks
MX	Mail Exchange

NA	Network Access
NAB	National Association of Broadcasters (USA)
NACD	National Association of Computer Dealers
NACS	NetWare Asynchronous Communication Server, National Advisory Committee on Semiconductors
NAEC	Novell Authorized Education Center
NAG	Numerical Algorithms Group, Oxford
NAK	Negative Acknowledge/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
NAPLPS	North American Presentation Layer/Level Protocol Syntax

NAS	Network Application Services, Network Application Support (DEC)
NASA	National Aeronautics and Space Administration, USA
NASI	NetWare Asynchronous Services Interface (Novell)
NASK	Naukowa i Akademicka Siec Komputerowa (Polen)
NAU	Network Addressable Unit
NAX	Nachrichten auf Fax
NB	Nota bene
NBF	Netbios Frame Transport Protocol
NBP	Netbios Protocol, Name Binding Protocol
NBS	National Bureau of Standards, USA (jetzt NIST)
NBT	NetBIOS on TCP/IP
NC	Newcastle Connection, No Connection, Network Control, Numerical Control
NCA	Network Communications Adapter
NCC	Network Control Center
NCEG	Numerical C Extensions Group
NCGA	National Computer Graphics Association
NCI	Network Control Information
NCMT	Numerical Control for Machine Tools
NCNCNC	No Coffee, No Chocolate, No Computer (Slang)
NCP	Network Control Processor/Protocol/Program, NetWare Core Protocol, Not Copy Protected
NCR	National Cash Register Co., USA
NCS	Network Computing System
NCSA	National Center for Supercomputing Applications, USA
NCSC	National Computer/Computing Security Center
NCSI	Network Communications Services Interface
NCSL	National Computer System Laboratory
ND	Natürliche Dummheit (Gegensatz zu KI)
NDA	Norddeutsche Datenautobahn
NDIS	Network Device/Driver Interface Specification (Microsoft)
NDL	Network Database Language
NDM	Normal Disconnected Mode
NDP	Numeric Data Processor
NDPS	Non-disruptive Path Switching
NDS	Netware Directory Services
NDT	Network Design Tool
NE	Network Element
NEARnet	New England Academic and Research Network
NEC	National Electric Code (Kabel), Nippon Electric Company
NECC	National Education Computing Conference, USA
NEOS	Networked Educational On-line System
NEST	Novell Embedded Systems Technology
net	network; Internet-Domain: Netzverwaltung
NET	Norme Europeenne de Telecommunication
NetBEUI	NetBIOS Extended User Interface (IBM)
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
NFF	No Fault Found
NFS	Network File System (Sun)
NFSS	New Font Selection Scheme
NGM	Netware Global Messaging
NGMHS	NetWare Global Message Handling System
NGO	Non Governmental Organization
NI	Network Interconnect
NIC	Network Information Center, Network Interface Card
NICAD	Nickel Cadmium (Akkumulator)
NICE	Network Information and Control Exchange (Protocol)
NICK	Netzweites Informationssystem im Campus der Universität Karlsruhe
NID	Next ID
NIDL	Network Interface Definition Language
NIH	National Institute of Health, USA
NIHCL	NIH Class Library
NII	National Information Infrastructure, USA
NIIT	National Information Infrastructure Testbed
NIMH	Nickel Metal Hydride (Akkumulator)
NIMP	NetWare Internetwork Messaging Protocol
NIPS	Network I/O Per Second
NIR	Network Information Retrieval
NIS	Network Information Service
NISA	New Information Services Architecture
NISO	National Information Standards Organization
NIST	National Institute of Standards and Technology, USA
NITC	National Information Technology Center
NIU	Network Interface Unit
NJE	Network Job Entry (IBM)
NL	Network Layer, New Line
NLE	Nonlinear Editing
NLM	Netware Loadable Modul (Novell)
NLQ	Near Letter Quality
NLS	National/Native Language Support
NLSP	Netware Link Service Protocol
NLUUG	Niederländische UNIX User Group
NMA	Network Management Architecture
NMC	Network Management Center
NMI	Nicht maskierbarer Interrupt
NMM	NetWare Management Map (Novell)
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 (Internet)
NOC	Network Operation Center
NOMA	National Online Media Association
NOP	No Operation
NOR	not or
NORC	Naval Ordnance Research Calculator
NOS	Network Operating System
NPA	Network Performance Analyzer, Network Printer/Printing Alliance
NPAP	Network Printing Alliance Protocol
NPI	Network Printer Interface
NPL	Nonprocedural Language, National Physical Laboratory, England
NPTN	National Public Telecomputing Network, USA
NQE	Network Queuing Environment
NQS	Network Queuing System
NREN	National Research and Education Network
NRM	Normal Response Mode
NRTL	Nationally Recognized Testing Laboratory (USA)
NRZ	non return to zero
NRZI	non return to zero, invert on ones
NS	New Signal, Network Supervisor
NSA	National Security Agency, USA
NSAP	Network Service Access Point
NSC	National Support Center
NSF	Non Standard Format, National Science Foundation, USA
NSFNet	National Science Foundation Network (Internet)
NSL	Network Service Layer
NSP	Network Services Protocol, Native Signal Processing
NSSDC	National Space Science Data Center, USA
NSTL	National Software Testing Labs, USA
NSTVA	Nebenstellen-Vermittlungsanlage
NT	Network Termination (ISDN), New Technology
NTAS	Windows NT Advanced Server
NTC	Negative Temperature Coefficient
NTF	No Trouble Found
NTFS	New Technology File System (Microsoft)
NTG	Nachrichtentechnische Gesellschaft e. V.
NTIA	National Telecommunications and Information Administration, USA
NTN	Network Terminal Number
NTP	Network Termination Point, Network Time Protocol
NTS	Network Transport Services (IBM)
NTSA	Networking Technical Support Alliance
NTSC	National Television Standards/System Committee, USA
NTT	Nippon Telegraph + Telephone
NUA	Network User Address
NUI	Network User Identification/Interface
NURBS	Non Uniform Rational B-Spline
NUMA	Non Uniform Memory Address
NVE	Nummer der Versandeinheit

NVM	Non Volatile Memory
NVN	National Videotext Network
NVO	Non-Visual User Object
NVoD	Near Video on Demand
NVP	Network Voice Protocol (Internet), Nominal Velocity of Propagation
NVR	Non Volatile RAM
NYSERNet	New York State Educational and Research Network
OAF	Origin Address Field
OAI	Offene Anwendungs- und Interkommunikationssysteme, Open Applications Interface
OAM	Operation, Administration, Maintenance
OBEX	Object Exchange
OBST	Object Management System of STONE
OBO	Or Best Offer (Slang)
OC-3	Optical Carrier Level 3
OCCC	Obfuscated C Code Contest
OCE	Open Collaborative Environment (Apple)
OCL	Operator/Operation Control Language
OCLC	Online Computer Library Center
OCR	Optical Character/Curve Recognition
OCT	Object Code Translator
OCTD	Observation, Conclusion, Temporary Data
ODA	Open/Office Document Architecture
ODAC	Open Document Architecture Consortium
ODAM	Open Distributed Application Model
ODAPI	Open Database Application Programming Interface
ODBC	Open Data Base Connectivity (Microsoft)
ODBMS	Object Oriented Database Management System
ODCS	Open Distributed Computing Structure
ODD	Open Data Desktop
ODETTE	Organisation for Data Exchange by Tele-Transmission in Europe
ODI	Open Data-Link Interface (Novell)
ODIF	Open/Office Document Interchange Format
Odinsup	ODI-NDIS Supplementary Driver
ODL	Object Definition Language
ODM	Object Data Manager
ODMG	Object Database Management Group
ODN	Optical Data Network
ODP	Open Distributed Processing
ODS	Offenes Deutsches Schulnetz, Open Data Services
OE	Output Enable
OEF	Origin Element Field
OEM	Original Equipment Manufacturer, Odd Even Merge (Graph, Netz)
OES	Odd Even Sort (Graph, Netz)
OFB	Output Feedback Mode
OFD	Optical Frequency Division Demultiplexer/Demultiplexing
OFFIS	Oldenburger Forschungs- und Entwicklungsinstitut für

OFM	Optical Frequency Division Multiplexer/Multiplexing
OFS	Object File System, Output Field Separator
OFTP	Odette File Transfer Protocol
OH	Off Hook
OID	Object Identifier
OIDC	Object Identifier Component
OIDL	Object Interface Definition Language
OIRT	?
OIS	Office Information System
OIT	Object Identifier Tree
OIV	Object Idenitfier Value
OIW	OSI Implementors' Workshop
OLAP	Online Analytical Processing
OLE	Object Linking and Embedding
OLI	Optical Phone Line Interface
OLIT	Open Look Intrinsic Toolkit
OLIX	On-Line Informationssystem UNIX
OLT	Optical Line Termination
OLTP	On Line Transaction Processing
OM	Old Man (KW-Amateur), Object Manager, Optimal Mismatch
OMA	Open Management Architecture
OME	On-board Modul Extension
OMF	Object Management Facility/Framework, Object Module Format
OMG	Object Management Group
OMI	Open Messaging Interface, Open Microprocessor Systems Initiative
OMR	Optical Mark Recognition
OMT	Object Modelling Technique
ONC	Open Network Computing (Sun)
ONI	Operator Number Identification
ONM	Open Network Management
ONMS	Open Network Management System
ONN	Open Network Node
ONU	Optical Network Unit
ONP	Open Network Provision, Open Networking Platform
OOA	Object Oriented (System) Analysis
OODBMS	Object Oriented Database Management System
OOF	Out of Frame
OOK	On Off keying
OOL	Object Oriented Language
OOP	Object Oriented Pleasure, Out Of Print (Slang)
OOPL	Object Oriented Programming Language
OOPS	Object Oriented Programming System
OOPSLA	Object Orientied Programming Systems, Languages and Applications
OPA	Open Protocol Architecture
OPAC	Online Public Access Catalogue
OPAL	Optische Anschlußleitung
OPD	Oberpostdirektion, Original Point of Distribution
OPI	Open Prepress Interface

OPM	Operations Per Minute
OPT	Open Protocol Technology
OPUS	Octal Program Updating System
OQL	Object Query Language
OQPSK	Offset Quadrature Phase Shift Keying
ORB	Object Request Broker
org	organizational; Internet-Domain: Organisationen in den USA
OROM	Optical Read Only Memory
ORS	Output Record Separator
OS	Operating System
OSA	Open Scripting/System Architecture
OSAF	Origin Subarea Address Field
OSC	Open System Center (IBM)
OSCAR	Orbiting Satellite Carrying Amateur Radio
OSD	On Screen Display
OSE	Open Systems/Software Environment
OSF	Open Software Foundation
OSI	Open Systems Interconnect/Interconnection (ISO)
OSIE	Open Systems Interconnection Environment
OSITOP	Open Systems Interconnection Technical and Office Protocols
OSM	On Screen Menue
OSPF	Open Shortest Path First (Internet)
OSPI	Operating System Programming Interface
OSQL	Object Structured Query Language
OSS	Open Source Solution
OSTA	Optical Storage Technology Association
OSTM	Open System Transaction Management
OSTP	Office of Science and Technology Policy
OT	Object Technology, Off Topic (Slang)
OTA	U. S. Congress Office of Technology Assessment
OTD	Optical Time Division
OTDM	Optical Time Division Multiplexer/Multiplexing
OTDR	Optical Time Domain Reflectometer
OTF	Open Token Foundation
OTI	Original Transmitter Identification
OTOH	On The Other Hand (Slang)
OTP	One-Time Password/Programmable
OTR	One Touch Recording
OV	Overflow
OVL	Overlay
OVSt	Ortsvermittlungsstelle
OWG	Optical Waveguide
OWL	Object Windows Library

PA	Power Amplifier, Public Address
PABX	Private Access/Automatic Branch Exchange
PACS	Picture Archiving and Communications System, Personal Access Communication System
PaCT	PBX and Computer Teaming

PACX	Private Automatic Computer Exchange
PAD	Packet Assembler and Dissassembler
PaDL	Pattern Development Language
PAK	Packet (Arcnet)
PAL	Programmable Array Logic, Paradox/Programming Application Language, Phase Alternating Line/Loop (TV), Privileged Architecture Library (DEC)
PALC	Plasma Addressed Liquid Crystal (Display)
PAM	Personal Application Manager
PAP	Password Authentication Protocol, Printer Access Protocol, Public Access Profile, 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
PBE	Prompt By Example
PBM	Pipelined Burst Mode
PBX	Private Branch Exchange
PC	Personal Computer, Path Control, Program Counter
PCA	Parallel Channel Adapter
PCB	Printed Circuit Board, Program Control Block
PCC	Papas Computer-Club
PCD	Photo Compact Disc
PCI	Protocol Control Information, Peripheral Component/Controller Interface, Peripheral Component Interconnect (Bus), Programmable Communication Interface
PCL	Procedure, Process Control Language, Printer Control Language (HP)
PCM	Puls-Code-Modulation, Physical Connection Management
PCMCIA	PC Memory Card International Association
PCN	Personal Communication Network
PCNFS	Personal Computer Network File System
PCR	Phase Change Recording
PCS	Personal Communication System, PC Service (OSF), Project Control System. Print Contrast Signal, Personal Conferencing Specification
PCSA	Personal Computing Systems Architecture (DEC)
PCTE	Portable Common Tool Environment
PCTS	POSIX Conformance Test Suite
PCWG	Personal Conferencing Work Group
PD	Public Domain
PDA	Personal Digital Assistant
PDD	Physical Device Driver
PDES	Product Data Exchange using STEP
PDF	Portable Document Format, Processor Defined Function, Program Development Facility
PDH	Plesiochronous Digital Hierarchy
PDI	Packet Driver Interface

PDIAL	Public Dialup Internet Access List
PDL	Page Description Language, Primary Defect List, Program Design/Description Language
PDLC	Polymer Dispersed Liquid Crystal
PDN	Public Data Network
PDO	Portable Distributed Object, Packet Data Optimized
PDP	Plasma Display Panel, Programmable Data Processor
PDS	Public Domain Software, Packet Driver Specification, Partitioned Data Set
PDSS	Program/Post Development Support System
PDT	Programmable Drive Table
PDTE	Packet-mode DTE
PDU	Protocol Data Unit (OSI)
PDV	Path Delay Value
PE	Parity Error, Processing Element, Protective Earth, Phase Encoded
PEA	Pocket Ethernet Adapter
PEARL	Process and Experiment Automation Realtime Language
PEL	Picture Element
PEM	Privacy Enhanced Mail, Privacy Enhancement for Electronic 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
PES	Processor Enhancement Socket, Positioning Error Signal
PET	Personal Electronic Transactor (Commodore), Print Enhancement Technology
PEX	PHIGS Extension to X
PFA	Power FORTRAN Accelaerator
PFD	Programming Facility for Display Users
PFR	Power Fail Restart
PGA	Pin Grid Array, Professional Graphics Adapter, Program Global Area
PGP	Pretty Good Privacy/Piracy
PHCB	Page Header Control Bit
PHIGS	Programmers Hierarchical Interactive Graphics System
PHY	Physical Access, Physical Layer Protocol
PI	Program Interruption
PIA	Peripheral Interface Adapter
PIC	Programmable Integrated Circuit, Priority Interrupt Controller
PICS	Protocol Implementation Conformance Statement
PID	Process Identification Number
PIF	Programminformation, Postscript Interchange Format
PIK	Programmer's Imaging Kernel
PIM	Personal Information Manager/Management
PIN	Persönliche Identifikations-Nummer, Processor Independent NetWare
PINET	Physics Information Network
PIO	Parallel/Programmable I/O (Chip)
PIP	Peripheral Interchange Program, P-Internet-Protocol,

	Periodical Informational Posting, Picture in Picture
PIPO	Parallel In, Parallel Out
PIS	Personal-Informations-System
PIT	Programmable Interval Timer
PIU	Path Information Unit
PIW	Photo Imaging Workstation
PIXIT	Protocol Implementation Extra Informations for Testing
PJL	Printer Job Language
PK	Primary Key
PKCS	Public Key Cryptography Standards Group
PKI	Philips Kommunikationsindustrie AG
PL	Physical Layer, Presentation Layer, Programming Language, Payload Length
PLA	Programmable Logic Array
PLATO	Programmed Logic for Automatic Teaching Operations
PLC	Programmable Logic Controller
PLCC	Plastic Leadless Chip Carrier
PLD	Programmable Logic Device
PLE	Programmable Logic Element
PLIC	Programmable Logic Integrated Circuit
PLL	Permanent Logical Link, Phase Locked Loop
PLMN	Public Land Mobile Network
PLP	Presentation Level Protocol
PLR	Packet Lost Ratio
PLS	Programmable Logic Sequencer, Physical Layer Signaling
pLSI	Programmable Large Scale Integration
PLV	Presentation/Production Level Video
PM	Phasenmodulation, Preventive Maintenance, Presentation Manager, Performance Management
PMA	Physical Medium Attachment, Performance Management and Accounting
PMAP	Port Mapper
PMD	Physical Medium Dependent (Layer)
PMFJIB	Pardon Me For Jumping In But (Slang)
PML	Programmable Macro Logic
PMMU	Paged Memory Management Unit
PMOS	p-Channel MOS (Transistor)
PMR	Private Mobile Radio
PMW	Project Manager Workbench
PMxAS	Primär-Multiplex-Anschluß (ISDN)
PN	Path Number
PNCP	Peripheral Network Control Point
PNNI	Private Network/Node to Network Interface
PNW	Personal NetWare (Novell)
PO	Paper Out, Parity Odd
POCSAG	Post Office Standardisation Advisory Group
PODA	Priority-Oriented Demand Assignment
POE	Power Open Environment (IBM)
POET	Persistent Objects and Extended Database Technology
POF	Plastic Optical Fiber

POH	Path Section Overhead
POL	Problem Oriented Language, Point of Learning
PON	Passive Optical Network
POP	Point of Presence, Post Office Protocol (Internet)
POR	Pacific Ocean Region, Power-on Reset
POS	Point of Sales
POSE	Portable Operating System Extension
POSI	Promotion Conference for OSI, Japan
POSIX	Portable Operating System Interface for Computer Systems
POST	Power-on Self Test
POTS	Plain Old Telephone Service?System
POV	Point Of View (Slang)
POWER	Performance Optimized With Enhanced RISC (IBM/Motorola)
PP	Physical Partition, Portable Part, Predictive Pipelining
PPC	Personal Productivity Center, Production Planning and Control
PPDS	Personal Printer Data Stream
PPF	Plain Paper Fax
PPH	Pages per Hour
PPID	Parent Process Identification Number
PPL	Point-to-Point Link, Process-to-Process Link
PPM	Parts per Mille, Pages per Minute
PPP	Point-to-Point Protocol (Internet)
PPS	Production Planning System
PQET	Print Quality Enhancement Technology
PR	Pacing Response
PReP	PowerPC Reference Platform
PRI	Primary Rate Interface
PRISM	Parallel Reduced Instruction Set Multiprocessor
PRMD	Private Management Domain (X.400)
PRML	Partial Response Maximum Likelihood
PRNET	Packet Radio Network
PRNG	Pseudo Random Number Generator
PRO	Precision RISC Organization
PROFS	Professional Office System (IBM)
PROLOG	Programming in Logic
PROM	Programmable Read Only Memory
PRN	Printer
PRS	Print Management Service (OSF)
PRTI	Professional Realtime Interface
PS	Personal System (IBM), Proportional Spacing
PSC	Pittsburgh Supercomputer Center, Product Service Center
PSCT	Polymer Stabilized Cholesteric Texture
PSD	Platform Specific Driver
PSDN	Packet/Public Switched/Switching Data Network
PSE	Personen-Such-Einrichtung
PSE	Personen-Such-Einrichtung, Port Switched Ethernet
PSF	Permanent Swap File
PSK	Phase Shift Keying
PSN	Packet Switch Node

PSP	Program Segment Prefix
PSRAM	Pseudo Static Random Access Memory
PST	Pre-Structured Technology
PSTN	Public Switched/Switching Telephone Network
PSW	Program Status Word
PT	Page Table, Packet/Payload Type
PTB	Physikalisch-Technische Bundesanstalt, Braunschweig
PTD	Parallel Transfer Disk Drive
PTF	Program Temporary Fix, Pthread Framework
PTI	Payload Type Identifier/Indicator
PTN	Plant Test Number
PTT	Post, Telephon, Telegraph
PTZ	Posttechnisches Zentralamt, Darmstadt
PU	Physical Unit
PUL	Packet Underway Limit
PUS	Processor Upgrade Socket
PV	Physical Volume
PVC	Permanent Virtual Channel/Circuit/Connection
PVM	Parallel/Passthru Virtual Machine (IBM)
PVN	Private Virtual Network
PWB	Programmer's Workbench
PWI	Public Windows Initiative/Interface
PWR	Power (Saft)
PzM	Punkt-zu-Punkt (-Verbindung, ISDN)
PzM	Punkt-zu-Mehrpunkt (-Verbindung, ISDN)

Q&A	Questions and Answers
QAM	Quadraturamplitudenmodulation
QBE	Query By Example
QBF	Query By Form
QCIF	Quarter Common Intermediate Format
QDL	Quadri Data Layer
QFA	Quick File Access
QFP	Quadratic Flatpack
QIC	Quarter Inch Cartridge
QID	Quanteninformationsdynamik
QLLC	Qualified Logical Link Control
QMF	Query Management Facility
QOS	Quality of Service
QPSK	Quadrature/Quaternary Phase Shift Keying
QRI	Queue Response Indicator
QSDPCM	Quadtree-strukturierte Differenz-Puls-Code-Modulation
QTAM	Queued Teleprocessing Access Method

RA	Regulärer Ausdruck, Return Authorization
RACE	Research And Development in Advanced Communication Technologies in Europe
RACF	Resource Access Control Facility (IBM)

RAD	Rapid Application Development
RAG	Row Adress Generator
RAI	Remote Alarm Indication
RAID	Reundant Array of Inexpensive/Independent Discs
RALU	Register Arithmetic Logic Unit
RAM	Random Access Memory
RAMP	Remote Access Maintenance Protocol
RAP	Rapid Application Prototyping
RARE	Reseaux Associes de/pour la Recherche en Europe
RARP	Reverse Address Resolution Protocol (Internet)
RAS	Random Access Storage, Remote Access Service, Row Address Strobe, Reliability, Availability and Serviceability
RASMED	Redundant Array of Slightly More Expensive Disks
RATP	Reliable Asychronous Transfer Protocol (Internet)
RAW	Read After Write
RBBS	Remote Bulletin Board System
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
RCDATA	Replaceable Character Data
RCP	Retransmission Control Procedure
RCS	Resource Construction Set, Revision Control System, Record Communication Switching System
RCV	Receive
RD	Receive Data, Request Disconnect, Recursive Design
R&D	Research and Development
RDA	Remote Data/Database Access
RDB	Relational Database, Receive Data Buffer
RDBMS	Relationales Datenbank-Management-System
RDD	Replaceable Database Driver
RDP	Reliable Data Protocol (Internet)
RDRAM	Rambus DRAM
RDS	Radio Data System, Remote Data Services
RDSR	Receiver Data Service Request
RDSS	Radio Determination Satellite Services
RDST	Rotating Head Digital Storage Tape
RE	Reverse Engineering, Regular Expression, Report Engine
REJ	Reject
REM	Remark, Remote, Ring Error Monitor
REO	Removable Erasable Optical Disk
REP	Reply to Message Number
REQ	Request
RES	Reset, Remote Execution Service
RET	Resolution Enhancement Technology (HP)
REXX	Restructured Extended Executor (IBM)
RF	Radio Frequency, Remote File
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)
RFU	Reserved for Future Use
RGB	Rot, Grün, Blau (Farbmodell)
RGDI	Renaissance Graphics Device Interface
RGP	Raster Graphics Processor
RG/U	Radio Guide/Utility (Kabel)
RH	Request Header, Response Header
RHC	Regional (Bell) Holding Company
RHNT	Registration Hierarchical Name Tree
RI	Referential Integrity
RIB	RenderMan Interface Bytestream
RIF	Routing Information Field
RIFF	Resource Interchange File Format
RIM	Request Initialization Mode, Remote Installation and Maintenace
RIP	Routing Information Protocol, Raster Image Processor, Remote Image Protocol/Processing
RIPE	Reseaux IP Europeenne
RIPL	Remote Initial Program Loading
RISC	Reduced Instruction Set Code/Computer
RITA	Reduced Interim Type Approval
RJ	Registered Jack
RJE	Remote Job Entry (IBM)
RL	Remote Loopback, Real Life (Slang)
RLE	Run Length Encoded
RLG	Research Libraries Group
RLIN	Research Libraries Information Network (RLG)
RLL	Run Length Limited, Radio in the Local Loop
RLN	Remote LAN Node
RLSD	Received Line Signal Detected
RLV	Ringleitungsverteiler
RMON	Remote (Network) Monitoring
RMS	Root Mean Square, RAID Management Software, Record Management Services
RMT	Ring Management
RNA	Rufnummernanzeige
RNIS	Reseau Numerique a Integration de Services
RNR	Receive Not Ready (frame)
RO	Read Only
ROFLMAO	Rolling On The Floor Laughing My Ass Off (Slang)
ROI	Rectangle Of Interest
ROM	Read Only Memory

RoD/PO	Reading on Demand/Paper Option
ROPS	Remote Operations Service
ROS	Resident Operating System, Record On Silicon
ROSE	Remote Operations Service Element/Entity
ROTD	Release Of The Day
ROTFL	Rolling On The Floor Laughing (Slang)
ROW	Rest Of the World
RPC	Remote Procedure Call
RPE-LTP	Regular Pulse Excitation with Long-Term Prediction Loop
RPG	Record Program Generator (IBM)
RPL	Requested Privilege Level, Resident Programming Language
rpm	Revolutions per minute
RPN	Reverse Polish Notation
RPPROM	Reprogrammable PROM
RPS	Redundant Power Supply
RQBE	Relational Query By Example
RR	Receive/Receiver Ready, Real Reality, Round Robin
RRZN	Regionales Rechenzentrum Niedersachsen, Hannover
RS	Record Separator, Request to Send, Recommended Standard, Reed Solomon (Code)
RS232C	(Norm für serielle Schnittstelle)
RSA	Rivest, Shamir, Adleman
RSCS	Remote Spooling and Communications Subsystem
RSCV	Route Selection Control Vector
RSET	Reset
RSI	Repetitive Strain Injury
RSIS	Relocateable Screen Interface Specification
RSL	Request and Status Link
RSN	Real Soon Now (Slang)
RSOH	Regenerator Section Overhead
RSP	Required Space Character
RSPC	Reed Solomon Product Code
RSRB	Remote Source Route Bridging
RST	Receive Start Threshold, Reset and Restart; Readability, Signal Strength, and Tone
RSTE	Regenerator Section Terminating Equipment
RSTS	Resource Sharing and Time Sharing
RSX	Real Time Resource Sharing Executive
RT	Real/Run Time, Receive Timing, RISC Technology
RTAM	Remote Terminal Access Method
RTC	Real Time Clock
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)
RTFF	Read The Fantastic FAQ (Slang)
RTFM	Read The Fine/Fantastic/Funny ... Manual (Slang)
RTFS	Read The Funny Source (Slang)

RTK	Real Time Kernel
RTL	Register Transfer Level, Run Time Library, ResistorTransistor Logic
RTM	Real Time Modeling, Response Time Monitor, Read The Manual (Slang)
RTMP	Routing Table Maintenance Protocol
RTOS	Real Time Operating System
RTP	Rapid Transport Protocol, Rendezvous and Termination Protocol
RTOS	Real Time Operating System
RTS	Ready/Request To Send, Real Time System
RTSE	Reliable Transfer Service Element (OSI)
RTTI	Run Time Type Information/Identification
RTTY	Radio Teletype (Funkfernschreiben)
RTV	Real Time Video
RU	Request Unit, Response Unit
RUA	Remote User Agent
RUC	Reservation Upon Collision (Scheme)
RUM	Rechenzentrum Universität Mannheim
RUS	Rechenzentrum der Universität Stuttgart
RV	Real Virtuality
RVI	Reverse Interrupt
RWI	Reset Window Indicator
RWM	Read Write Memory
RX	Receive Data, Receiver
RZ	Rechenzentrum

SA	System Administrator, Structured Analysis
SAA	Systems 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, Service Access
SADT	Structured Analysis and Design Technique
SAINT	Symbolic Automatic Integrator
SAM	Serial Access Memory, Sequential Access Method, System Administration Manager, Smart ATM Modul
SAN	Satellite Access Network
SAP	Service Advertising Protocol (Novell), Service Access Point (OSI)
SAPI	Service Access Point Identifier
SAR	Specific Absorption Rate, Segmentation And Reassembling
SARM	Set Asynchronous Response Mode
SARME	Set Asynchronous Response Mode Extended
SAS	Statistical Analysis System, Single Attached Station (FDDI)
SASI	Shugart Associates System Interface
SATAN	Security Administrator's Toolfor Analyzing Networks
SAW	Surface Acoustic Wave (-Filter)
SB	Standby Indicator
SBC	Single Board Computer
SBCS	Single Byte Character Set

SBP	Structured Buffer Pool
SC	Send Common, Subcommittee, Secondary Channel
SCA	Security Control and Audit
SCADA	Supervisory Control And Data Acquisition
SCAM	SCSI Configured Automagically
SCART	Syndicat des Constructeurs d'Appareils Radio Recepteurs et Televiseurs
SCB	subsystem control Block
SCC	Standards Coordinating Committee (IEEE/CS), Serial Communication Controller, Synchronous Channel Check, Satellite control Center
SCCP	Signalling Connection Control Port
SCD	Structured Charts Diagram, Standard Color Display
SCF	System Control Facility
SCI	Scalable Coherent Interface
SCM	Software Configuration Management
SCMS	Serial Copy Management System
SCN	System Change/Commit Number
SCNR	Sorry, Could Not Resist (Slang)
SCO	The Santa Cruz Operation, Inc.
SCP	Session Control Protocol, Service Control Point, Subsystem Control Port
SCPI	Standard Commands for Programmable Instruments
SCR	Silicon Controlled Rectifier, Standard Context Routing
SCS	Secondary Clear to Send, System Communication Services, SNA Communication Service
SCSA	Signal Computing System Architecture
SCSI	Small Computer Systems Interface
SD	Send Data, Single Density, Structured Design, Starting Delimiter, Super Density (Compact Disc)
SDA	System Design Automation, Start of Domain Authority
SDDI	Shielded Twisted Pair Distributed Data Interface
SDF	Signal Delay Format, System Data Format, Space Delimited File/Format
SDH	Synchronous Digital Hierarchy
SDI	Selective Dissemination of Information
SDigN	Vorschriftensammlung für digitale Netze
SDK	Software Developer's Kit (Microsoft)
SDLC	Synchronous Data Link Control (IBM)
SDLP	Standard Device Level Protocol
SDM	Sub-rate Data Multiplexer, Short Data Message
SDMA	Space Division Multiple Access
SDMS	SCSI Device Management System
SDN	Software Defined Network
SDNS	Secure Data Network Service
SDP	Standard Directory Package (Apple)
SDR	Streaming Data Request
SDRAM	Synchronous DRAM
SDS	Software Distribution Service (OSF)

SDSC	San Diego Supercomputer Center
SDTV	Standard Definition Television
SDU	Service Data Unit
SDXC	Synchronous Digital Cross Connect
SEA	Society for Electronic Access
SEAL	Screening External Access Link
SEC	Single Error Correction
SECAM	Sequentiel Couleur Avec Memoire
SED	Smoke Emitting Diode
SEDAS	Standardregelungen einheitlicher Datenaustausch
SEM	Server Enhancement Module
SEO	Sequences Number
SEP	Someone else's problem
SER	Soft Error
SE-SCSI	Single Ended SCSI
SEU	Smallest Executable Unit
SF	Select Frequency, Store and Forward, Sign Flag
SFAusl	Vorschriftensammlung Fernmeldeverkehr Ausland
SFF	Small Form Factor
SFMJI	Sorry For My Jumping In (Slang)
SFPS	Secure Fast Packet Switching
SFQL	Structured Full-text Query Language
SFS	Secure File System
SFT	System Fault Tolerance, System File Table
SG	Signal Ground
SGA	System Global Area
SGI	Silicon Graphics, Inc.
SGML	Standard Generalized Markup Language
SGMP	Simple Gateway Management/Monitoring Protocol (Internet)
S/H	Sample and Hold
SHA	Secure Hash Algorithm
SHAR	Shell Archive
SHIA	Strongly Hedonistic Internet Area
SHIFT	Scalable, Heterogeneous, Integrated Facility
SHRS	Sehen, Hören, Riechen, Schmecken (Multimedia)
S-HTTP	Secure HTTP
SI	Shift In, Systeme International d'Unites, Signalling Rate Indicator, Source Index, System Information
SIA	Semiconductor Industries Association
SIAM	Supraleitendes Intelligentes Antennen-Modul
SIASL	Stranger In A Strange Land (Slang)
SIB	Scaled Index Based
SID	Station Identification, Symbolic Interactive Debugger, Silence Descriptor
SIG	Selective Integral Ground, Software/Special Interest Group
SIGCAT	Special Interest Group on CD-ROM Applications and Technology
SIGPLAN	Special Interest Group on Programming Languages
SIH	Super Information Highway

SII	Static Invocation Interface
SILK	Speech, Image, Language, Knowledge
SIM	Set Initialization Mode, Subscriber Identification Module
SIMD	Single Instruction Multiple Data (stream)
SIMM	Single In-Line Memory Module
SIMPLE	Sheer Idiots Monopurpose Programming Language Environment
SIN	Service Indicator (ISDN)
SINA	SISY Integriertes FTP-Server Archiv, Karlsruhe
SIO	Serial Input Output, Simultaneous Interface Operation
SIP	Simple Internet Protocol, SMDS Interface Protocol
SIPO	Serial In Serial Out
SIPP	Single In-Line Pin Package, Simple Polygon Processor, Simple Internet Protocol Plus
SIR	Serial Infrared Communication (HP)
SISD	Single Instruction Single Data(stream)
SISO	Serial In, Serial Out
SISY	Software Informations-System, Karlsruhe
SIT	Sspecial Information Tone
SIWG	Special Interest Working Group
SK	Synchronknoten, Schneider & Koch, Ettlingen
SL	Session Layer
SLC	Synchronous Link Control
SLIP	Serial Line Interface Protocol (Internet)
SLSI	Super Large Scale Integration
SM	Security Management, Set Mode
SMA	Sharing with Minimum Allocation
SMART	Self Monitoring, Analysis and Reporting Technology
SMB	Server Message Block
SMC	Standard Microsystems, Inc.
SMCC	Sun Microsystems Computer Corp.
SMD	Storage Modul Device, Surface Mounted Device
SMDR	Station Message Detail Recording
SMDS	Switched Multi-Megabit Data Service, USA
SMDSU	Switched Multi-Megabit Data Service Unit
SMF	Single Mode Fiber, System Manager Facility
SMFA	System Management Functional Area
SMI	Structure of Management Information
SMIT	System Management Interface Tool (IBM)
SMM	System Management Mode
SMOP	Small/Simple Matter Of Programming
SMP	Simple Management Protocol (Internet), Standard Message Package (Apple), Symmetrical Multi-Processing
SMPS	Switching Mode Power Supply (Schaltnetzteil)
SMPTE	Society of Motion Picture and Television Engineers, USA
SMS	Service Management System, Subsystem Management Service (OSF), Short Message Service, Storage Management Services
SMSC	Short Message Service Center
SMT	Surface Mounted Technology, Station Mangement
SMTP	Simple Mail Transfer Protocol (Internet)

SMU	System Management Utility
SMXQ	Sharing with Maximum Queues
S/N	Signal/Noise (ratio)
SN	Sequence Number
SNA	Systems Network Architecture (IBM)
SNADS	SNA Distribution Services
SNAP	Streamlined Network Accelerated Protocol, Subnetwork Access 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)
SNMPV2	Simple Network Management Protocol Version 2 (Internet)
SNOBOL	String Oriented Symbolic Language
SNR	Signal to Noise Ratio
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
SOC	System On a Chip
SODIMM	Small Outline DIMM
SOE	Standard Operating Environment
SOGT	Senior Officials Group on Telecommunications
SOH	Start of Header/Heading, Section Overhead
SOHO	Small Office, Home Office
SOL	Siemens Optical Link, Simulation Oriented Language
SOM	Start Of Message, System Object Model (IBM)
SONET	Synchronous Optical Network
SONETT	Soziales Netz Telecommunikation
SOP	Standard Operating Procedure
SOS	Silicon On Sapphire, Sophisticated Operating System
SOUP	Simple Offline Usenet Packet
SP	Space, Stack Pointer, System Program, Service Provider, Segment Priority
SPA	Software Publishers' Association, Service Point Application
SPAG	Standards Promotion and Application Group (OSI)
SPARC	Scalable Processor Architecture
SPC	Sub-Process Control, Small Peripheral Controller, Statistical Process Control
SPDL	Standard Page Description Language
SPDT	Single-Pole Double-Throw (Relay)
SPE	Synchronous Payload Envelope
SPEC	Systems PerformanceEvaluation Corporation
SPF	Shortest Path First, System Programming Facility
SPG	Service Protocol Gateway
SPI	Service Provider Interface

SPICE	Simulation Program with Integrated Circuit Emphasis
SPL	Spooler/Spooling
SPLD	Simple PLD
SPM	System Performance Monitor (IBM)
SPMD	Single Program Multiple Data
SPOOL	Simultaneous Peripherals Operation On Line
SPP	Sequencial Packet Protocol, Scalable Parallel Processing
SPS	Speicherprogrammierte Steuerung, Session Presentation Services, Standby Power System
SPT	Sectors Per Track
SPV	Semipermanente Verbindung
SPX	Sequenced 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, Shift Register, Source Routing
SRAM	Shadow/Static Random Access Memory
SRB	Source Route Bridging (IBM)
SRC	System Resource Controller
SRD	Secondary Receive Data, Screen Reader System
SREJ	Selective Reject
SRL	Structural Return Loss
SRM	Send Routing Message
SRMA	Split-Channel Reservation Multiple Access
SRO	Sharable and Read Only (memory)
SRPI	Server-Requester Programming Interface
SRQ	Service Request
SRR	Secondary Receiver Ready
SRS	Secondary Request to Send
SRT	Source Routing Transparent
SRUC	Split Reservation Upon Collision
SS	Sampled Servo, Select Standby, Single Sided, Single Shift, Stack Segment
SSA	Serial Storage Architecture
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
SSDC	Stack Segment Descriptor Cache
SSEC	Selective Sequence Electronic Calculator (IBM)
SSGA	System Support Gate Array
SSI	Small Scale Integration
S-SIMM	Single RAS SIMM
SSL	Secure Socket Layer
SSMA	Spread Spectrum Multiple Access
SSO	System Security Officer

SSP	Service Switching Point
SST	Simple SIPP Transition
SSTP	Screened Shielded Twisted Pair
SSTV	Slow Scan Television
SSU	Session Support Utility (DEC)
ST	Send Timing, Segment Type, Stream Protocol
STA	Spanning Tree Algorithm
STACK	Start Acknowledgement
STC	Subtechnical Committee (ETSI)
STD	Subscriber Trunk Dialing, State Transition Diagram
STDA	StreetTalk Directory Assistance (Banyan)
STDM	Statistical Time Division Multiplexing
STE	Signalling Terminal
STEP	Standard for the Exchange of Product Data
STL	Standard Template Library
STM	Synchronous Transfer/Transport Mode/Modul, Streams Manager
STN	Scientific & Technical Information Network, Super Twisted Nematic Field Effective Material
STONE	Structured and Open Environment
STP	Shielded Twisted Pair, Signal Transfer Point
STR	Synchronous Transmitter Receiver
STRT	Start Initialization
STS	Synchronous Transfer/Transport Signal
STT	Secure Transaction Technology
STU	Schere – Tippex – Uhu (-Verfahren)
STX	Start of Text
SU	Switching Unit
SUB	Substitute
SUM	Simple User Model
SUMR	Satellite User Mapping Register
SUN	Stanford University Network
SURAnet	Southeastern Universities Research Association Network
SUTP	Screened Unshielded Twisted Pair
SUUG	Soviet UNIX User Group
SV	Standortverteiler
SVC	Switched Virtual Call/Channel/Circuit/Connection
SVD	Simultaneous Voice and Data
SVGA	Super Video Graphics Array
S-VHS	Super Video Home System
SVID	Unix System V Interface Definition
SVP	Scan-line Video Processor
SVVS	(UNIX) System V Validation Suite
SW	Short Wave, schwarz-weiß, Software, Status Word
SWL	Short Wave Listener
SWF	Short Wave Fading
SWIM	Super Wozniak Integrated Machine
SWING	Subscriber Wireless Network Gateway
SXXM	Unextended Numbering Set Mode
SYLK	Symbolic Link Format (Microsoft)

SYN	Synchronous Idle
SYSOP	System Operator
TA	Terminaladapter, Informationstechnische Anschlußdose
TAB	Tabulator
TAC	Terminal Access Controller
TAD	Telephone Answering Device
TAE	Telekommunikations-Anschluß-Einheit
TAI	Temps Atomique International
TAM	Telephone Answering Machine; Telefon, Anrufbeantworter, Modem
TAN	Transaction Number, Total Area Network
TANJ	There Ain't No Justice (Slang)
TANSTAAFL	There Ain't No Such Things As A Free Lunch (Slang)
TAP	Terminal Access Point
TAPI	Telephony Application Programming Interface (Microsoft)
TAPR	Tucson Amateur Packet Radio Group
TAR	Tape Archive (Format)
TAXI	Transparent Asynchronous Transceiver Interface
TBETSI	Technischer Beirat für Normungsfragen des ETSI
TBF	Technisch betriebliche Funktionsbedingungen
TC	Technical Committee, Transmission Control/Convergence
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, Time Compression Multiplexing Terminal Controller Module
TCNS	Thomas Conrad Networking System
TCO	Tjänstemännens Centralorganisation (Schweden), Total Cost of Ownership
TCOS	IEEE Technical Committee on Operating Systems
TCP	Transmission Control Protocol/Program (Interent), Tape Carrier Package
TCP/IP	Transmission Control Protocol/Internet Protocol (Internet)
TCSEC	Trusted Computer System Evaluation Criteria (Orange Book)
TCU	Transmission Control Unit
TCW	Tagged Control Word
TD	Transmit Data
T-DAB	Terrestrial Digital Audio Broadcasting
TDCC	Transportation Data Coordinating Comittee
TDD	Time Division Duplex, Telecommunication Device for the Deaf
TDI	Transport Driver Interface
TDM	Time Division Multiplexer/Multiplexing
TDMA	Time Division Multiple/Multiplex Access
TDR	Time Domain Reflectometer/Reflectometry
TDRS	Tracking and Data Relay Satellite
TDSR	Transmitter Data Service Request

TDSV	Telekommmunikations-Datenschutzverordnung
TDU	Topology Database Update
TDX	Time Domain Crosstalk
TE	Terminal Equipment
TECO	Tape/Text Editor and Corrector
TED	Telefonischer Ermittlungsdienst
TEI	Terminal Endpoint Identifier
TEL	Trans Europe Line
Telex	Telegraph Exchange (Fernschreiben)
Telnet	Telecommunications Network
Temex	Telemetry Exchange
TETRA	Trans European Trunked Radio System
TF	Trägerfrequenz
TFA	Transparent File Access
TFD	Thin Film Diode
TFDD	Text File Device Driver
TFEL	Thin Film Electroluminescent
TFS	Translucent File System (Sun)
TFT	Thin Film Transistor
TFTP	Trivial File Transfer Protocol (Internet)
TFTS	Terrestrial Flight Telephone System
TG	Transmission Group
TGN	Transmission Group Number
TGS	Ticket Granting Server
TH	Transmission Header
THT	Token Holding Timer
THWLAIAS	The hour was late, and I am senile (Slang)
THX	Tomlinson Holman's Experiment
TI	Transaction Identifier, Texas Instruments Inc.
TIA	Telecommunications Industries Association, Thanks In Advance (Slang)
TIC	Token Ring Interface Coupler
TID	Target Identifier
TIDSV	Telekommunikations- und Informationsdienstunternehmen-Datenschutzverordnung
TIES	Time-Independent Escape Sequence, Telecom Information Exchange Services, Genf
TIFF	Tag/Tagged Image File Format
TIGA	Texas Instruments Graphics Architecture
TINAR	This Is Not A Review (Slang)
TIP	Terminal Interface Processor, Transputer Image Processing, Technology Integration Program
TK	Telekommunikation
TKO	Telekommunikationsordnung
TL	Transport Layer
TLB	Translation Look Aside Buffer
TLI	Transport Layer/Library Interface
TLNKG	Teilnehmerkennung (NUI)
TLQ	Top Letter Quality
TLU	Table Lookup

TM	Test Mode, Transaction Monitor
TMC	Traffic Message Channel
TMD	Time Division Multiplexing
TME	Tivoli Management Environment
TMN	Telecommunication Management Network
TMSL	Test and Measurement Systems Language
TN	Twisted Nematic, TelNet
TNA	Temex-Netzabschluss
TNC	Terminal Node Controller
TNFEM	Twisted Nematic Field Effective Material
TNHD	The New Hackers's Dictionary
TNN	Transport Network Node
TNPC	Taiwan New PC Consortium
TNX	Thanks (Slang)
TO	Timeout
TOEIC	Test of English for International Communication
TOF	Top of Form
TOP	Technical and Office Protocol, Table Of Pages
TOPFET	Temperature and Overload Protected FET
TOS	Type Of Service, Tramiel's Operating System (Atari)
TP	Transport Protocol, Transaction Processing, Turbo PASCAL, Tiefpass
TP0	Transport Protocol Class 0 (OSI)
TP4	Transport Protocol Class 4 (OSI)
TPA	Transient Program Area
TPC	Transaction Processing/Performance Council
TPDDI	Twisted Pair Distributed Data Interface
TPF	Transmission Priority Field
TPH	Transport Packet Header
tpi	tracks per inch
TPM	Third Party Maintenance, Transactions Per Minute
TPS	Transactions Per Second, Transaction Processing System
TPSE	Transaction Processing Service Element
TPTB	The Powers That Be (Slang)
TPU	Text Processing Utility, Turbo PASCAL Unit
TQM	Total Quality Management
TR	Terminal Ready, Task Register
TRAC	Telecommunications Recommendations Application Committee
TRAU	Transcoder and Rate Adaption Unit
TRMPWR	Terminator Power (Leitung, SCSI)
TRON	The Realtime Operating System Nucleus
TRT	Token Ring Terminal, Token Rotation Timer
TS	Transport Station
TSA	Technical Support Alliance
TSAP	Transport Service Access Point (OSI)
TSAPI	Telephony Server/Service Application Programming Interface
TSM	Terminal Server Manager (DEC)
TSO	Time Sharing Option (IBM)
TSP	Traveling Salesman Problem

TSR	Terminate and Stay Resident
TSS	Temex-Schnittstelle, Time Sharing System, Telecommunications Standards Sector (ITU), Task State Segment
TSSDC	Task State Segment Descriptor Cache
TST	Transmit Start Threshold
TSTN	Triple Supertwisted Nematic
TSW	Tele-Software
TT	Terminal Timing
TTBOMK	To The Best Of My Knowledge (Slang)
TTFN	Ta Ta For Now (Slang)
TTL	Transistor-Transistor-Logik, Time-To-Live
TTRT	Target Token Rotation Time
TTS	Transaction Tracking System
TTU	Teletex-Telex-Umsetzer
Ttx	Teletex
TTX	Videotext
TTXAU	Teletex Access Unit
TTY	Teletype (Fernschreiben)
TTYL	Type/Talk To You Later (Slang)
TU	Tributary Unit
TUBA	TCP and UDP with Big Addresses
TUeB	Temex-Uebertragungsbaugruppe
TUG	Tributary Unit Group, TeX User Group
TUI	Text User Interface
TUSTEP	Tübinger System von Textverarbeitungsprogrammen
TVI	Television Interference
TVR	Temex-Vermittlungsrechner
Tx	Telex (Fernschreiben)
TX	transmit

UA	User Agent (OSI), User Area, Unnumbered Acknowledgement
UAC	Unbalanced Asynchronous Class
UAE	Unrecoverable Application Error, Universelle Anschlußeinrichtung
UART	Universal Asynchronous Receiver Transmitter
UBC	University of British Columbia
UCB	University of California at Berkeley
UCFS	Unix Connectivity File Service
UCL	University College London, Universal Communications Language
UCLA	University of California at Los Angeles
UCSD	University of California at San Diego
UDC	User Defined Command
UDF	User Defined Function, Universal Disc Format
UDI	Universal Document Identifier
UDLC	Universal Data Link Control
UDL/I	Unified Design Language for Integrated Circuits
UDM	Unternehmensdatenmodell
UDP	User Datagram Protocol (Internet)
UDSV	Teledienst-Unternehmen-Datenschutzverordnung
UE	Unterhaltungs-Elektronik

UEB	Übertragungseinheit mit Basisbandverfahren
UEC	User Environment Component
UEM	Übertragungseinheit mit Modemverfahren
UEV	User End of Volume
UFS	Universal File System
UG	User Group, Underrun Goal
UHF	Ultrahigh Frequency (300 MHz – 3 GHz)
UHL	User Header Label
UI	UNIX International, Unnumbered Information, User Interface
UIAP	UNIX International Asia/Pacific
UID	User Identifier
UIFRAME	Unnumbered Information Frame
UIL	User Interface Language
UIM	User Interface Management
UIMS	User Interface Management System
UIT	Union Internationale des Telecommunications
UL	Underwriters' Laboratory, Upload
ULA	Uncommitted Logic Array
ULN	Universal Link Negotiation
ULP	Unit in the Last Place, Upper Layer Protocol
ULSI	Ultra Large Scale Integration
UMA	Upper Memory Area
UMB	Upper Memory Block
UMTS	Universal Mobile Telecommunications System
UN	United Nations
UNC	Unbalanced Normal Class, Universal Naming Convention
UN/GTDI	UN Guidelines for Trade Data Interchange
UNI	User Network Interface
UNICOM	Universal Integrated Communication system
UNIVAC	Universal Automatic Computer
UNMA	Unified Network Management Architecture
UP	Unnumbered Poll
UPC	Universal Printer Controller, Universal Product Code
UPE	User Portability Extension
UPI	United Press International
UPL	User Program Language
UPM	User Profile Management (IBM)
UPN	Umgekehrte Polnische Notation
UPPS	Universal Portable Protocol Stack
UPS	Uninterruptable Power Supply
URI	Universal Resource Identifier
URL	Uniform Resource Locator (Internet)
URN	Universal Resource Number
US	Unit Separator
USAN	University Satellite Network
USART	Universal Synchr. Asynchr. Receiver Transmitter
USB	Universal Serial Bus
USG	UNIX System Group (Novell)
USL	UNIX System Laboratories (AT&T, Novell)

USLE	UNIX System Laboratories Europe
USO	UNIX Software Operation
USRT	Universal Synchronous Receiver Transmitter
USSA	Universal Storage System Architecture
USTA	US Telephone Association
USV	Unterbrechungsfreie Stromversorgung
UT	Universal Terminal/Time
UTC	Universal Time Coordinated
UTI	Universal Text Interchange
UTL	User Trailer Label
UTLB	Unified TLB
UTP	Unshielded Twisted Pair
UTSL	Use The Source, Luke (Slang)
UUCP	UNIX to UNIX Copy Program (UNIX-Protokoll)
UULP	Unified User Level Protocol
UUS	User to Usewr Signalling
UUU	Unknown UNIX User (J. Random Hacker?)
UVL	User Volume Label

V.24	(CCITT-Norm für eine serielle Schnittstelle)
V+D	Voice + Data
VA	Volladdierer, Virtual Address
VAB	Value Added Business
VAD	Value Added Driver/Disaster/Dealer
VADD	Value Added Disk Driver
VADS	Value Added Data Service
VAFC	VESA Advanced Feature Connector
VAL	Voice Application Language
VAM	Virtual Access Method, Verband der Anbieter von Mobilfunkdiensten, Bonn
VAN	Value Added Network
VANS	Value Added Network Services
VAP	Value Added Process
VAR	Value Added Reseller/Retailer
VAS	Value Added Service
VASCOM	Value Added Services Network Communication
VAST	Vector and Array Syntax Translator
VAX	Virtual Address Extension (DEC)
VB	Variable Block
VBA	VisualBASIC for Applications
VBFS	Very Big File System
VBL	Vertical Blanking
VBN	Vermittelndes/Vermitteltes/Vorläufer Breitbandnetz (Telekom)
VBNS	Very High Backbone Network Service
VBR	Variable Bit Rate
VC	Virtual Circuit/Channel/Connection
VCC	Virtual Channel Connection
VCD	Virtual Communications Driver
VCI	Virtual Channel Identifier

VCL	Virtual Channel Link
VCO	Voltage Controlled Oscillator
VCPI	Virtual Control Programm Interface
VCR	Video Cassette Recorder
VCU	Voice Communication Unit
VD	Volume Descriptor
VDD	Virtual Device/Display Driver
VDDM	Virtual Device Driver Manager
VDE	Video Display Editor, Verband Deutscher Elektrotechniker
VDEW	Vereinigung Deutscher Elektrizitätswerke
VDI	Virtual Device Interface, Video Display Interface
VDM	Virtual DOS Machine
VDM-SL	Vienna Development Method - Semantic Language
VDN	Virtual Data Network
VDS	Virtual DMA Service
VDT	Video Display Terminal
VDU	Video/Visual Display Unit
VE	Volksempfänger
VEMM	Virtual Expanded Memory Manager
VESA	Video Electronics Standards Association
VEX	Video Extensions to XWS
VFD	Vacuum Fluorescent Display
VFIP	Voice File Interchange Protocol (Internet)
VFO	Variable Frequency Oscillator
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
VGC	Video Graphics Controller
VHDL	VHSIC Hardware Description Language
VHDSL	Very High Bit Rate Digital Subscriber Line
VHE	Virtual Home Environment
VHF	Very High Frequency (30 – 300 MHz)
VHS	Video Home System, Very High Speed, Virtual Host Storage
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/Software (Banyan)
VIO	Video/Virtual Input Output
VIOLD	Vision Impaired On-line Documentation (DEC)
VIP	Variable Information Processing
VIS	Voice Information system
VITA	VMEbus International Trade Association
VITAL	Virtually Integrated Technical Architecture Lifecycle (Apple)
VJ	Van Jacobson (Compression)
VJHC	Van Jacobson Header Compression

VLAN	Very/Virtual Local Area Network
VLB	VESA Local Bus
VLDW	Very Long Data Word
VLF	Very Low Frequency
VLIS	Verkehrs-Leit und Informations-System
VLIW	Very Long Instruction Word
VLM	Virtual Loadable Module
VLR	Visitor Location Register
VLSI	Very Large Scale Integration
VM	Virtual Machine, Virtual Memory
VMA	Virtual Memory Access
VMC	VESA Media Channel
VME	Virtual Memory Executive, Versacard Modified for Eurocard (Bus)
VMM	Virtual Machine/Memory Manager
VMS	Virtual Memory system, Voice Messaging System
VMT	Virtual Method Table, Virtual Memory Technique
VMTP	Versatile Message Transaction Protocol (Internet)
VNA	Virtual Network Architecture
VNET	Virtual Network
VNS	Virtual Notebook System
VO	Virtual Object
VoD	Video on Demand
VOF	Vollzugsordnung für den Funkdienst
VOL	Volume Label
VP	Virtual Path
VPA	Virtual Proxy Agent
VPC	Virtual Path Connection
VPD	Virtual Printer Device
VPDS	Virtual Private Data Service
VPI	Virtual Path Identifier
VPM	Virtual Protocol Machine
VPN	Virtuelles Privates Netz
VPRT	Verband Privater Rundfunk und Telekommunikation
VPS	Video Programm/Programmier- System, Verbindungsprogrammierte Steuerung
VPT	Virtual Printing Technology
VR	Virtual Route, Virtual Reality, Voltage Regulator
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
VRML	Virtual Reality Modeling Language
VRN	Virtual Route Number
VROOMM	Virtual Realtime Object-Oriented Memory Management
VRPRQ	Virtual Route Pacing Request
VRPRS	Virtual Route Pacing Response
VRU	Voice Response Unit
VS	Virtual Storage
VSA	Virtual Storage Access

VSAM	Virtual Storage Access Method
VSAT	Very Small Aperture Terminal
VSE	Virtual Storage Extended
VSF	Vertical Scanning Frequency
VSI	Verband der Software-Industrie Deutschland e. V.
VSK	Video-Sender-Kennung
VSL	Virtual Socket Library
VSM	Virtual Storage Management
VSOS	Virtual Storage Operating System
VSt	Vermittlungsstelle
VSX	Verification Suite for X/Open
VT	Vertical Tabulator, Video-Terminal, Virtual Terminal, Visualization Tool
VTAM	Virtual Terminal/Telecommunications Access Method
VTCO	Virtual Terminal Control Object
VTM	Verband der Telekommunikationsnetz- und Mehrwertdiensteanbieter, Köln
VTOC	Volume Table Of Contents
VTP	Virtual Terminal Protocol
VTS	Virtual Terminal Service
VUE	Visual User Environment
VUI	Video User Interface
VUIT	Visual User Interface Tool (DEC)
VUP	VAX Unit of Performance
VWM	Virtual Warehouse Manager
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
WACS	Wireless Access Communications System
WAD	Wählautomat für Datenverbindungen
WAENA	Wide Area Educational Network
WAIS	Wide Area Information Servers/Service
WAITS	Wide Area Information Transfer System
WAM	Warren Abstract Machine
WAMKSAM	Why Are My Kids Staring At Me? (Slang)
WAMPUM	Ward's Automated Menu Package Using Microcomputers
WAN	Wide Area Network
WAP	Wissenschaftlicher Arbeitsplatz
WARC	World Administrative Radio Conference
WATS	Wide Area Telephone/Telecommunications Service
WATTC	World Administrative Telegraph and Telephone Conference
WBC	Wide Band Channel
WC	Workgroup Computing, World Coordinate
WCS	Writable Control Store
WD	Western Digital

WDM	Wavelength Division Multiplexing
WE	Write Enable
WEEB	Western European EDIFACT Board
WELL	Whole Earth 'Lectronic Link
WFS	Woodstock File Server
WFW	Windows For Workgroups (Microsoft)
WG	Working Group (ISO)
WGDTB	Working Group on Digital Television Broadcasting
WGS	Work Group System
WHNF	Weak Head Normal Form
WIC	Wireless Indoor Communication
WIMP	Windows, Icons, Menus, Pointing (siehe auch MUFF)
WIN	Wissenschaftsnetz des DFN (X.25), Wireless Inhouse Network
WKS	Well Known Service/Sockets (Internet)
WLAN	Wireless LAN
WMF	Windows Metafile Format
WNIC	WAN Interface Coprocessor
WNIM	WAN Interface Module
WOM	Write-Only-Memory
WOMBAT	Waste Of Money, Brains, And Time (Slang)
WOP	Werkstattorientierte Programmierung
WORM	Write Once, Read Many/Multiple
WOSA	Windows Open Systems Architecture (Microsoft)
WP	Word Processor, Write Protected
WPHD	Write Protected Hard Disk
wpm	words per minute
WPS	Workplace Shell
WRT	With Respect To (Slang)
WS	Window Size, Workstation
WST	World System Teletext
WSTS	World Semiconductor Trade Statistics
WT	Write Through
WWW	W3, World Wide Web, Weltweites Warten, Weingartener Wein-Wandertag
WWWW	World Wide Web Wizard/Worm
WYSIWYG	What You See Is What You Get
WYSIWIS	What You See Is What I See

XA	Extended Architecture/Attribute
XAPIA	X.400-API Association
XCHG	Exchange
XCHS	Extended Cylinders, Heads, Sectors
XCMD	Extended Mode Command (Apple), External Command
XDCS	X/Open Distributed Computing Services
XDMCP	X Display Manager Control Protocol
XDR	External Data Representation
XFCN	External Function
XFS	X11 File System
XGA	Extended Graphics Array (IBM)

XID	Exchange Identifier/Identification
XIE	X Image/Imaging Extension (Standard)
XIE-DIS	XIE Document Imaging Subset
XKB	X Keyboard Extension
XLATE	Translate
XLINK	Extended Local Informatik Netz Karlsruhe
XLM	Excel Macro Language (Microsoft)
XMIT	Transmit
XMM	Extended Memory Manager
XMP	X/Open Management Protocol
XMS	Extended Memory Specification
XMT	Transmit
XNS	Xerox Network Services/System
XOFF	Exchange/Transmitter Off
XON	Exchange/Transmitter On
XOR	Exclusive or (auch: EOR)
XPC	X Performance Characterization Group
XPEL	X Protocol Engine Library
XPG	X/Open Portability Guide
XSC	Extended Scientific Calculation
XSI	X/Open Systenm Interface Specification
XSMD	Extended Storage Module Drive
XSMP	X Session Manager Protocol
XT	Extended Technology, Crosstalk
XTAL	Crystal
XTCLK	External Transmit clock
XTI	X/Open Transport Interface
XUMA	Expertensystem Umweltgefährlichkeit von Altlasten
XVE	X Visual Extensions
XVT	Extensible Virtual Toolkit
XWS	X-Window-System
YABA	Yet Another Bloody Acronym
YAFIYGI	You Asked For It, You Got It
YAODL	Yet Another Object Description Language
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
Z-CAV	Zoned Constant Angular Velocity
ZDL	Zero Delay Lockout
ZDR	Zone Density Recording
ZGDV	Zentrum für Graphische Datenverarbeitung
ZGS	Zeichengabesystem
ZIF	Zero Insertion Force
ZIP	Zone Information Protocol, Zone Improvement Plan

ZKA	Zentraler Kreditausschuß
ZOFF	Zeichenorientierte Fenster-Funktionen
ZSL	Zero Slot LAN
ZVE	Zählervergleichseinrichtung
ZVEI	Zentralverband der Elektrotechnik- und Elektronikindustrie
ZZF	Zentralamt für Zulassungen im Fernmeldewesen
ZZK	Zentraler Zeichenkanal (ISDN)

G Requests For Comment (RFCs)

Das Internet wird nicht durch Normen, sondern durch RFCs (Request For Comment) beschrieben, etwa 2000 an der Zahl. Wird ein RFC durch einen neueren abgelöst, bekommt der neue auch eine neue, höhere Nummer. Es gibt also keine Versionen oder Ausgaben wie bei den DIN-Normen. Einige RFCs sind zugleich FYIs (For Your Information) mit eigener Zählung. Diese enthalten einführende Informationen. Hier folgt eine Auswahl, nach der Nummer sortiert:

1	Host Software (1969)
681	Network Unix (1975)
814	Name, Addresses, Ports, and Routes (1982)
902	ARPA-Internet Protocol policy (1984)
1000	The Request For Comments Reference Guide (1987)
1034	Domain names – concepts and facilities (1987)
1087	Ethics and the Internet (1989)
1094	NFS: Network File System Protocol specification (1989)
1118	Hitchhiker's Guide to the Internet (1989)
1150	FYI on FYI: Introduction to the FYI notes (FYI 1) (1990)
1173	Responsibilities of Host and Network Managers (1991)
1175	FYI on Where to Start (FYI 3) (1990)
1178	Choosing a name for your computer (FYI 5) (1990)
1180	TCP/IP Tutorial (1991)
1198	FYI on the X window system (FYI 6) (1991)
1207	FYI on Questions and Answers (FYI 7) (1991)
1208	Glossary of Networking Terms (1991)
1244	Site Security Handbook (FYI 8) (1991)
1281	Guidelines for the secure operations of the Internet (1991)
1295	User bill of rights for entries and listing in the public directory (1992)
1296	Internet Growth (1981 – 1991) (1992)
1310	Internet standards process (1992)
1325	FYI on Questions and Answers (FYI 4) (1992)
1331	Point-to-Point Protocol (PPP) (1992)
1336	Who's who in the Internet (1992)
1361	Simple Network Time Protocol (1992)
1378	PPP AppleTalk Control Protocol (1992)
1392	Internet Users' Glossary (FYI 18) (1993)
1402	There is Gold in them thar Networks! (FYI 10) (1993)
1432	Recent Internet books (1993)
1436	Internet Gopher Protocol (1993)
1441	SMTP Introduction to version 2 of the Internet-standard Network Management Framework (1993)
1459	Internet Relay Chat Protocol 91993)
1460	Post Office Protocol - Version 3 (1993)

1462 FYI on What is the Internet (FYI 20) (1993)
1463 FYI on Introducing the Internet (FYI 19) (1993)
1466 Guidelines for Management of IP Address Space (1993)
1470 FYI on a Network Management Tool Catalog(FYI 2) (1993)
1475 TP/IX: The Next Internet (1993)
1491 A Survey of Advanced Usages of X.500 (FYI 21) (1993)
1500 Internet Official Protocol Standards (1993)
1501 OS/2 User Group (1993)
1506 A Tutorial on Gatewaying between X.400 and Internet Mail (1993)
1510 The Kerberos Network Authentication Service (1993)
1511 Common Authentication Technology Overview (1993)
1521 MIME (Multipurpose Internet Mail Extensions), Part One (1993)
1522 MIME (Multipurpose Internet Mail Extensions), Part One (1993)
1539 The Tao of IETF, A Guide for New Attendees of the Internet Engineering Task Force (FYI 17) (1993)
1591 Domain Name System Structure and Delegation (1994)

H 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)
Slip	Unterbekleidung (1) Schlupfschuh (0) Internet-Protokoll (2)
Diäresis	Durchfall (0) Diakritisches Zeichen (Umlaute) (2) Ernährungslehre (0)
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 159 Punkte: steigen Sie schnell in die EDV ein und überlassen Sie das Rechnen künftig dem Computer.
- 79 bis 159 Punkte: mit etwas Fleiß wird aus Ihnen ein EDV-Experte.
- 16 bis 78 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 (aber er versuchte es wenigstens).

I Literatur

Die Auswahl ist subjektiv und enthält Texte, die wir noch lesen wollen, schon haben oder sogar schon gelesen haben. Die angeführte Electronic Information ist auf `ftp.ciw.uni-karlsruhe.de` und anderen per Anonymous-FTP verfügbar. Die Referenz-Handbücher wurden schon erwähnt. Darüber hinaus verwenden wir (innerhalb der Gruppen nach Verfasser alphabetisch geordnet):

1. Literaturlisten

   - Newsgruppen:
     de.etc.lists (wechselnde Listen aus dem deutschsprachigen Raum)
     news.lists (internationale Listen)
     alt.books.technical
     biz.books.technical
     misc.books.technical
   - RFC 1175 (FYI 3): FYI on Where to Start –
     A Bibliography of Internetworking Information
     ftp.ciw: /pub/docs/net/rfc/rfc1175
     1990, 45 S., ASCII
     Empfehlungen und kurze Kommentare, Erklärungen
   - X Technical Bibliography, presented by The X Journal
     ftp.ciw: /pub/docs/xws/xbiblio.ps.gz
     1994, 22 S., Postscript
     Kurze Inhaltsangaben, teilweise kommentiert

   J. December Information Sources: The Internet and Computer-Mediated Communication
   ftp.ciw: /pub/docs/net/general/cmc.gz
   1994, 33 S., ASCII
   Hinweise, wo welche Informationen im Netz zu finden sind.

   S. Ko A Concise Guide to UNIX Books
   Netnews: misc.books.technical oder comp.unix.questions
   ftp.ciw: /pub/docs/unix/unix-books
   1993, 22 S., ASCII
   Empfehlungen und kurze Kommentare

   D. A. Lamb Software Engineering Readings
   Netnews: comp.software-eng
   ftp.ciw: /pub/docs/misc/sw-engng-reading
   1994, 10 S., ASCII Teilweise kommentiert

   R. E. Maas MaasInfo.DocIndex
   ftp.ciw: /pub/docs/net/general/maasinfo-idx

1994, 20 S., ASCII
Bibliografie von rund 100 On-line-Texten zum Internet

J. Quarterman RFC 1432: Recent Internet Books
ftp.ciw: /pub/docs/net/rfc/rfc1432
1993, 15 S., ASCII
Empfehlungen und kurze Kommentare

C. Spurgeon Network Reading List: TCP/IP, Unix and Ethernet
ftp.ciw: /pub/docs/net/general/reading-list.ps.gz und .txt.gz
1993, ca. 50 S., Postscript und ASCII
Ausführliche Kommentare und Hinweise

M. Wright Yet Another book List (YABL)
ftp.ciw: /pub/docs/misc/yabl.gz
1993, ca. 100 S., ASCII
Tabellarisch, kurze Kommentare

2. Lexika, Glossare, Wörterbücher

- Newsgruppen:
  news.answers
  de.etc.lists
  news.lists
- RFC 1392 (FYI 18): Internet Users' Glossary
  ftp.ciw: /pub/docs/net/rfc/rfc1392
  1993, 53 S.
- Duden Informatik
  Dudenverlag, Mannheim, 1993, 800 S., 42 DM
  Nachschlagewerk, sorgfältig gemacht, theorielastig,
  Begriffe wie Ethernet, LAN, SQL, Internet fehlen.
- Fachausdrücke der Informationsverarbeitung Englisch – Deutsch, Deutsch – Englisch
  IBM Deutschland, Form-Nr. Q12-1044, 1698 S., 113 DM
  Wörterbuch und Glossar

W. Alex Abkürzungs-Liste ABKLEX (Informatik, Telekommunikation)
ftp.ciw: /pub/misc/abklex/
Abkürzungsliste aus diesem Text, aktuelle Arbeitsfassung

V. Anastasio Wörterbuch der Informatik Deutsch – Englisch – Französisch – Italienisch – Spanisch
VDI-Verlag, Düsseldorf, 1990, 400 S., 128 DM

A. E. Cawkell Encyclopedic Dictionary of Information Technology and Systems
Saur, München, 1993, 350 S., 190 DM

F. Krückeberg, O. Spaniol Lexikon Informatik und Kommunikationstechnik
VDI-Verlag, Düsseldorf, 1990, 693 S., 168 DM

M. Peschke, P. Wennrich Encyclopedic Dictionary of Electronics, Electrical Engineering and Information Processing
Englisch – Deutsch, Deutsch – Englisch
Saur, München, 1990, 8 Bände zu je 400 S., 1920 DM

A. Ralston, E. D. Reilly Encyclopedia of Computer Science
Chapman + Hall, London, 1993, 1558 S., 60 £
Ausführliche Erläuterungen

E. S. Raymond The New Hacker's Dictionary
The MIT Press, Cambridge, 1993, 505 S., 40 DM
Siehe auch ftp.ciw: /pub/docs/net/general/jarg300.txt.gz
Begriffe aus dem Netz, die nicht im Duden stehen

H.-J. Schneider Lexikon der Informatik und Datenverarbeitung
Oldenbourg, München, 1991, 989 S., 128 DM
Ethernet, SQL stehen darin, Internet nicht.

P. Wennrich International Dictionary of Abbreviations and Acronyms of Electronics, Electrical Engineering, Computer Technology and Information Processing
Saur, München, 1992, 2 Bände, 960 S., 336 DM

3. Informatik

– Newsgruppen:
comp.* (alles, was mit Computer Science zu tun hat, mehrere hundert Untergruppen)
de.comp.* (dito, deutschsprachig)
alt.comp.*
biz.comp.*
maus.sci.informatik
zer.z-netz.wissenschaft.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., 50 DM
Digitale Schaltungen, Rechnerarchitektur, Betriebssysteme am Beispiel von UNIX

L. Goldschlager, A. Lister Informatik
Hanser und Prentice-Hall, München, 1990, 366 S., 40 DM
Einführung, ähnlich wie Bauer + Goos

D. E. Knuth The Art of Computer Programming, 3 Bände
Addison-Wesley, zusammen 330 DM
Klassiker, stellenweise mathematisch, 7 Bände geplant

W. Schiffmann, R. Schmitz Technische Informatik
Springer, Berlin, 1993/94, 1. Teil Grundlagen der digitalen Elektronik, 282 S., 38 DM; 2. Teil Grundlagen der Computertechnik, 283 S., 42 DM

U. Schöning Theoretische Informatik kurz gefaßt
BI-Wissenschaftsverlag, Mannheim, 1992, 188 S., 20 DM
Automaten, Formale Sprachen, Berechenbarkeit, Komplexität

K. W. Wagner Einführung in die Theoretische Informatik
Springer, Berlin, 1994, 238 S.,

H. Waldschmidt Informatik für Ingenieure
Oldenbourg, München, 1987, 258 S., 40 DM
Algorithmen und Programme, Programmierfehler, Ergänzung zu einem Programmierkurs

4. Algorithmen, Numerische Mathematik

– Newsgruppen:
sci.math.*
zer.z-netz.wissenschaft.mathematik

G. Engeln-Müllges, F. Reutter Formelsammlung zur Numerischen Mathematik mit C-Programmen
BI-Wissenschaftsverlag, Mannheim, 1990, 744 S., 88 DM
Algorithmen und Formeln der Numerischen Mathematik samt C-Programmen. Auch für FORTRAN, PASCAL, BASIC und MODULA erhältlich

E. Horowitz, S. Sahni Algorithmen
Springer, Berlin, 1981, 770 S., 116 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

F. Stummel, K. Hainer Praktische Mathematik
Teubner, Stuttgart, 1982, 367 S., 40 DM

N. Wirth Algorithmen und Datenstrukturen
Teubner, Stuttgart, 1983, 320 S., 42 DM
Viel zu Datenstrukturen, weniger zu Algorithmen

5. Betriebssysteme

- Newsgruppen:
  comp.os.*
  de.comp.os.*
- 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 im Verzeichnis /pub/os2/misc
  auch auf ftp.uni-stuttgart.de in /pub/soft/os2/info/redbooks

L. Bic, A. C. Shaw Betriebssysteme
Hanser, München, 1990, 420 S., 58 DM
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

6. UNIX allgemein

- Newsgruppen:
  comp.unix.*
  comp.sources.unix
  comp.std.unix
  de.comp.os.unix
  fr.comp.os.unix
  alt.unix.wizards
  cern.unix
  fido.ger.unix
  fido.unix
  maus.os.unix

maus.os.unix
zer.z-netz.unix
zer.z-netz.rechner.unix.*

M. J. Bach Design of the UNIX Operating System
Prentice-Hall, London, 1987, 512 S., 52 US-$
Filesystem und Prozesse, wenig zur Shell

S. R. Bourne Das UNIX System V (The UNIX V Environment)
Addison-Wesley, Bonn, 1988, 464 S., 62 DM
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

J. Gulbins UNIX
Springer, Berlin, 3. Aufl. 1988, 773 S., 84 DM
Benutzung von UNIX, ausführlich, geht in die Einzelheiten

H. Hahn A Student's Guide to UNIX
McGraw-Hill, New York, 1993, 633 S., 66 DM
Gute Ergänzung zu unserem Werk, ohne C, mit Internet-Diensten

M. Harlander Introduction to UNIX
ftp.ciw: /pub/docs/unix/unix-int.ps.gz
1991, 51 S., Postscript

J. A. Illik (siehe unter Programmieren in C)

B. W. Kernighan, P. J. Plauger Software Tools
Addison-Wesley, Reading, 1976, 338 S., 38 US-$
Grundgedanken einiger UNIX-Werkzeuge, Programmierstil

B. W. Kernighan, R. Pike Der UNIX-Werkzeugkasten
Hanser, München, 1986, 402 S., 76 DM
Gebrauch der UNIX-Kommandos, fast nichts zum `vi(1)`

D. G. Korn, M. I. Bolsky The Kornshell, Command and Programming Language
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, 1986, 224 S., 47 US-$
Beschreibung aller UNIX System Calls

A. T. Schreiner Professor Schreiners UNIX-Sprechstunde
Hanser, München, 1987, 316 S., 64 DM
Shellscripts und kurze C-Programme für verschiedene Zwecke

R. M. Stallman The GNU Manifesto
ftp.ciw: /pub/docs/misc/manifest-gnu
1985, 8 S., ASCII
Ziele des GNU-Projekts

W. R. Stevens Advanced Programming in the UNIX Environment
Addison-Wesley, Reading, 1992, 744 S., 110 DM
Ähnlich wie Rochkind

S. Strobel, T. Uhl LINUX - vom PC zur Workstation
Springer, Berlin, 1994, 238 S., 38 DM

L. Wirzenius, M. Welsh LINUX Information Sheet
Netnews: comp.os.linux
ftp.ciw: /pub/docs/unix/linux-info
1993, 6. S., ASCII
Anfangsinformation zu LINUX, was und woher.

C. Zimmermann, A, W. Kraas MACH
Springer, Berlin, 1992, 210 S., 78 DM
Konzepte und Programmierung von MACH

7. UNIX Einzelthemen

– Newsgruppen:
comp.unix.*

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. Cameron, B. Rosenblatt Learning GNU Emacs
O'Reilly, Sebastopol, 1991, 442 S., 21 £

F. da Cruz, C. Gianone C-Kermit
Heise, Hannover, 1994, 650 S., 90 DM
Kermit-Terminalemulation und -Fileübertragung

I. F. Darwin Checking C Programs with lint
O'Reilly, Sebastopol, 1988, 82 S., 10 £

B. Goodheart UNIX Curses Explained
Prentice-Hall, Englewood-Cliffs, 1991, 287 S., ca. 80 DM
Einzelheiten zu `curses(3)` und `terminfo(4)`

L. Lamb Learning the vi Editor
O'Reilly, Sebastopol, 1990, 192 S., 17 £

E. Nemeth, G. Snyder, S. Seebass UNIX System Administration Handbook
Prentice-Hall, Englewood-Cliffs, 1990, 624 S., 47 US-$
Empfehlung eines Stuttgarter Kollegen

A. Oram, S. Talbott Managing Projects with make
O'Reilly, Sebastopol, 1993, 149 S., 35 DM

G. Staubach UNIX-Werkzeuge zur Textmusterverarbeitung
Springer, Berlin, 1989, 157 S., 38 DM
`awk(1)`, `lex(1)` und `yacc(1)`

W. R. Stevens UNIX Network Programming
Prentice Hall, Englewood Cliffs, 1990, 772 S., 60 US-$
C-Programme für Clients und Server der Netzdienste

J. Strang u. a. termcap & terminfo
O'Reilly, Sebastopol, 1988, 270 S., 17 £

I. A. Taylor Taylor UUCP
ftp.ciw: /pub/docs/unix/uucp.ps.gz
1993, 93 S., Postscript

L. Wall, R. Schwartz Programming Perl
O'Reilly, Sebastopol, 1991, 482 S., 22 £

8. Grafik

– Newsgruppen:
comp.graphics.*
alt.graphics.*
fido.ger.grafik
zer.z-netz.rechner.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, 1994, 500 S., 98 DM
GKS für FORTRAN-Programmierer

J. D. Foley Computer Graphics – Principles and Practice
Addison-Wesley, Reading, 1992, 1200 S., 83 US-$
Standardwerk zur Computer-Grafik

T. Gaskins The PHIGS Programming Manual
O'Reilly, Sebastopol, 1992, 908 S., 102 DM
Lehrbuch und Nachschlagewerk, auch 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, 354 S., 63 US-$

H. Kopp Graphische Datenverarbeitung
Hanser, München, 1989, 211 S., 40 DM
mathematische Methoden, Algorithmen, GKS

L. Kosko PHIGS Reference Manual
O'Reilly, Sebastopol, 1992, 1000 S., 40 US-$

9. Netze (TCP/IP, OSI, Internet)

– Newsgruppen:
comp.infosystems.*
comp.internet.*
comp.protocols.*
alt.best.of.internet
alt.bbs.internet
alt.internet.*
de.comm.internet
de.comp.infosystems
fr.comp.infosystemes
fido.ger.internet
vmsnet.infosystems.*

– Internet Resources Guide
NSF Network Service Center, Cambridge, 1993
ftp.ciw: /pub/docs/net/general/resource-guide-help
ftp.ciw: /pub/docs/net/general/resource-guide.ps.tar.gz
ftp.ciw: /pub/docs/net/general/resource-guide.txt.tar.gz
Beschreibung der Informationsquellen im Internet

– Das Telekom-Buch 93/94
Deutsche Bundespost Telekom, Bonn, 1993, 320 S., 10 DM
Übersicht über die Netzdienste der Bundespost, leichtverständlich

S. Carl-Mitchell, J. S. Quarterman Practical Internetworking
with TCP/IP and UNIX
Addison-Wesley, Reading, 1993, 432 S., 52 US-$

D. E. Comer Internetworking with TCP/IP (4 Bände)
Prentice-Hall, Englewood Cliffs, I. Band 1991, 550 S., 90 DM;

II. Band 1991, 530 S., 88 DM; IIIa. Band (BSD) 1993, 500 S., 86 DM; IIIb. Band (AT&T) 1994, 510 S., 90 DM
Prinzipien, Protokolle und Architektur des Internet

EARN Guide to Network Resource Tools
ftp.ciw: /pub/docs/net/general/nettools.ps.gz
1993, 70 S., Postscript
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

A. Gaffin, J. Heitkötter Big Dummy's Guide to the Internet
ftp.ciw: /pub/docs/net/general/bdgtti2.ps.gz
1993, 220 S., Postscript, andere Formate im Netz
Einführung in die Dienste des Internet

H. Hahn, R. Stout 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, Standardwerk

Ch. Hedrick Introduction to the Internet Protocols
ftp.ciw: /pub/docs/net/general/tcp-ip-intro.ps.gz und .doc.gz
1988, 20 S., ASCII und Postscript

Ch. Hedrick Introduction to Administration of an Internet-based Local Network
ftp.ciw: /pub/docs/net/general/tcp-ip-admin.ps.gz und .doc.gz
1988, 39 S., ASCII und Postscript
Adressen, Routing, Netztopologie im Internet

K. Hughes Entering the World-Wide Web: A Guide to Cyberspace
ftp.ciw: /pub/docs/net/www/hughes-guide.ps.gz
1993, 20 S., Postscript
Erklärungen, Entstehung, Glossar

B. P. Kehoe Zen and the Art of the Internet
ftp.ciw: /pub/docs/net/general/zen.ps.gz
1992, 100 S., Postscript
Einführung in die Dienste des Internet

E. Krol The Hitchhikers Guide to the Internet
ftp.ciw: /pub/docs/net/general/hitchhg.txt
1987, 16 S., ASCII
Erklärung einiger Begriffe aus dem Internet

E. Krol The Whole Internet
O'Reilly, Sebastopol, 1992, 376 S., 25 US-$

J. Lammarsch, H. Steenweg Internet & Co
Addison-Wesley, Bonn, 1994, 218 S., 60 DM

T. L. LaQuey User's Directory of Computer Networks
Digital Press, Bedford, 1990, 653 S.,
Ins einzelne gehende Informationen über zahlreiche Netze

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

D. Raggett HTML+ (Hypertext Markup Language)
ftp.ciw: /pub/docs/net/www/htmlplus.ps.gz
1993, 34 S., Postscript
Vorschlag für den Hypertext-RFC

M. T. Rose The Open Book
Prentice-Hall, Englewood Cliffs, 1990, 682 S., 64 US-$
OSI-Protokolle, Vergleich mit TCP/IP

M. Scheller u. a. Internet: Werkzeuge und Dienste
Springer, Berlin, 1994, 280 S., 49 DM

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

– Newsgruppen:
comp.windows.x.*
fr.comp.windows.x11

– 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, Bedford, 1993, 400 S., 22 £

Zusammenfassung aus den Referenz-Handbüchern

11. Programmieren allgemein

- Newsgruppen:
  comp.programming
  comp.unix.programmer
  comp.lang.*
  comp.software.*
  comp.software-eng
  comp.compilers
  de.comp.lang.*
  fido.ger.compilerbau
  maus.comp.compiler

A. V. Aho u. a. Compilers, Principles, Techniques and Tools
Addison-Wesley, Reading, 1986, 796 S., 78 DM
Zum tieferen Verständnis von Programmiersprachen

B. Beizer Software Testing Techniques
Van Nostrand-Reinhold, 1990, 503 S., 43 US-$

N. Ford Programmer's Guide
ftp.ciw: /pub/docs/misc/pguide.txt
1989, 31 S., ASCII
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 praktischer Bedeutung.

E. Horowitz Fundamentals of Programming Languages
Springer, Berlin, 1984, 446 S., ??? DM
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, 1987, 360 S., 90 DM

S. Pfleeger Software Engineering: The Production of Quality
Software
Macmillan, 1991, 480 S., 22 £(Studentenausgabe)
Empfehlung aus dem Netz

R. W. Sebesta Concepts of Programming Languages
Benjamin/Cummings, Redwood City, 1993, 560 S., 65 US-$
ähnlich wie Horowitz

I. Sommerville Software Engineering
Addison-Wesley, Reading, 1992, 688 S., 52 US-$
Wie man ein Programmierprojekt organisiert;
Werkzeuge, Methoden; sprachenunabhängig

N. Wirth Systematisches Programmieren
Teubner, Stuttgart, 1993, 160 S., 27 DM
Allgemeine Einführung ins Programmieren, PASCAL-nahe

A. Zeidler, R. Zellner Software-Ergonomie
Oldenbourg, München, 1994, 292 S., 68 DM

12. Programmieren in C

- Newsgruppen:
  comp.lang.c
  comp.std.c
  de.comp.lang.c
  maus.lang.c
- Microsoft Quick-C- und C-6.0-Handbücher
  mehrere Bände bzw. Ordner

C. Brinkschulte Macintosh Programmieren in C
Springer, Berlin, 1992, 404 S., 89 DM

H. M. Deitel, P. J. Deitel C How to Program
Prentice Hall, Englewood Cliffs, 1994, 926 S., 74 DM
Enthält auch C++. Ausgeprägtes Lehrbuch.

A. R. Feuer Das C-Puzzle-Buch
Hanser Verlag, München, 1991, 196 S., 38 DM
Kleine, feine Aufgaben zu C-Themen

O. Hartwig C Referenz-Handbuch
Sybex, Düsseldorf, 1987, 432 S., 54 DM (vergriffen?)
Nachschlagewerk für K&R-C und ANSI-C

R. House Beginning with C
An Introduction to Professional Programming
International Thomson Publishing, Australien, 1994, 568 S., 64 DM
Ausgeprägter Lehrbuch-Charakter, ANSI-C, vorbereitend auf C++

J. A. Illik Programmieren in C unter UNIX
Sybex, Düsseldorf, 1992, 750 S., 89 DM
Lehrbuch, C und UNIX mit Schwerpunkt Programmieren

R. Jones, I. Steart The Art of C Programming
Springer, Berlin, 1987, 186 S., 52 DM

B. W. Kernighan, D. M. Ritchie The C Programming Language
Deutsche Übersetzung: Programmieren in C
Zweite Ausgabe, ANSI C

Hanser Verlag, München, 1990, 283 S., 56 DM
Standardwerk zur Programmiersprache C, Lehrbuch

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

R. Ward Debugging C
Addison-Wesley, Bonn, 1988, 322 S., 68 DM
Systematische Fehlersuche, hauptsächlich in C-Programmen

13. Objektorientiertes Programmieren

– Newsgruppen:
comp.lang.object
comp.lang.c++
comp.lang.objective-c
comp.std.c++
de.comp.lang.c++

G. Booch Object-Oriented Analysis and Design with Applications
Benjamin + Cummings, Redwood City, 1994, 590 S., 112 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++

B. J. Cox, A. J. Novobilski Object-Oriented Programming
Addison-Wesley, Reading, 1991, 270 S., 76 DM
Objective C

R. Klatte u. a. C-XSC
Springer, Berlin, 1993, 269 S., 74 DM
auch auf englisch erhältlich
C++-Klassenbibliothek für wissenschaftliches Rechnen

B. Stroustrup The C++ Programming Language
Addison-Wesley, Reading, 2. Aufl. 1991, 669 S., 76 DM
Lehrbuch, Klassiker für C++

14. Programmieren in anderen Sprachen

– Newsgruppen:
comp.lang.*
de.comp.lang.*

W. Gehrke FORTRAN 90 Referenz-Handbuch
Hanser, München, 1991, 964 S., 168 DM
Beschreibung des Standards, kein Lehrbuch für Anfänger

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, BI-Hochschultaschenbücher Band 609,
Bibliographisches Institut, Mannheim, 1983, 467 S., 30 DM
Band 7: Übungen und Tests in FORTRAN 77, Band 617
Bibliographisches Institut, Mannheim, 1984, 329 S. 30 DM
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, 1992, 280 S., 28 DM
Lehrbuch für Anfänger

P. H. Winston, B. K. P. Horn LISP
Addison-Wesley, Reading, 1993, 611 S., 75 DM

15. Anwendungen

– Newsgruppen:
comp.theory.info-retrieval
comp.databases.*

– Guide to Commands
STN International c/o FIZ Karlsruhe, 1991, 314 S.
Beschreibung der Retrieval-Sprache Messenger

M. Gossens u. a. The LaTeX-Companion
Addison-Wesley, Reading, 1994, 530 S., 40 US-$

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., 78 DM
Standardwerk zu LaTeX

P. Luidl Typographie
Schlüter, Hannover, 1991, 146 S., 79 DM
Herkunft, Aufbau und Anwendung von Druckschriften, Layout

H. Partl u. a. LaTeX-Kurzbeschreibung
ftp.ciw: /pub/docs/latex/lkurz.ps.gz und lkurz.tar.gz
1990, 46 S., Postscript und LaTeX-Quellen
Einführung, mit deutschsprachigen Besonderheiten (Umlaute)

E. D. Stiebner Handbuch der Drucktechnik
Bruckmann, München, 1992, 362 S., 98 DM

F. W. Weitershaus Duden Satz- und Korrekturanweisungen
Dudenverlag, Mannheim, 1980, 268 S., 17 DM (vergriffen?)
Hilfe beim Herstellen von Druckvorlagen

16. Sicherheit

- Newsgruppen:
  comp.security.*
  alt.security.*
  de.comp.security
  sci.crypt
  comp.virus
  alt.comp.virus
  fido.ger.virus
  maus.comp.virus
- RFC 1244 (FYI 8): Site Security Handbook
  ftp.ciw: /pub/docs/net/rfc/rfc1244
  1991, 101 S., ASCII
  Sicherheits-Ratgeber für Internet-Benutzer
- Department of Defense Trusted Computer Systems
  Evaluation Criteria (Orange Book)
  ftp.ciw: /pub/docs/net/secur/orange-book.gz
  1985, 120 S., ASCII
  wird abgelöst durch:
  Federal Criteria for Information Technology Security
  ftp.ciw: /pub/docs/net/secur/fcvol1.ps.gz und fcvol2.ps.gz
  1992, 2 Bände mit zusammen 500 S., Postscript
  Die amtlichen amerikanischen Sicherheitsvorschriften

F. L. Bauer Kryptologie
Springer, Berlin, 1994, 369 S., 48 DM

R. L. Brand Coping with the Threat of Computer Security Incidents
A Primer from Prevention through Recovery
ftp.ciw: /pub/docs/net/secur/primer.ps.gz
1990, 44 S., Postscript

D. A. Curry Improving the Security of Your UNIX System
ftp.ciw: /pub/docs/net/secur/secdoc.ps.gz

1990, 50 S., Postscript
Hilfe für UNIX-System-Manager, mit Checkliste

H.-L. Drews u. a. Lexikon Datenschutz und Informationssicherheit
Siemens AG, Berlin, 1993, 353 S., 110 DM (!)
Technische und rechtliche Themen; zu teuer

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

17. Geschichte der Informatik

- Newsgruppen:
  comp.society.folklore
  alt.folklore.computers
  de.alt.folklore.computer
- Kleine Chronik der IBM Deutschland
  1910 – 1979, Form-Nr. D12-0017, 138 S.
  1980 – 1991, Form-Nr. D12-0046, 82 S.
  Reihe: Über das Unternehmen
  IBM Deutschland
- Die Geschichte der maschinellen Datenverarbeitung Band 1
  Reihe: Enzyklopädie der Informationsverarbeitung
  IBM Deutschland, 228 S., Form-Nr. D12-0028
- 100 Jahre Datenverarbeitung Band 2
  Reihe: Über die Informationsverarbeitung
  IBM Deutschland, 262 S., Form-Nr. D12-0040

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., 6 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 (vergriffen)

B. Sterling A short history of the Internet
ftp.ciw: /pub/docs/history/origins
1993, 6 S., ASCII

K. Zuse Der Computer - Mein Lebenswerk
Springer, Berlin, 3. Aufl. 1993, 220 S., 58 DM
Autobiografie Konrad Zuses

18. Computerrecht

- Newsgruppen:
  comp.society.privacy
  comp.privacy
  alt.privacy
  comp.patents
  maus.recht
  cl.datenschutz.*
  de.soc.recht
  fido.ger.recht
  zer.z-netz.datenschutz.*
  zer.rechtswesen.*
  zer.z-netz.jura.*
- Computerrecht (Beck-Texte)
  Beck, München, 1994, 13 DM

U. Dammann, S. Simitis Bundesdatenschutzgesetz
Nomos Verlag, Baden-Baden, 1993, 606 S., 38 DM
BDSG mit Landesdatenschutzgesetzen und Internationalen Vorschriften; Texte, kein Kommentar

G. v. Gravenreuth Computerrecht von A – Z (Beck Rechtsberater)
Beck, München, 1992, 17 DM

H. Hubmann, M. Rehbinder Urheber- und Verlagsrecht
Beck, München, 1991, 319 S., 40 DM

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

19. Philosophische Feigenblätter

- Newsgruppen:
  comp.ai.philosophy
  sci.philosophy.tech

J. Ladd Computer, Informationen und Verantwortung
in: Wissenschaft und Ethik, herausgegeben von H. Lenk
Reclam-Band 8698, Ph. Reclam, Stuttgart, 15 DM

H. Lenk Chancen und Probleme der Mikroelektronik
und: Können Informationssysteme moralisch verantwortlich sein?

in: Hans Lenk, Macht und Machbarkeit der Technik
Reclam-Band 8989, Ph. Reclam, Stuttgart, 1994, 152 S., 6 DM

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. (vergriffen?)

J. Weizenbaum Die Macht der Computer und die Ohnmacht der Vernunft (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

20. Nicht einzuordnen

- Newsgruppen:
  alt.fan.hofstadter

D. R. Hofstadter Gödel, Escher, Bach - ein Endloses Geflochtenes Band
dtv/Klett-Cotta, München, 1992, 844 S., 30 DM

R. Kurzweil KI - Das Zeitalter der künstlichen Intelligenz
Hanser, München, 1993, 552 S., 98 DM (vergriffen?)
(im Original: The Age of Intelligent Machines
MIT Press, Cambridge, 1992, 565 S., 63 DM)
Beide Bücher haben nichts unmittelbar mit UNIX, C oder dem Internet zu tun, aber viel mit unserem Verhältnis zum Computer und zu unserem eigenen Denken.

21. Zeitschriften

- 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.
- Dr. Dobb's Journal
  Miller Freeman Inc., San Mateo, USA, monatlich, ca. 180 S.
  Software Tools for the Professional Programmer; viel C und C++

- unix/mail
  Hanser Verlag, München, sechsmal im Jahr, ca. 70 S.
  für Entwickler und Benutzer
- UNIX Open
  Aktuelles Wissen Verlagsgesellchaft 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 Netz-Anschriften einiger Verlage:

- Verlag Heinz Heise (Redaktion IX), Hannover
  E-Mail: post@ix.de
  WWW (URL): http://www.ix.de
- O'Reilly, Sebastopol, USA
  E-Mail: nuts@ora.com
  Gopher: gopher.ora.com
- Springer, Berlin, Göttingen, Heidelberg usw.
  E-Mail: svserv@vax.ntp.springer.de
  aFTP: trick.ntp.springer.de
  Gopher: trick.ntp.springer.de
  WWW: http://www.springer.de

J Zeittafel

Diese Übersicht ist auf die Karlsruher Verhältnisse zugeschnitten. Ausführlichere Angaben sind den im Anhang I *Literatur* in Abschnitt *Geschichte* aufgeführten Werken zu entnehmen. Die meisten Errungenschaften entstanden nicht zu einem Zeitpunkt, sondern entwickelten sich über manchmal lange Zeitspannen, so daß vor viele Jahreszahlen *um etwa* zu setzen ist. Das Deutsche Museum in München zeigt in den Abteilungen *Informatik* und *Telekommunikation* einige der hier genannten Maschinen.

ca. 10^8 v. Chr.	Der beliebte Tyrannosaurus rex hatte zwei Finger an jeder Hand und rechnete vermutlich im Dualsystem, wenn überhaupt.
ca. 2000 v. Chr.	Die Babylonier verwenden für besondere Aufgaben ein gemischtes Stellenwertsystem zur Basis 60.
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).
600	Die Inder entwickeln das heute übliche reine Stellenwertsystem. Die Null ist jedoch älter. Etwa gleichzeitig entwicklen die Mayas in Mittelamerika ein Stellenwertsystem zur Basis 20.
1200	Leonardo von Pisa, genannt Fibonacci, setzt sich für die Einführung des indisch-arabischen Systems im Abendland ein.
1550	Die europäischen Rechenmeister verwenden sowohl die römische wie die indisch-arabische Schreibweise.
1617	John Napier erfindet die Rechenknochen (Napier's Bones).
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 dualen Darstellung von Zahlen. In der Folgezeit technische Verbesserungen an vielen Stellen in Europa.
1801	Joseph Marie Jacquard erfindet die Lochkarte und steuert Webstühle damit.
1821	Charles Babbage stellt der Royal Astronomical Society eine programmierbare mechanische Rechenmaschine vor, die jedoch keinen wirtschaftlichen Erfolg hat. Er denkt auch an das Spielen von Schach oder Tic-tac-toe auf Maschinen.
1840	Samuel Morse entwickelt einen aus zwei Zeichen plus Pausen bestehenden Telegrafencode, der die Buchstaben entsprechend ihrer Häufigkeit codiert.
1847	George Boole entwickelt die symbolische Logik.

1890 **Herman Hollerith** erfindet die Lochkartenmaschine und setzt sie bei einer Volkszählung in den USA ein. Das ist der Anfang von IBM.

1891 Die Stanford-Universität in Kalifornien wird gegründet.

1894 **Otto Luegers** *Lexikon der gesamten Technik* führt unter dem Stichwort *Elektrizität* als Halbleiter Aether, Alkohol, Holz und Papier auf.

1910 Gründung der Deutschen Hollerith Maschinen GmbH, Berlin, der Vorläuferin der IBM Deutschland.

1924 Aus der Tabulating Machine Company von Herman Hollerith, später in Computing-Tabulating-Recording Company umbenannt, wird die International Business Machines (IBM). **Eugen Nesper** schreibt in seinem Buch *Der Radio-Amateur*, Verlag Julius Springer, Berlin, jeder schlechte Kontakt habe gleichrichtende Eigenschaften, ein Golddraht auf einem Siliziumkristall sei aber besonders gut als Kristalldetektor geeignet; eine heiße Spur.

1937 **Alan Turing** veröffentlicht sein Gedankenmodell eines Computers.

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.

1942 Die Purdue University beginnt mit der Halbleiterforschung und untersucht Germaniumkristalle.

1944 Die Zuse Z4 wird fertig (2200 Relais, mechanischer Speicher). Sie arbeitet von 1950 bis 1960 in der Schweiz.

1945 **Konrad Zuse** entwickelt den Plankalkül, die erste höhere Programmiersprache. **William Bradford Shockley** startet ein Forschungsprojekt zur Halbleiterphysik in den Bell-Labs.

1946 **John von Neumann** veröffentlicht sein Konzept eines Computers. **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 t, ist 5,5 m hoch und 24 m lang, braucht für eine Addition 0,2 ms, ist an der Entwicklung der Wasserstoffbombe beteiligt und arbeitet bis 1955. Sie ist der Urahne der UNIVAC.

1948 **Claude E. Shannon** begründet die Informationstheorie. **John Bardeen**, **Walter Houser Brattain** und **William Bradford Shockley** entwickeln in den Bell-Labs den Transistor, der 10 Jahre später die Vakuumröhre ablöst.

1949 Erster Schachcomputer: Manchester MADM.

1952 IBM bringt ihre erste elektronische Datenverarbeitungsanlage, die IBM 701, heraus. Vorschläge für integrierte Schaltkreise, [illegible] 1954 Remington Rand bringt die erste UNIVAC heraus, IBM die 650. Silizium beginnt, das Germanium zu verdrängen.

1955 IBM entwickelt die erste höhere Programmiersprache: FORTRAN (Formula Translator) und verwendet Transistoren.

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 t. 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 Angew. Mathematik) eine Vorlesung *Programmieren mathematischer und technischer Probleme für die elektronische Rechenmaschine Z22*. 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. JACK ST. CLAIR KILBY baut bei Texas Instruments den ersten IC.

1960 Programmieren steht noch in keinem Studienplan, sondern ist freiwillig. Die Karlsruher Z22 läuft Tag und Nacht. Die Programmiersprache COBOL wird veröffentlicht. Ein Computerspiel namens *Spacewar* läuft auf einer PDP-1 im MIT.

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. An der Purdue University wird die erste Fakultät für Informatik (Department of Computer Science) gegründet. Texas Instruments und Fairchild nehmen die Serienproduktion von ICs (Chips) auf.

1963 Weltweit arbeiten etwa 16.500 Computer.

1964 Die Programmiersprache BASIC erscheint. In den USA wird der Begriff *Computer Science* geprägt. IBM legt das Byte zu 8 Bits fest (IBM 360). Ein Chip enthält auf 0,5 cm^2 10 Transistoren.

1966 Die TH Karlsruhe erhält eine Electrologica X 8, die bis 1973 betrieben wird. Gründung des Karlsruher Rechenzentrums.

1967 Erster elektronischer Taschenrechner (Texas Instruments).

1968 Die Programmiersprache PASCAL kommt heraus. Der Begriff Informatik wird nach französischem Vorbild geprägt.

	Die Firma Intel wird gegründet.
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. In den Bell Labs UNIX in Assembler auf einer DEC PDP 7. Beginn des ARPANET-Projektes und damit der TCP/IP-Protokolle.
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 Mikroprozessor (Intel 4004).
1972	IBM entwickelt das Konzept des virtuellen Speichers. Xerox, DEC und Intel entwickeln den Ethernet-Standard. Das ARPANET wird der Öffentlichkeit vorgestellt. Ein Student namens STEPHAN G. WOZNIAK lötet sich einen Computer zusammen, der den Smoke-Test nicht übersteht. In der Bundesrepublik arbeiten rund 8.200 Computer.
1974	Der erste programmierbare Taschenrechner kommt auf den Markt (Hewlett-Packard 65), Preis ca. 2500 DM. HMI-NET am Hahn-Meitner-Institut in Berlin.
1975	UNIX wird veröffentlicht, Beginn der BSD-Entwicklung. Die Zeitschrift *Byte* wird gegründet. Erste, mäßig erfolgreiche Personal Computer.
1976	STEVEN P. JOBS und STEPHAN G. WOZNIAK gründen die Firma Apple und bauen den Apple I. Er kostet 666,66 Dollar.
1978	In der Bundesrepublik arbeiten rund 170.000 Computer. Der Commodore PET – ein Vorläufer des C64 – kommt heraus.
1979	Faxdienst in Deutschland eingeführt. Die Zusammenarbeit von Apple mit Rank Xerox führt zur Apple Lisa, ein Mißerfolg, aber der Wegbereiter für den Macintosh.
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. Die Firma SUN wird gegründet, entscheidet sich für UNIX und baut die ersten Workstations.
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.
Entwicklung des X-Window-Ssystems am MIT.

1985 MS-Windows 1.0, IBM 3090 und IBM Token Ring Netz.
IBM 3090 angekündigt.

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 erhält Lehrauftrag *Einführung in UNIX*.
Die *Toten Hosen* spielen zur ersten Vorlesung den Song: *Hey! Hier kommt Alex, Vorhang auf*. Es sind aber nur wenige Hörer dabei.

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 (CD-ROM, WORM).
UNIX System V Version 4.
Die mittlere Computerdichte in technisch orientierten Instituten und Familien erreicht 1 pro Mitglied.

1991 Das UNIX-System OSF/1 mit dem Mach-Kernel der Carnegie-Mellon-Universität kommt heraus.
Anfänge von LINUX, einem freien UNIX aus Finnland.
Der Vektorrechner im Rechenzentrum wird erweitert auf den Typ S600/20. MS-DOS 5.0 für PCs.
IBM, Apple und Motorola kooperieren mit dem Ziel, einen Power PC zu entwickeln.

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 MS-DOS Version 6.0. Microsoft kündigt Windows-NT 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.

1994 Weltweit 10 Mio. installierte UNIX-Systeme prognostiziert.
Das Internet umfaßt etwa 2 Mio. Knoten und 20 Mio. Benutzer.

Wie man sieht, mußten viele Dinge erfunden und entwickelt werden, damit heute die Kinder MUDs oder Nethack spielen können.

> 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

Sach- und Namensverzeichnis

Einige Begriffe sind unter ihren Oberbegriffen zu finden, beispielsweise Gerätefile unter File oder Bourne-Shell unter Shell. Verweise (*s.* ...) zeigen entweder auf ein von uns bevorzugtes Synonym, auf einen Oberbegriff oder auf die deutsche Übersetzung eines englischen Fachwortes.