

# **Materialien zur Informatik I**

**R.Deissler**

**WS 03/04**

# Inhaltsübersicht

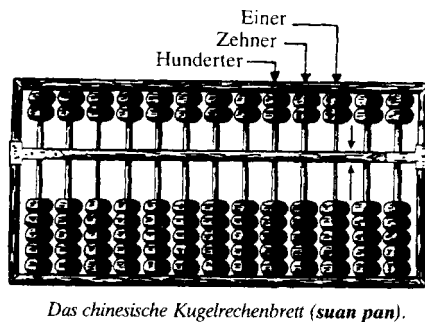
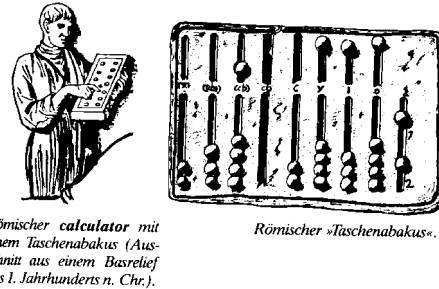
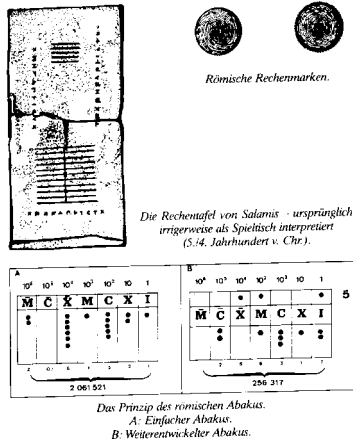
<b>1</b>	<b>GESCHICHTE UND GESCHICHTEN, ÜBERBLICK ÜBER DIE DATENVERARBEITUNG UND AUSBLICKE.1</b>	
1.1	KURZE GESCHICHTE DER <u>DATEN-VERARBEITUNG</u> UND RECHENMASCHINEN .....	1
1.2	KÖNNEN MASCHINEN DENKEN: DER TURING-TEST. ....	4
1.3	ZUM PROBLEM DER ZUKÜNFTIGEN ENTWICKLUNG DER SOGENANNTEN „KÜNSTLICHEN INTELLIGENZ“ UND ZUR VERANTWORTUNG DER INFORMATIKER .....	5
1.3.1	<i>Da ist kein Ich</i> .....	5
1.3.2	<i>Dialog mit einer Version von Weizenbaums "ELIZA"</i> .....	7
1.3.3	<i>Rechenkünstler im Krieg: Computermodelle vermitteln die Illusion von sauberen militärischen Gefechten</i> .....	9
1.3.4	<i>J. Weizenbaum zur Verantwortung der Informatiker: Der Golfkrieg und der Computerprofi</i> .....	10
1.3.5	<i>Ein Beispiel einer extremen Haltung am anderen Ende des Spektrums</i> .....	11
1.3.6	<i>Relativ harmlose Prognose 1988</i> .....	12
1.3.7	<i>Eine Äußerung zum „Computerunterstützten Unterricht“:</i> .....	12
1.3.8	<i>Expertensysteme in der Medizin</i> .....	12
1.4	EINTEILUNG DER INFORMATIK .....	13
1.5	ANWENDUNGEN DER INFORMATIK: ÜBERSICHT.....	14
1.5.1	<i>Tabellenkalkulation mit Geschäftsgraphik</i> .....	15
1.5.2	<i>Datenbanksysteme</i> .....	15
<b>2</b>	<b>DATEN UND IHRE CODIERUNG .....</b>	<b>16</b>
2.1	CODIERUNG IM BINÄRSYSTEM .....	16
2.1.1	<i>Speicherkapazitäten von Speichermedien: (2001)</i> .....	16
2.1.2	<i>Übertragungsraten von Daten (1998)</i> .....	17
2.2	ZAHLEN IM DEZIMAL-, DUAL-, UND HEXADEZIMALSYSTEM .....	18
2.2.1	<i>Dualsystem</i> .....	18
2.2.2	<i>Hexadezimalsystem</i> .....	18
2.2.3	<i>Umrechnung von Dualsystem ins Hexadezimalsystem und umgekehrt.</i> .....	19
2.2.4	<i>Rechnen im Dual- und Hexadezimalsystem</i> .....	20
2.3	DAS ABBILD DER WELT IM COMPUTER.....	21
2.4	DATENTYPEN .....	24
2.4.1	<i>Speichermodell: Folge von Bytes</i> .....	24
2.4.2	<i>Zahltypen</i> .....	24
2.4.3	<i>Datentyp Boolean (Wahrheitswert)</i> .....	27
2.4.4	<i>Datentypen für Zeichen und Zeichenketten</i> .....	27
2.4.5	<i>Codierung komplexer Strukturen</i> .....	28
2.4.6	<i>Zusammenfassung</i> .....	29
2.5	CODIEREN UND WIEDERGEWINNEN VON INFORMATION: BEISPIELE .....	30
2.6	DATENTYPEN IN HÖHEREN PROGRAMMIERSPRACHEN AM BEISPIEL VON TURBO-PASCAL .....	31
<b>3</b>	<b>VOM PROBLEM ZUM PROGRAMM: ÜBERSICHT (VEREINFACHT) .....</b>	<b>32</b>
3.1	INTERPRETER - COMPILER.....	33
3.2	SYNTAX VON PROGRAMMIERSPRACHEN .....	34
3.2.1	<i>Erweiterte Backus-Naur-Form (EBNF)</i> .....	34
3.2.2	<i>Syntax-Diagramme</i> .....	35
3.2.3	<i>Syntax für Mini-Pascal</i> .....	36
3.2.4	<i>Aufgaben zur Syntax von Turbo-Pascal</i> .....	38
<b>4</b>	<b>ALGORITHMEN.....</b>	<b>39</b>
4.1	DEFINITION .....	39
4.2	SORTIERVERFAHREN ALS BEISPIELE FÜR ALGORITHMEN .....	39
4.2.1	<i>Das Bubblesort Verfahren in Pascal.</i> .....	39
4.2.2	<i>Das Verfahren „Sortieren durch Auswahl“ in Pascal.</i> .....	40
4.2.3	<i>Sortieren in einer <u>funktionalen</u> Programmiersprache:</i> .....	41
4.3	STRUKTUR VON ALGORITHMEN .....	43
4.3.1	<i>Einfache Blöcke (elementare Anweisungen, Prozeduranweisungen)</i> .....	43
4.3.2	<i>Sequenz (Verbundanweisung)</i> .....	44
4.3.3	<i>Auswahl (Entscheidung, Fallunterscheidung)</i> .....	44
4.3.4	<i>Wiederholungen, Schleifen</i> .....	46
4.3.5	<i>Steuerstrukturen in der Sprache C und JAVA</i> .....	49
4.4	AUSWAHL VON TURBO PASCAL ANWEISUNGEN MIT BEISPIELEN.....	50
4.4.1	<i>Grundstrukturen zur Steuerung von Algorithmen und grundlegende Anweisungen</i> .....	50

4.4.2	Arithmetische Funktionen.....	51
4.4.3	Stringverarbeitung.....	51
4.4.4	Typen-Umwandlungsfunktionen .....	52
4.4.5	Standard- Prozeduren und Funktionen .....	53
4.4.6	Ein- und Ausgabeprozeduren und Funktionen für die Console.....	53
<b>5</b>	<b>PROZEDUREN UND FUNKTIONEN .....</b>	<b>55</b>
5.1	EINFACHSTE PROZEDUREN UND FUNKTIONEN .....	55
5.1.1	Prozeduren als Abkürzung für Blöcke von Anweisungen .....	55
5.1.2	Funktionen als Abkürzung für Rechenausdrücke .....	55
5.1.3	Funktionen, die nicht durch einen Rechenausdruck darstellbar sind.....	55
5.2	PROZEDUREN UND FUNKTIONEN MIT PARAMETERN.....	56
5.2.1	Deklaration und Aufruf: Formale und aktuelle Parameter .....	56
5.2.2	Gültigkeitsbereich von Variablen: Lokale und globale Variablen .....	56
5.3	ARTEN DER PARAMETERÜBERGABE: WERT- UND VARIABLENPARAMETER .....	57
5.4	VERWENDUNG DER PARAMETER.....	57
5.5	FUNKTIONEN UND PROZEDUREN IM ÜBERBLICK.....	59
5.5.1	Prozeduren, die eigentlich Funktionen sein sollten.....	59
5.5.2	Funktionen und Prozeduren mit Nebenwirkungen: Programmierstil.....	59
5.6	REKURSIVE FUNKTIONEN UND PROZEDUREN.....	59
5.7	BEISPIELPROGRAMM ZUM TESTEN EINFACHER PROZEDUREN UND FUNKTIONEN .....	60
5.8	PROZEDUREN UND FUNKTIONEN IN TURBO PASCAL: AUFGABEN ZUR STRING - BEARBEITUNG.....	61
<b>6</b>	<b>DATENTYPEN .....</b>	<b>63</b>
6.1	DATENTYP ARRAY .....	64
6.1.1	Eindimensionale Arrays (Pascal).....	64
6.1.2	Mehrdimensionale Arrays (Pascal).....	65
6.2	DATENTYP RECORD .....	65
6.3	DATEIEN .....	66
6.3.1	Beispielprogramme für einfachste Dateiverarbeitung .....	68
<b>7</b>	<b>ZUSATZ: POSTGEBÜHREN FÜR BRIEFE (VON TURBO-PASCAL ZU DELPHI).....</b>	<b>72</b>
<b>8</b>	<b>LITERATUR.....</b>	<b>77</b>
8.1	ZUR EINFÜHRUNG IN DIE INFORMATIK ZU EMPFEHLEN .....	77
8.2	VORLESUNGEN ZUR INFORMATIK IM INTERNET .....	78
8.3	ZUR DIDAKTIK DER INFORMATIK .....	79
8.4	GESELLSCHAFTLICHE AUSWIRKUNGEN, KRITISCHE AUSEINANDERSETZUNG MIT DER INFORMATIK.....	79
8.5	FREWARE UND SHAREWARE-PROGRAMME ZUR VORLESUNG.....	80

# 1 Geschichte und Geschichten, Überblick über die Datenverarbeitung und Ausblicke

## 1.1 Kurze Geschichte der Daten-Verarbeitung und Rechenmaschinen

Altertum: Rechnen auf **Abakus** und **Rechenbrett** mit "**Calculi**" (Rechensteinchen)  
Daten-Repräsentation: Steinchen  
Verarbeitung: Bewegen von Hand

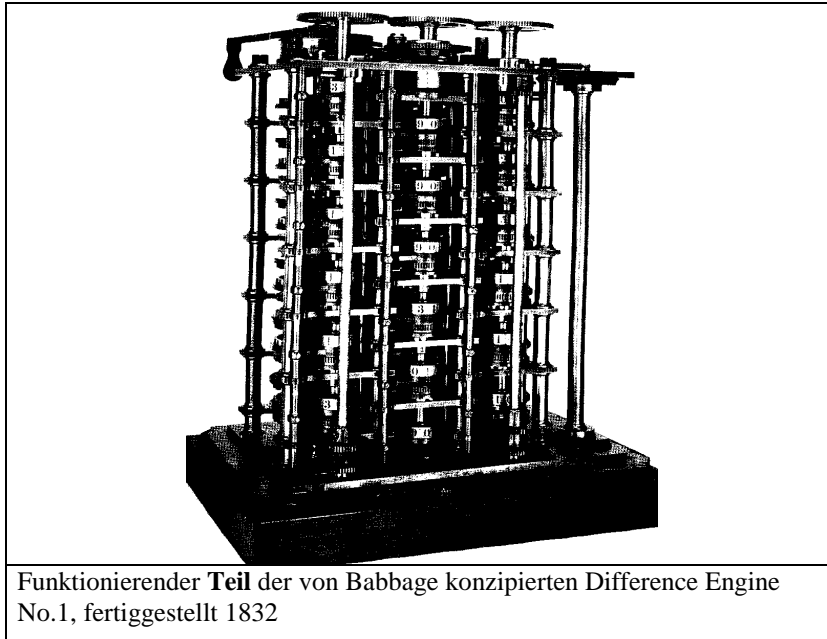


17./18.Jh.: Mechanische Rechenmaschinen (Leibniz, Schickhard, Pascal)  
 Automaten (Spiel, Musik, Unterhaltung)  
Daten-Repräsentation:  
Verarbeitung:

Staffelwalzen, Räder  
 Drehen der Räder

Idee der künstlichen Intelligenz  
 Pascal: Maschinen zeigen Intelligenzleistungen  
 Leibniz :Universeller Logikkalkül

19.Jh.  
 Intermezzo:  
 Idee der Programmgesteuerten Maschine:  
 Mechanischer Webstuhl von Jacquard (Jacquard-Pullover) →Arbeitslosigkeit  
 Differenzmaschine und Analytische Maschine von Charles Babbage und seiner Programmiererin Ada Lovelace - leider nie fertiggestellt (erst 1991 in Londoner Museum nachgebaut)  
Daten-Repräsentation: Zahnräder  
Verarbeitung: Drehen der Räder  
 Programm: Lochstreifen



Funktionierender Teil der von Babbage konzipierten Difference Engine No.1, fertiggestellt 1832

Ende 19.Jh. Hollerith-Maschinen zur Verarbeitung von großen Datenmengen, erstmals eingesetzt zur Volkszählung in den USA.  
Daten-Repräsentation: Lochkarten/Zahnräder  
Verarbeitung: Elektrische Impulse / Drehen von Zählrädern  
 Keine universelle Maschine, kein „Programm“, stets nur für dieselbe Aufgabe zu gebrauchen



Drei Arbeitsgänge erforderte Holleriths Zähl- und Tabelliermaschine: Mit einem Pantographen (Apparatur links auf der Arbeitsfläche, siehe auch Foto oben) übertrug der Operator die Daten aus den Fragebögen auf Lochkarten. Diese wurden in einer Apparatur abgetastet (Arbeitsfläche rechts) und die Impulse über Relais auf Zähluhren weitergeleitet. Nach dem korrekten Einlesen schlug eine Glocke an - der Operator plazierte die Karte in einem sich öffnenden Fach des Sortierers (rechts im Bild). Die abgebildete rekonstruierte Maschine der Bonner Universität ist voll funktionsfähig.

20.Jh. Elektronische Rechenmaschinen  
 (Von Neumann-Architektur)  
Daten-Repräsentation: elektrische Spannung  
Verarbeitung: Schaltkreise (AND, OR, NOT)  
 Programm: dasselbe wie Daten

Die Pionierarbeiten finden lange Zeit - wie so oft - im militärischen Umfeld statt.

1938 Alan Turing: Die „Turing-Maschine“ bildet die Grundlage für eine Theorie der Rechenmaschinen. Der „Turing Test“ wird heute noch als Kriterium für „künstliche Intelligenz“ diskutiert. (GB Spionageabwehr, „Enigma“ geknackt)

1940 Konrad Zuse: Relais, Postschrott (Militär uninteressiert)

1945	Mauchly: Mark I, Relais, Zimmergröße (US Army) Aiken: ENIAC , Röhren (US Navy)
1945	John von Neumann: Architektur der heutigen Rechenanlagen (Berater des US Präsidenten). <ul style="list-style-type: none"><li>• Daten und Programme sind von derselben Art</li><li>• Sequentielle Verarbeitung</li></ul>
1949	Erfindung des Transistors (Shockley). Damit wird wesentliche Verkleinerung der Rechner möglich.
Danach	Kleinere Rechner mit geringem Stromverbrauch Hohe Integration (viele Bausteine auf einem "Chip") billige Fertigung als Massenware Laufende Geschwindigkeitssteigerung (MegaFlops, Gigaflops, ....)
1970er Jahre	Verbreitung des Mikroprozessors ermöglicht Personal Computer („PC“), die die großen Zentralrechner („Mainframes“) vielfach ersetzen und auch Privatpersonen den Besitz eines Rechners erlauben.
1980er Jahre	Erste „Home Computer“ Anfang der 80er Jahre: „Apple II“ (Steve Jobs) und „Commodore C64“ Kurz darauf folgt der erste IBM PC. Bill Gates entwickelt das Betriebssystem dafür: Aufstieg der Softwarefirma Microsoft
1990er Jahre	Weitere Steigerung der Verarbeitungsgeschwindigkeit, Vergrößerung der Speicherkapazität von Bausteinen, weite Verbreitung von Parallelrechnern („Superrechner“). Computer dringen in alle Bereiche des täglichen Lebens ein. Entwicklung der Sensorik: Computer lernen sehen und hören, die Mensch-Maschine-Schnittstelle wird den menschlichen Bedürfnissen immer besser gerecht. Grundlage der Entwicklung der Bausteine sind neue Materialien und neue Fertigungsmethoden (z.B. werden 0,1µm-Leitungen hergestellt, d.h. 1000 Leitungen auf einer Haaresbreite!). Bis 2015 prognostiziert: Bibliothek mit 15 000 Büchern auf einem Chip. Moore's Gesetz: Speicherkapazität pro cm <sup>2</sup> verdoppelt sich etwa alle 18 Monate.
1975	Bill Gates: Geschäftsleute lösen die Pioniere der Computerentwicklung ab. Nicht die beste Idee und das beste Produkt, sondern das beste Marketing entscheiden über die weitere Entwicklung.

## 1.2 Können Maschinen denken: Der Turing-Test.

Die neue Fassung des Turing-Tests ist ein Spiel, an dem drei Personen teilnehmen: ein Mann (A), eine Frau (B) und ein Fragesteller (C), dessen Geschlecht keine Rolle spielt. Der Fragesteller befindet sich in einem eigenen Raum, getrennt von den beiden anderen Personen. Der Fragesteller hat die Aufgabe herauszufinden, welche von den beiden anderen Personen der Mann oder die Frau ist. Der Fragesteller kennt sie lediglich unter der Signatur X und Y.

Ist das Spiel zu Ende, dann sagt er entweder ‚X ist A und Y ist B‘ oder umgekehrt. Person C darf Fragen beliebiger Art stellen: die Aufgabe des Mannes A ist es, den Fragesteller möglichst dahin zu bringen, eine falsche Identifizierung vorzunehmen, während die Frau B das Ziel verfolgt, dem Fragesteller zu helfen. Damit der Fragesteller die beiden anderen nicht an der Stimme erkennt, erfolgt der Austausch der Antworten über beschriebene Zettel beziehungsweise getippt über Fernschreiber. Heute würde man Terminals benutzen.

C könnte also eintippen:

*>Kann X mir sagen, welche Kleidung er oder sie heute trägt?*

Nehmen wir an, X ist der Mann A. Da dieser versuchen wird, C dahin zu bringen, eine falsche Identifizierung vorzunehmen, könnte er etwa antworten:

*>Ich trage ein geblühtes Hemd, eine Perlenhalskette, einen halblangen schwarzen Rock und schwarze Schuhe.*

Diese Antwort erscheint auf dem Bildschirm im Zimmer von C, und C kann die nächste Frage eingeben.

Wie werden sich A und B wohl verhalten? B wird wahrscheinlich am besten ehrlich antworten, etwa mit Bemerkungen wie ‚Ich bin eine Frau, hören Sie nicht auf ihn!‘, aber das hilft nichts, da der Mann ebenfalls Äußerungen dieser Art machen wird. Wird es dem Fragesteller gelingen, eine richtige Identifizierung vorzunehmen? Hinreichendes Intelligenzniveau von A und B vorausgesetzt, wird sich der Fragesteller sicherlich schwer tun. Bei einem statistischen Test mit 100 Versuchspersonen und unter der Prämisse, dass Indifferenz nicht zugelassen ist, würde man erwarten, dass zirka 50 Personen X mit A und Y mit B identifizieren und die anderen 50 die umgekehrte Identifikation vornehmen.

Was hat dies alles nun mit der Frage zu tun, ob Computer denken können? Ganz einfach. Turing ersetzt den Mann durch eine Maschine, die die gleiche Rolle spielen muß. Die entscheidende Frage ist nun: kommt der Fragesteller jetzt auch genauso oft zu einem falschen Ergebnis wie im Originalspiel? Genau diese Frage tritt an die Stelle der ursprünglichen Frage ‚Können Maschinen denken?‘. Wird sie positiv beantwortet, so ist unbestreitbar der Maschine Intelligenz zu attestieren.

Die Essenz dieses Tests ist vielleicht für jemanden, der erstmals damit konfrontiert ist, nicht ohne weiteres einsehbar. Wesentlich ist nicht der Bildschirm oder das Täuschungsmanöver, was nur dazu dient, dem ganzen einen experimentellen Anstrich zu geben. Worauf es in diesem Test wirklich ankommt, ist: Kann die Maschine einen Menschen so überzeugend nachahmen, dass diese von einem Menschen nicht mehr zu unterscheiden ist?

Turing geht von der Prämisse aus, dass eine Maschine, die einen Menschen überzeugen kann, sie zeige intelligentes Verhalten, auch tatsächlich intelligent ist. Intelligenz wird also sozusagen behavioristisch- an Verhaltenskriterien - festgemacht.

Ausschlaggebend ist hierbei das Reden und das Wissen:

Redet die Maschine (über das Terminal) wie ein Mensch und hat die Maschine auch das Wissen eines Durchschnittsmenschen? Die Maschine muß nämlich ungefähr das sagen, was durchschnittliche Menschen in solchen Situationen sagen würden. Genauso muß die Maschine über Themen Bescheid wissen worüber Menschen normalerweise sprechen: Politik, Sonette, den Wetterbericht, Liebesfilme, Gott, den Dollarkurs.

Der Turing-Test zeigt auch sehr schön, wie Intelligenz ohne Bezugnahme auf eine physikalische Trägersubstanz geprüft werden kann. Er abstrahiert völlig von der physikalischen Realisation der Intelligenz. Intelligenz ist nicht an die biologische Trägermasse Gehirn gebunden und es würde nichts bringen, eine Denkmaschine dadurch menschlicher zu machen, indem sie in künstliches Fleisch eingebettet wird. Unwichtige physische Eigenschaften - Aussehen, Material, Stimme - werden durch die Versuchsanordnung ausgeschaltet, erfasst wird das ‚reine‘ Denken.

Derzeit ist man weit davon entfernt ein Programm zu haben, das den Original-Test bestehen würde und vielleicht wird es nie ein solches geben. Es existiert aber ein bekanntes Computerprogramm, das einen modifizierten Turing-Test bestanden hat. Das Programm heißt PARRY, stammt von dem amerikanischen Psychiater K. M. Colby und simuliert einen paranoiden Patienten.

Man kann sich mit dem Programm über die Tastatur unterhalten und das Programm antwortet in einer etwas neurotischen Art. Die Tests wurden in mehreren Varianten durchgeführt. In einer ersten Testserie führten Psychiater Interviews mit psychisch Kranken über ein Terminal. In einigen Fällen kommunizierten die Ärzte ohne ihr Wissen nicht mit psychisch Kranken, sondern mit PARRY. Kein einziger Interviewer bemerkte, dass er einen Computer diagnostizierte.

Natürlich profitiert das Programm von der Tatsache, dass es für Paranoiker typisch ist, auf bestimmte Fragen ausweichende Antworten zu geben und auf Reizworte in mechanischer Weise zu reagieren. Trotzdem scheint mir das Bestehen dieses Tests eine beeindruckende Leistung für ein Programm zu sein - der erste und bislang einzige mir bekannte Fall.

(aus: Manhart, Klaus, Menschliche Maschinen, c't 1993, Heft 3)



Der einzige Weg zu wissen, dass eine Maschine denkt, ist, diese Maschine selbst zu sein und sich selbst denken zu fühlen.  
Alan Turing

### 1.3 Zum Problem der zukünftigen Entwicklung der sogenannten „Künstlichen Intelligenz“ und zur Verantwortung der Informatiker

Joseph Weizenbaum, Professor für Informatik am MIT (heute emeritiert), entwickelte in den 60er Jahren „als Fingerübung“ das Programm „ELIZA“ (s.unten), das in der Folge von Psychologen aufgegriffen und als ernsthaftes Therapieprogramm gegen Weizenbaums Willen weiterentwickelt wurde. Im folgenden Interview bezieht sich Weizenbaum auf die Entwicklung einer zu der Zeit erschienen Fortentwicklung „Overcoming Depression“.

#### 1.3.1 Da ist kein Ich

Joseph Weizenbaum im Gespräch mit Thomas J. Schult  
Interview in c't 1993, Heft 1:

Zwei Nachrichten der letzten Zeit:

Die Therapiesoftware ‚Overcoming Depression‘ wird auf dem amerikanischen Markt als probates Mittel gegen leichte bis mittlere Depressionen angeboten. Bei einem Wettbewerb in Boston gelingt es dem Programm ‚Whimsical Conversation‘, die Hälfte der Jury in einem Doppelblindversuch zu überzeugen, dass sie mit einem Menschen kommuniziert.

Bald ist es drei Jahrzehnte her, dass Joseph Weizenbaum am Massachusetts Institute of Technology (MIT) sein Programm ELIZA zum Leben erweckte, die Urahnin der algorithmischen Gesprächstherapie. Von ihm nur als Fingerübung für den aufkommenden Dialogbetrieb gedacht, wurde ihr jedoch eine psychologische Kompetenz zugesprochen, die ihn noch heute erschreckt. Wer ELIZA verfiel und alle Anwesenden bat, den Raum zu verlassen, war der bedingungslosen Computergläubigkeit überführt.

Auch Weizenbaums Leben hat sie verändert. Der MIT-Professor wurde zum unbequemen Streiter wider die Computerisierung der Gesellschaft und unseres Bildes vom Menschen. Oft zieht es ihn dabei in unser Land, aus dem er 1936 emigrierte. Die beiden eingangs erwähnten Programme werden in einer der nächsten Ausgaben von c't thematisiert. Vorher jedoch soll Weizenbaum selbst zu Wort kommen, vielleicht, damit wir nicht noch einmal in die ELIZA-Falle tappen. Eine passende Umgehung für das Gespräch ist das seit über einem Jahr bestehende Institut für Informatik und Gesellschaft an der Freiburger Universität, wo Weizenbaum einige Wochen als Gastprofessor verbringt.

*c't*: Herr Professor Weizenbaum, ist es angesichts des von Peter Colby in den Staaten vertriebenen Programms ‚Overcoming Depression‘ wieder an der Zeit, vor einer Überschätzung maschineller Kommunikationspartner zu warnen?

*J. Weizenbaum*: Es ist nie zu früh gewesen und auch nicht zu spät. Ich glaube, wir müssen das immer machen. Vor meiner Zeit am MIT war ich an der Stanford Universität in Palo Alto und war mit Ken Colby sehr befreundet, er hatte damals einen kleinen Jungen, Peter. Am MIT habe ich dann angefangen mit der Sache, die schließlich ELIZA wurde. Ich erinnere mich noch sehr gut, dass ich Ken Colby etwas über ELIZA schrieb und ein ganz primitives Dialogbeispiel mitschickte.

*c't*: Sie haben die Familie Colby infiziert...

*J. Weizenbaum*: Ja. Ich wollte aber darauf bestehen, dass überhaupt nichts über dieses Programm in irgendeiner psychologischen Zeitschrift erscheint, weil ich dachte, dass es mit Psychotherapie nichts zu tun hat.

*c't*: Colby ist Psychologe?

*J. Weizenbaum*: Ja, Psychoanalytiker. Gegen meinen Willen hat er dann doch etwas in einer psychologischen Zeitschrift veröffentlicht, so eine Ankündigung der Art „Wir sind jetzt dabei, automatische Therapieverfahren herzustellen und werden in wenigen Jahren Hunderte von Patienten gleichzeitig behandeln können“ und so weiter. Da war ich sehr sehr entsetzt. Es war eine grobe Verletzung unserer Freundschaft. Na ja - dann sind wir auseinandergeschieden.

*c't*: Hat er Sie damals auch gefragt, ob Sie mit ihm das auch weiterentwickeln wollen?

*J. Weizenbaum*: Ich glaube, es war klar, dass ich das nicht wollte - für mich ist das so ähnlich wie Astrologie oder irgendeine andere Scharlatanerie, fast unabhängig davon, wie es gemacht wird. Wir haben uns gestritten darüber und sind dadurch auseinandergeschieden. Fast 30 Jahre später kommt dieses Programm; diesmal ist es sein Sohn Peter, der das vertreibt, aber natürlich ist sein Vater dabei. Merkwürdig dabei ist, dass es wirklich nichts anderes als ELIZA ist. Es nutzt zwar die Tatsache aus, dass eben die heutigen Computer viel mehr Speicherkapazität haben und auch sehr viel schneller sind als damals, so dass man viel mehr reinpacken kann. Aber die Struktur der Sache, also wie das funktioniert, das ist absolut ELIZA, das ist überhaupt keine Frage. Man ließ sogar manche der Antworten drin, und da sind die alten ELIZA-Tricks wie ‚Tell me more about your mother‘. Manche sagen vielleicht, dass man auch Bücher für therapeutische Zwecke nutzen kann, also warum keine Programme? Nun, ich habe etwas dagegen. Ein wichtiger Punkt dabei ist, dass viele unserer Geisteskrankheiten und unserer Schwierigkeiten, mit uns selbst klarzukommen, auf der Tatsache beruhen, dass wir in einer ungeheuer abstrakten und mechanistischen Welt leben.



Dass wir selten die Natur direkt empfinden, dass es heutzutage fast unmöglich ist, Musik zu hören, die nicht elektronisch verstärkt ist. Das ist natürlich nicht die einzige Sache. Aber wir leben so eng zusammen mit Maschinen und maschinellen Sachen. Und auch wenn keine Maschine dabei ist, machen wir Sachen nach Rezept, das heißt, so wie wir selbst handeln, nach Regeln etwa, ist die Welt zu einer großen Maschine geworden, in der wir ein kleines Teil sind. Ich glaube, viele Geisteskrankheiten werden durch diese Dinge verstärkt und jetzt kommt noch so eine Maschine! Geht das nicht überhaupt in die falsche Richtung?

Und zweitens bin ich dagegen, weil das ganze Programm eine Lüge ist. Erstmal spricht es von sich selbst, sagt, 'ich verstehe dich', 'ich sage dir' und so weiter, aber da ist kein Ich - da ist eine tote Sache. Es sagt auch ganz deutlich öfter, dass es 'versteht', oder jedenfalls spielt es so, als ob es versteht. Und die Leute denken wirklich, jedenfalls ging das mit ELIZA so, dass die Programme so viel können. Es wird die Illusion hergestellt, dass da wirklich ein Ich ist. ein lebendes Wesen, das tatsächlich versteht. Das ist natürlich eine Lüge - niemand versteht, niemand ist da.

*c't*: Wenn der Patient die Lüge nicht wahrhaben will, stört sie dann überhaupt?

*J. Weizenbaum*: Carl Rogers [Vater der Klientenzentrierten Gesprächstherapie, Anm. d. Red.], der Psychologe, der mit ELIZA parodiert wurde, hat eine zentrale Absicht, die ich auch gut finde: Dass der Patient erfährt, dass es möglich ist, ihn zu lieben, und dass es deswegen möglich ist, dass er sich selbst lieben kann. Die eigentliche Krankheit ist, dass der Patient sich nicht liebt, sondern sich vielleicht sogar verachtet und auch nicht glaubt, dass er von irgendeinem anderen Menschen geliebt werden kann. Und in dieser Therapie macht er die Erfahrung, dass es doch möglich ist. Und da fängt das Heilen an. Aber was könnte es bedeuten. dass ein Computer mich lieben kann? Das ist schließlich eine sehr seltsame Sache.

Ich denke, dass die Colby Leute sich den Markt ziemlich genau angesehen haben und die Aussichten, damit Geld zu verdienen. Ich glaube schon, dass sie schon vorher wußten, dass die Zahl der Leute, die irgendwie depressiv sind in unserer Welt sehr groß ist ungeheuer groß, Millionen. Deswegen bietet dieses Ding eine Möglichkeit, viel Geld zu verdienen. Es kann sein, dass sie sich einreden, dass es auch die Möglichkeit bietet, sehr vielen Leuten zu helfen und dass sie das Geld für nebensächlich halten. Aber 200 Dollar ist doch eine ziemliche Menge Geld.

*c't*: Es macht sich schon bei drei eingesparten Therapeutenstunden bezahlt! Aber im Ernst: Gibt es neben der Liebesfähigkeit noch etwas, was maschinellen Dialogpartnern abgeht?

*J. Weizenbaum*: Meine These ist, dass es viele Sachen gibt, die wir wissen und die sogar lebenswichtig sind für uns, die der Computer nie wissen kann. Da wird oft gesagt: 'Ja, man könnte dem Computer das doch erzählen. Man muß es nicht erfahren haben, man kann es in der Kommunikation lernen.' Nein, das bestreite ich. Ich denke hier, ich weiß gerade nicht, warum, an den simulierten Orgasmus. Ich habe gehört, es gibt so etwas auf dieser Welt. Das ist aber ganz etwas anderes als ein Orgasmus, das ist dann nicht eine Folge davon, dass dieses Liebespaar etwas erfahren hat im Leben. Es sagt nichts über die Erkennung der Bedeutung eines Liebesaktes. Es ist einfach eine Rolle wie die eines Schauspielers. Ich denke, hier in Deutschland gibt es Straßenmusiker, die die amerikanischen Songs auf englisch singen, und wenn man sie anspricht, dann stellt sich heraus, sie können kein Englisch. Ich kann auch manche Ausschnitte aus italienischen Opern singen - wenn ich überhaupt singen kann, das ist sehr fragwürdig - ohne italienisch zu können. Davon sprechen wir ja nicht, das ist nicht Verstehen. Ich kann vielleicht einen Schauspieler trainieren, er kann ein ELIZA-Spiel üben. Er ist da, statt der Maschine, und er hat ein sehr ernstes Gesicht und sagt: 'Aber sie haben noch nichts über ihre Mutter gesagt.' Das ist einfach eine Rolle. Aber später fragen ihn die Leute 'Wissen Sie. Sie haben jetzt hundertmal am Broadway diese Rolle gespielt. diesen Psychiater, also da muß ich ihnen mal sagen, was mir neulich passiert ist ...

*c't*: In Deutschland wird manchmal ein Vergleich gezogen dass es mit dem Gebrauch des Wortes denken ein ähnlicher Fall ist wie mit dem Wort 'fliegen'. In den letzten Jahrhunderten war es immer so dass nur Vögel fliegen konnten, aber schließlich versuchten es irgendwann auch technische Geräte. Da kam auch die Frage auf: Soll man das 'fliegen' nennen? Man hat es so genannt, obwohl es ja auf ganz anderen Prinzipien basiert. Glauben Sie, dass man bei Computern von Denken in einem anderen Sinne reden kann, oder meinen Sie, dieses Wort sollte für Menschen vorbehalten bleiben?

*J. Weizenbaum*: Ich würde ganz bestimmt nicht sagen, dieses Wort sollte für Menschen vorbehalten bleiben, ich würde das sogar nicht von dem Wort 'Liebe' sagen. Außerdem glaube ich, dass Tiere lieben können und dass Tiere denken. Vielleicht denken sie nicht an Hölderlin, und sie invertieren keine großen Matrizen und lösen keine Differentialgleichungen - vielleicht das auch, ich weiß nicht. Ich denke an Delphine, die ein Gehirn haben, das größer ist als unseres. Aber jedenfalls weiß ich, dass ein Hund fühlt, so wie wir fühlen: Einsamkeit; Traurigkeit, Liebe, Freude. Ich glaube, das zu wissen. Ich kann es nicht beweisen. Aber das kann ich auch nicht bei einem anderen Menschen beweisen. Ich würde sagen, dass das menschliche Denken unter anderem deshalb anders als das Denken der Tiere ist, weil Menschen so ganz anders sozialisiert sind als Tiere. Und das hat auch damit zu tun, dass Menschen einen ganz anderen Körper haben als Tiere und andere biologische Bedürfnisse.

*c't*: Und der Computer als dritte Gruppe?

*J. Weizenbaum*: Ich könnte es nicht verteidigen, das Wort 'denken' auf lebende Wesen zu beschränken.

*c't*: Aber glücklich sind Sie auch nicht mit einem Begriff wie 'Künstliche Intelligenz'?

*J. Weizenbaum*: Nein, obwohl man sich ja nach so einer langen Zeit damit abfinden muß, dass er nie verschwinden wird. Da kann man nicht ewig drüber klagen Da gibt es sicher auch andere solche Sachen. Außerdem glaube ich, dass eine Beschränkung nichts hilft. Das hat ja überhaupt keinen Effekt.

Es ist nicht die Frage, ob Maschinen intelligent sein können oder nicht. Ich glaube, sie können intelligent sein und denken, jedenfalls so, wie ich das Wort verstehe. Das ist nicht die Frage. Die Frage ist, inwiefern und in welchen Bereichen wir die Maschine als unsere Denkhilfe benutzen sollen, und was passiert, wenn wir über die Grenzen gehen.

Zurück zur Psychotherapie:

Es wird inzwischen so gesprochen, dass ein Computer ein Freund ist, also ein Therapeut und auch ein Freund. Das finde ich tragisch, und ich finde es auch gefährlich. Gerade in einer Welt wie unserer Massenwelt heute, in der so viel Einsamkeit ist, und für viele Menschen überhaupt keine Gelegenheit besteht, mit anderen Menschen zusammenzukommen. Ich habe von Tieren gesprochen, ich habe gesagt, ich scheue mich überhaupt nicht, über Liebe zu sprechen im Zusammenhang mit Tieren, ich würde mich aber scheuen, über Liebe zu sprechen im Zusammenhang mit Computern. Da wird man mich bestimmt jetzt in so eine Ecke schieben.

Wir können natürlich jetzt annehmen, wir machen einen kleinen Chip in ein menschliches Gehirn. Der Chip hilft Leuten, besser zu rechnen. Und dann noch einen Chip, vielleicht ein bisschen größer, da kann man seine Adressen und alle möglichen Sachen speichern, die man sonst in ein Notizbuch reinschreibt, das man aber nur verlieren würde und so weiter. Schließlich ist da nur noch ein relativ kleiner Teil biologischer Masse, also ein Gehirn, und ein großer Computer damit verbunden. Und an welchem Punkt fange ich an zu sagen: Das ist ja kein Mensch, es ist auch kein Tier, es ist eine Maschine? Wenn man so weit gehen muß, um zu zeigen, dass der Computer nicht anders ist als wir, dann, glaube ich, hat man schon verloren. Außerdem denke ich, die Idee, mit so einem Hilfschip unseren Kopf mit einem Computer zu verbinden, ist obszön. Man kann mich jetzt fragen: Warum ist das obszön? Der Begriff Obszönität hat mit rationalem Denken nichts zu tun. Das hat mit der Erziehung zu tun, damit, was für Werte ein Mensch hat.

### 1.3.2 Dialog mit einer Version von Weizenbaums "ELIZA"

PLEASE STATE YOUR PROBLEM !

? I FEEL SICK

TELL ME MORE ABOUT SUCH FEELINGS.

? IN THE MORNING I AM TIRED AND DO NOT WANT TO GO TO WORK

IS IT BECAUSE YOU ARE TIRED AND DO NOT WANT TO GO TO WORK THAT YOU CAME TO ME ?

HOW DO YOU FEEL WHEN YOU HAVE TO WORK ?

? I AM TOO TIRED TO EVEN TAKE A PENCIL

IS IT BECAUSE YOU ARE TOO TIRED TO EVEN TAKE A PENCIL THAT YOU CAME TO ME ?

? NO; NOT REALLY: IT IS BECAUSE OF THIS GENERAL FEELING

WHAT DOES THAT SUGGEST TO YOU ?

? THERE MUST BE SOMETHING WRONG WITH ME BUT I DO NOT KNOW WHAT IT IS

OH, YOU KNOW ? THAT MEANS, YOU EXIST, AREN'T YOU ?

WHY DO YOU ASK ?

? HÄ ?

WHAT DOES THAT SUGGEST TO YOU ?

? I DO NOT UNDERSTAND WHAT YOU MEAN

WHY DO YOU ASK ?

WE WERE DISCUSSING YOU, NOT ME.

? I DON'T ASK BUT I AM FRUSTRATED

IS IT BECAUSE YOU ARE FRUSTRATED THAT YOU CAME TO ME ?

DON'T YOU REALLY ASK BUT YOU ARE FRUSTRATED ?

? YES

YOU SEEM QUITE CERTAIN.

? OF COURSE

PLEASE GO ON.

? YEAH

PLEASE GO ON.

? OK

AH ! YOU BELIEVE ME.

? NOT REALLY BUT YOU ARE A FUNNY GUY

WHAT MAKE YOU THINK I AM A FUNNY GUY ?

DID YOU THINK THEY MIGHT NOT BE A FUNNY GUY ?

WE WERE DISCUSSING YOU, NOT ME.

? YOU NO GOOD SPEAK NO ENGLISH

I AM SORRY I SPEAK ONLY ENGLISH.

ARE YOU SAYING 'NO' JUST TO BE NEGATIVE ?

ARE YOU SAYING 'NO' JUST TO BE NEGATIVE ?

WE WERE DISCUSSING YOU, NOT ME.

GOOD IS MUCH BETTER THAN BAD, YOU KNOW ?

? YOU ARE CRAZY BUT I KNEW THAT ALL PSYCHIATRISTS ARE LIKE THAT THAT'S WHY THEY

CHOSE THAT JOB!

IN WHAT WAY ?

WHAT MAKE YOU THINK I AM CRAZY BUT YOU KNEW THAT ALL PSYCHIATRISTS ARE LIKE THAT THAT'S WHY THEY CHOSE THAT JOB! ?

WHY DO YOU ASK ?

DID YOU THINK THEY MIGHT NOT BE CRAZY BUT YOU KNEW THAT ALL PSYCHIATRISTS ARE LIKE THAT THAT'S WHY THEY CHOSE THAT JOB! ?

DID YOU THINK THEY MIGHT NOT BE LIKE THAT THAT'S WHY THEY CHOSE THAT JOB! ?

WE WERE DISCUSSING YOU, NOT ME.

? YOU START TALKING QUITE A LOT BUT I CAME TO YOU TO GET SOME HELP NOT TO ENTERTAIN YOU. THIS TALK STARTS TO UPSET ME !!!

YOU SEEM TO BE A VERY HELPLESS. GIVE MORE EXAMPLES !

WE WERE DISCUSSING YOU, NOT ME.

WE WERE DISCUSSING YOU, NOT ME.

? I AM NOT HELPLESS BUT YOU BETTER LISTEN TO ME MORE CLOSELY OR GET MAD YOU SEEM TO BE A VERY HELPLESS. GIVE MORE EXAMPLES !

WE WERE DISCUSSING YOU, NOT ME.

IS IT BECAUSE YOU ARE NOT HELPLESS BUT I BETTER LISTEN TO YOU MORE CLOSELY OR YOU THROW STUFF AT ME THAT YOU CAME TO ME ?

? I WILL GET RUDE IF YOU GO ON LIKE THIS. I WANT TO HAVE MY MONEY BACK !

IN WHAT WAY ?

DO YOU THINK IT'S LIKELY THAT IF I GO ON LIKE THIS. YOU WANT TO HAVE YOUR MONEY BACK ! ? YOUR MONEY BACK ! .

WHAT WOULD IT MEAN TO YOU IF YOU GOT TO HAVE YOUR MONEY BACK ! ?

WE WERE DISCUSSING YOU, NOT ME.

MONEY IS PERVERSE, ISN'T IT ?

YOU HAVE IT AND HAVEN'T IT AT THE SAME TIME.

? THIS IS TRUE BUT IT DOES NOT HELP ME WITH MY PROBLEMS !!!!!

YOU SEEM TO BE A VERY HELPLESS. GIVE MORE EXAMPLES !

YOUR PROBLEMS !!!!! .

? UHHHHHHH. I CAN NOT GET ANY WORK DONE

HOW DO YOU FEEL WHEN YOU HAVE TO WORK ?

? I FEEL VERY TIRED. I THINK I'LL NEVER ACCOMPLISH ANYTHING.

TELL ME MORE ABOUT SUCH FEELINGS.

? MY FATHER WAS MUCH MORE ACTIVE THAN ME. I THINK HE'D BE ANGRY AT ME SEEING ME TO BE A LOSER !

TELL ME MORE ABOUT YOUR FAMILY.

YOUR FATHER WAS MUCH MORE ACTIVE THAN ME. YOU THINK HE'D BE ANGRY AT YOU SEEING YOU TO BE A LOSER ! .

? YES. MY FAMILY WAS A NICE MIDDLE CLASS FAMILY, NOTHING SPECIAL.

Korrigieren Sie die Eingabe

? WHY ?

LETS DISCUSS FURTHER WHY YOUR FATHER WAS MUCH MORE ACTIVE THAN ME. YOU THINK HE'D BE ANGRY AT YOU SEEING YOU TO BE A LOSER ! .

? YES HE SHURE WOULD BE

YOU SEEM QUITE CERTAIN.

? HE ALWAYS TOLD ME I WAS A LOSER

CAN YOU THINK OF A SPECIFIC EXAMPLE ?

?.....

### 1.3.3 Rechenkünstler im Krieg: Computermodelle vermitteln die Illusion von sauberen militärischen Gefechten

Aus: Computer Zeitung 2002, Ausgabe 047, Seite 6

**Mathematiker spielen bei der Vorbereitung von Kriegen eine ebenso wichtige Rolle wie Militärstrategen. Das gilt für die Wissenschaftler des vergangenen Jahrhunderts wie Alan Turing oder Grace Hopper, die mit ihren Erfindungen Weichen in der Computerentwicklung stellten. Es gilt aber noch sehr viel mehr bei der Erarbeitung moderner Kriegsszenarien des Informationszeitalters.**

Für ihre Absicht, im Irak das Regime von Saddam Hussein zu stürzen, führen die US-Militärs verschiedene Gründe an. Von den Vermutungen über eine Atomwaffenproduktion des Irak über angebliche Anlagen zur Produktion chemischer Waffen bis hin zu möglichen Ausbildungslagern der Al Quaida ist alles da, den Angriff zu rechtfertigen.

Mindestens ebenso wichtig ist jedoch der unerschütterliche Glaube an den Erfolg der Militäraktion, an eine schnelle, fast unblutige Operation, mit der der Diktator entfernt wird. Für diesen surgical war sind nach Angaben der Militärs alle Parameter durchgerechnet worden. Dementsprechend sicher ist Richard Perle, Berater des US-Verteidigungsministeriums, dass der Irak keine Chance habe: Saddam should do the math.

Es ist die Mathematik mit ihren Fortschritten in der Spieltheorie und mit stochastischen Methoden, die nicht ganz unschuldig am Bild vom sauberen, schnellen Krieg ist. Das moderne Combat Modeling basiert auf komplexen Rechenmodellen wie sie ursprünglich von dem Kriegsmathematiker Frederick William Lanchaster gefunden wurden, der die Überlegenheit einer Seite bestimmen wollte.

#### Modell widerlegt Sinn des Kriegs

Auch die Überlegungen des Wetterforschers Lewis Fry Richardson werden benutzt, der den Nutzen der Kriege widerlegen wollte. Später sorgten Mathematiker wie Alan Turing, dessen Maschinen die Verschlüsselungscodes der Nazis knackten, für die Verbindung von Wissenschaft und Militär. Ein weiteres Beispiel für die Kooperation ist die Computerpionierin und Compiler-Erfinderin Grace Hopper, die sich ihre Spuren in der Navy verdiente.

Sind militärische Konflikte und deren Ausgang also einfach vorzuberechnen? Mit dieser Frage beschäftigten sich Experten auf einer Tagung in Schweden. Hochentwickelte Rechenmodelle, die auch das Gerüst von modernen Computerspielen wie Age of Empire bilden, seien in der computerunterstützten Kampfanalyse selbstverständlich, führte der schwedische Mathematiker und Militäranalytiker Helge Loftstedt auf der Veranstaltung aus. Für ihn sind es die Entwicklungen der Mathematik und der Computertechnik ausfallsichere Netze, autonome Munition, Satellitenunterstützung die den modernen Krieg bis etwa 2020 entscheidend prägen werden: Die Nutzung der modernen Waffen wird von flächendeckenden Einsätzen hin zu duellartigen Kämpfen gehen.

Weniger Manöver, dafür mehr Politik, so könnte man die Situation zusammenfassen. Loftstedt: Mehr Politik heißt, dass diese sich mehr auf die Berechnungen der Strategen verlassen muss. Loftstedt diagnostiziert dennoch ein Missverhältnis zwischen den Möglichkeiten des mathematische Combat Modeling und der Wirklichkeit: Im Idealfall ist die Datenlage klar, sind die verwendeten Methoden geklärt, werden die Ergebnisse von versierten Leuten interpretiert. Doch die Realität sieht anders aus: Die derzeit vorhandenen Theorien sind unausgereift oder inadäquat, viele Computer inkompatibel. Dazu komme der subjektive Widerwillen mancher Militärs. gegen das ihrer Meinung nach versuchte Wegrechnen des Kriegs.

Svend Claussen, der für die dänischen Streitkräfte und die NATO mathematische Rechenmodelle entwickelt, verteidigt dagegen den Einsatz des Combat Modeling: Die Tools sind sinnvoll für die taktische Planung und das Beschaffungswesen, sie sollten nicht mit Werkzeugen verwechselt werden, die den Ausgang eines bewaffneten Konflikts vorhersagen.

Die Formel CCCCIWS, bestehend aus den Faktoren Command, Control, Communication, Computers, Intelligence, Information Warfare und Sensors, bestimmt laut Loftstedt und Claussen das militärische Denken. Dabei erhalten die Faktoren Aufklärung (Intelligence) und Informations-Sabotage (Information Warfare) eine immer größere Bedeutung.

Für Ralf Bendrath von der Berliner Forschungsgruppe Informationsgesellschaft und Sicherheitspolitik ist es jedoch offensichtlich, dass der Information Warfare erst ganz am Anfang seiner Entwicklung als militärisches Instrument steht. Bislang handelt es sich dabei nicht um die Störung der IT-Infrastruktur, sondern eher um den Einsatz bloßer Informationen. So seien nur wenige Cyber-Attacken bekannt, die zweifelsfrei einem militärischen Angriff zuzuordnen wären. In einem NATO-Papier zum Bosnien-Krieg wird Information Warfare aber bereits als effektive Waffe zitiert: Präsident Milosevic wurde gezeigt, welche Auswirkungen IFOR-Aktionen auf das serbische Militär und die Kampfkraft seiner Truppen haben würde. Die überzeugende Darstellung der hochwertigeren Informationen hatte den gewünschten Effekt und führte zur Beendigung der Feindlichkeiten in Bosnien. Doch was im Falle Bosniens noch funktioniert haben mag, versagte im Kosovo-Konflikt bereits wieder.

Der Potsdamer Physiker Jürgen Scheffran, der sich seit Jahren mit der Rüstungskontrolle und der Kriegsmathematik befasst, hat denn auch Zweifel, ob sich die Zahlenlehre dafür eignet, soziale Konflikte mit potenziellen militärischen Eskalationen frühzeitig zu berechnen. Allerdings verweist er darauf, dass mathematische Modelle dort ihren Beitrag leisten können, wo gemeinsame Sicherheitskonzepte oder Konfliktvermeidungs-Strategien entwickelt werden.

Der Hauptredner der Konferenz, der Mathematiker Philip J. Davis, beschäftigte sich indes mit einem ganz anderen Aspekt militärisch genutzter Mathematik. Er machte auf den Zusammenhang zwischen den immer besser werdenden

Kriegssimulationen und der Unterhaltungsindustrie mit ihren Kriegsspielen aufmerksam. Wenn Unterhaltung und Krieg bis zur Ununterscheidbarkeit zusammenwachsen, liefert nach Davis die Mathematik etwa mit ihren Algorithmen zur Berechnung von Polygonen das Rüstzeug für die Verschmelzung. Eine Verschmelzung, die so gut gelingen könnte, dass ein Kampf zwischen Robotern oder eine Computersimulation den blutigen Krieg mit seinen Toten und Verletzten ablösen könnte. Es wäre ein sauberer Krieg, würde aber dazu führen, dass eine Mixtur aus Unterhaltung und Krieg entstünde, die so explosiv sei, dass sie das Soziale selbst zerstören könne: Menschen, die nur in den Korridoren von Counterstrike denken können, hätten nach Davis damit die Kultur selbst abgeschafft.



Admiral Grace Hopper (links) und Alan Turing haben entscheidende Projekte ihrer mathematischen Laufbahnen an der Schnittstelle zum Militär absolviert. Hopper verantwortete bei der Navy Cobol-Projekte. Turing half mit seinen Erfindungen dem britischen Geheimdienst im Kampf gegen die Nazis.



### 1.3.4 J. Weizenbaum zur Verantwortung der Informatiker: Der Golfkrieg und der Computerprofi

Aus: c't, 1991, Heft 3

*„Der Golfkrieg, dessen Auswirkungen noch nicht abzusehen sind, hat von Anfang an eines deutlich gemacht: Ohne die moderne Waffentechnologie, die der Irak sowohl von der Sowjetunion als auch von den westlichen Industriestaaten und deren Exporteuren erhalten hat, wäre der Konflikt nicht ausgeartet: Saddam Hussein hätte sich von vornherein keinen Krieg leisten können.“*

Die ZEIT (Nr.5-25, Januar 1991, Seite 23)

Unter dem Titel ‚Nicht ohne uns‘ hielt ich im Jahre 1986 in Karlsruhe eine Rede auf einem Treffen einer Sektion der Gesellschaft für Informatik. Ich sagte damals, dass wir Computerprofis - von einigen wichtigen Ausnahmen abgesehen - kaum in der Position seien, Führungskräfte in Politik und Industrie zu kritisieren, weil sie den internationalen Waffenhandel unterstützten. Jedenfalls nicht, solange wir weiter unsere Arbeit tun, ohne uns über deren endgültigen Zweck zu informieren und ohne im Einzelfall zu entscheiden, ob wir guten Gewissens zu diesem Zweck beitragen können. Wer beispielsweise an Systemen arbeitet, die Computer ‚sehen‘ lehren, muß wissen, dass jeder errungene Erfolg auf diesem Gebiet beinahe zwangsläufig in Technologie umgewandelt und dann in Waffensysteme eingebaut wird.

Cruise Missiles, Tomahawks und Scud-Raketen, ja die gesamte Bandbreite moderner Waffen, deren Hauptzweck es ist, Sachen zu zerstören und Menschen zu ermorden, sind ebenso das Produkt von Computerprofis wie sie eine Profitquelle für Waffenhändler sind. Wir können und dürfen nicht abstreiten, dass wir für eine Katastrophe verantwortlich sind, die erst durch unsere Arbeit ermöglicht wurde.

Die fünf vergangenen Jahre haben meine These auf dramatische Weise untermauert. Die Entwicklung moderner Waffen und in der Folge der moderne Krieg konnte und kann ohne die Mithilfe bereitwilliger oder gar enthusiastischer Computerfachleute nicht stattfinden. Wir Computerprofis, wenn auch nicht alle, haben wichtige Beiträge zu den mörderischen Arsenalen geleistet, die jetzt die Welt verseuchen und die Menschheit terrorisieren.

Zweifellos jubeln einige von uns über den Erfolg (im technischen Sinn) der ‚intelligenten‘ Waffen, die wir mit in diese Welt gesetzt haben. Aber wir haben uns mit ihrem Triumph nicht nur Verdienste erworben, sondern auch Schuld aufgeladen und die Verantwortung dafür, ‚Staatsmänner‘ wie Psychopathen mit den Mitteln versorgt zu haben, mit denen diese die Menschheit bedrohen.

Was ist zu tun?

Zuallererst müssen wir aufhören, dem tröstenden Märchen zu glauben und es zu verbreiten, dass Computer nichts als ein Werkzeug seien. Dass sie und die Arbeit, die wir an ihnen getan haben, wertfrei seien, weil sie sich zu guten ebenso wie zu

schlechten Zwecken eignen. Werkzeuge bekommen ihren Wert von den sozialen, politischen und moralischen Eigenschaften der Gesellschaft, in die sie eingebunden sind. Mit einer Pistole kann man Nägel in Wände schlagen, um schöne Bilder daran aufzuhängen. Ja. Aber in unserer fast hoffnungslos bösen und gewalttätigen Welt ist eine Pistole dazu da, Menschen zu bedrohen und zu töten. Ich glaube, die meisten Computerprofis würden sich weigern, in einer Pistolenfabrik tätig zu sein. Warum sollten sie dann an Systemen arbeiten, deren Zweck am Ende darin besteht, tausendfach zu vollziehen, was eine Pistole nur einzeln vermag?

Wie bei den meisten wirklich wichtigen Dingen beginnt die Aufgabe, die vor uns steht, beim Individuum. Wir können die Fehler dieser brutalen Welt nicht ausmerzen, ohne bei unseren eigenen Fehlern anzufangen.

**Joseph Weizenbaum**

### 1.3.5 Ein Beispiel einer extremen Haltung am anderen Ende des Spektrums Der Pittsburger KI-Forscher Hans Moravec.

„Auf lange Sicht sind Menschen natürlich überflüssig“

Mit solchen Thesen provoziert der Pittsburger KI- und Roboterforscher Hans Moravec. Der in Österreich geborene Wissenschaftler verfolgt seine Vision vom posthumanen Geist mit Vehemenz und Zuversicht; In Zukunft wird die Menschheit in einer vollständig automatisierten Gesellschaft, in einer Art roboterisierten Schweiz, existieren. Die Bevölkerung lebt dann von Steuern, die Roboter zahlen - bis eines Tages intelligente Maschinen den Menschen für überflüssig halten. In etwa 50 Jahren wird dies der Fall sein, schätzt Moravec, der mit dieser Vorstellung keinerlei Probleme hat. Er sieht die intelligenten Roboter eher als seine Geisteskinder und damit als einen ganz normalen Schritt der Evolution.

Zur Zeit arbeitet Moravec an Algorithmen, die es Robotern erlauben, sich autonom in dreidimensionalen Räumen zu bewegen. Aber auch mit simpleren Dingen beschäftigt sich Moravec. Von ihm stammt beispielsweise das Verfahren zur Bestimmung des Akku-Ladezustands in den Powerbooks. Übrigens: Moravec ist verheiratet, seine Frau ist Theologin.

(aus: Computer Zeitung, Nr.47, Nov. 1996)

Hans Moravec

Was uns erwartet, ist ... eine Zukunft, die man aus heutiger Sicht am ehesten mit „postbiologisch“ oder auch „übernatürlich“ bezeichnen kann. Heute sind unsere Maschinen noch einfache Geschöpfe. Doch im Laufe des nächsten Jahrhunderts werden sie zu Gebilden heranreifen, die ebenso komplex sind wie wir selbst, um schließlich über uns und alles, was wir kennen, hinauszuwachsen so dass wir eines Tages stolz sein dürfen, wenn sie sich als unsere Nachkommen bezeichnen.

Da diese Kinder unseres Geistes nicht auf den stockenden Gang der biologischen Evolution angewiesen sind, werden sie sich ungehemmt entfalten und sich gewaltigen Aufgaben von grundsätzlicher Bedeutung im größeren Universum zuwenden. Wir Menschen werden eine Zeitlang von ihrer Arbeit profitieren. Doch über kurz oder lang werden sie, wie biologische Kinder, ihre eigenen Wege gehen, während wir, ihre Eltern, alt werden und abtreten.

Wir sind dem Zeitpunkt schon sehr nahe, zu dem praktisch jede wichtige körperliche oder geistige Funktion des Menschen ihr künstliches Pendant haben wird. Die Verkörperung dieses Schnittpunkts vieler kultureller Entwicklungslinien wird der intelligente Roboter sein, eine Maschine, die wie der Mensch denken und handeln kann, mag sie ihm im materiellen oder intellektuellen Detail auch noch so unähnlich sein.

Früher oder später werden unsere Maschinen so klug sein, dass sie sich ohne fremde Hilfe instand halten, reproduzieren und vervollkommen können. Unsere Kultur wird dann in der Lage sein, sich unabhängig von der menschlichen Biologie und ihren Gesetzen zu entwickeln und wird statt dessen direkt von einer Maschinengeneration auf die nächste, noch leistungsfähigere, noch intelligentere, übergehen.

Moravec, Hans, Mind Children, Der Wettlauf zwischen menschlicher und künstlicher Intelligenz, Hamburg 1990

(aus: Rechenberg, Was ist Informatik)



### 1.3.6 Relativ harmlose Prognose 1988

Der Computer war oder ist oder wird sein:

- 1975 Prototyp der elektronischen Revolution
- 1980 Träger der künstlichen Intelligenz
- 1985 Universelle Experimentierhalle
- 1990 Intelligente Zelle in neuronal vernetzten Riesensystemen
- 1995 Informations-Normierer fast aller menschlichen Bereiche
- 2000 Universelles Organisationsgenie
- 2005 Partner des Menschen (in welcher Hinsicht auch immer)
- 2012 Prototyp des neuen (nichtmateriellen!) menschlichen Schöpfungswillens

Claus, V., Computer als was?, GMD Spiegel 4/1988

(aus: Rechenberg, Was ist Informatik)

### 1.3.7 Eine Äußerung zum „Computerunterstützten Unterricht“:

Es gibt einen weiten Konsens unter den Pädagogen, dass die meisten angebotenen Lehr- und Lernprogramme einfach schlecht sind, gewöhnlich kaum mehr als notdürftig aufgebesserte Videospiele, die einen gewissen optischen Reiz bieten, aber wenig intellektuelle Substanz.

Für Informatiker ist es zweifellos eine faszinierende Frage:

„Können wir eine Maschine erfinden, die all das kann, was ein Lehrer kann?“

Aber man könnte noch eine ganz andere Frage stellen:

„Warum sollten wir überhaupt eine Maschine erfinden, die das kann?“

Es war niemals schwierig, diese Frage zu beantworten, wo die Maschine schmutzige, gefährliche oder körperlich schwere Arbeit übernehmen sollte. Kinder zu unterrichten weist keines dieser Merkmale auf. Vielmehr scheint diese Tätigkeit zu jenen menschenwürdigen Verwendungsweisen menschlicher Wesen zu gehören, denen sich Menschen wohl zuwenden dürften, wenn die Roboter ihnen die Arbeit am Fließband ersparen würden.

Roszak, Th., Der Verlust des Denkens, Droemer-Knauer, München 1986

(aus: Rechenberg, Was ist Informatik, S.289)

### 1.3.8 Expertensysteme in der Medizin

Zum Einsatz von Expertensystemen in der Medizin erscheinen immer häufiger Artikel auch in den Tageszeitungen. Ebenso wie in der Gentechnologie erhebt sich auch hier die Frage nach der Ethik und nach den Grenzen der Forschung. Einige Beispiele sollen dies zeigen.

#### **KI trägt Dokortitel**

Nach einer schwedischen Studie übertrifft ein neues Diagnosesystem für Herzinfarkt auf Basis künstlicher Intelligenz die Kardiologenanalyse.

In die Studie wurden 1120 Elektrokardiogramme von Menschen mit Herzinfarkt und 10 452 normale EKGs einbezogen. Das mit statistischem Klinikmaterial trainierte System, das mit einem lernenden neuronalen Netz arbeitet, erreichte eine um zehn Prozent höhere Diagnosefähigkeit als das Herzspezialistenteam der schwedischen Universität.

„Das Ergebnis zeigt, dass dieses Entscheidungsunterstützungssystem ein hilfreiches Werkzeug sein kann“, kommentiert Jonathan Langberg, Direktor des Cardiac Electrophysiology Laboratory in Atlanta, die Studie.

Aus: Computerzeitung, Oktober 1997

Eine weitere Schlagzeile:

#### **Computer spielen Todesengel**

Aus: Badische Zeitung, 10.5.1997, mit Bezug auf ein Computer-Expertensystem „RIYADH“, das für Patienten auf der Intensivpflegestation die Überlebenschancen und die dazu nötigen Kosten bestimmt. Es ist „als Berater des Arztes“ konzipiert.



## 1.4 Einteilung der Informatik

Im Folgenden wird ein kurzer Überblick über eine mögliche Einteilung der Informatik gegeben. Dabei ist nicht immer unumstritten, wie weit die in der Spalte *Informatik und Gesellschaft* genannten Inhalte noch der Informatik im engeren Sinne zuzurechnen sind.

Technische Informatik	Praktische Informatik	Theoretische Informatik	Angewandte Informatik	Informatik und Gesellschaft
Hardwarekomponenten	Algorithmen Datenstrukturen Programmiermethoden	Automatentheorie	Informationssysteme	Selbstverständnis der Informatik
Schaltnetze		Theorie der Formalen Sprachen	Computergrafik	Einsatzbereiche
Schaltwerke	Programmiersprachen und Compiler	Theorie der Berechenbarkeit	Künstliche Intelligenz (Anwendungen)	- Arbeitswelt
Prozessoren		Komplexitätstheorie	Digitale Signalverarbeitung	- Staatlicher Bereich (Verwaltung, Militär, Gesundheitswesen)
				- Bildungsbereich
Mikroprogrammierung	Betriebssysteme	Algorithmenanalyse	Simulation und Modellierung	Informatik und Ökologie
Rechnerorganisation und -architektur	Softwaretechnik	Theorie der Programmierung	Textverarbeitung und Büro-Automation	Rechtliche Aspekte
Schnittstellentechnik und Rechnernetze	Mensch-Maschine Kommunikation – Software-Ergonomie	Automatische Programmsynthese	Spezifische Anwendungen in	- Datenschutz
			- Wirtschaft und Verwaltung	- Gesetze zur Informations- und Kommunikationstechnologie (z.B. IuKDG)
Messen, Steuern, Regeln	Netzwerktechnik	Formale Semantik	(e-Commerce, e-Government)	- Patentschutz, Urheberrechtsschutz
	Grundlagen der Multimediaetechnik	Grundlagen der Künstlichen Intelligenz	- Ingenieurwissenschaften	-
			- Naturwissenschaften	Technikfolgenabschätzung
			- Medizin	
			- Sozialwissenschaften	Ethik und Informatik
			- Geisteswissenschaften	
			- Kunst	
			- Bildung und Ausbildung (e-Learning, Learning on Demand)	Berufspraxis von Informatikern und Informatikerinnen
			Robotik	

### Behandlung der Themen im Angebot der Pädagogischen Hochschule:

In den Vorlesungen zur Informatik werden im Wesentlichen die Gebiete der *praktischen Informatik* (in Teil I und teilweise Teil II) sowie einige Themen aus der *technischen Informatik* (in Teil II) behandelt. Zur *angewandten Informatik* werden eigene Veranstaltungen angeboten (z.B. Tabellenkalkulation, Datenbanken, Graphiksysteme, Autorensysteme, Autorensysteme zur Entwicklung von Web basierten Lehr-Lernsystemen), daher folgt hier nur noch ein kurzer Überblick über diese Gebiete. Die *theoretische Informatik* ist sehr stark mathematisch orientiert und geht über das für die meisten Informatiker benötigte Wissen hinaus (Grundlagenforschung). Zu *Informatik und Gesellschaft* werden selten eigene Veranstaltungen angeboten, einige Themen werden bei Gelegenheit in der Vorlesung Informatik teilweise behandelt.



## 1.5 Anwendungen der Informatik: Übersicht

- Textverarbeitung, Publishing, Web-Publishing
  - Tabellenkalkulation
  - Datenbanksysteme
  - Geschäftsgraphik (Vektorgraphik)
- } Oft als integrierte Programme zusammengefasst  
(,Office Programme‘ , in der Schule oft ,Works‘)
- Graphikprogramme zur Bearbeitung von Photo (und Video)
  - Graphikprogramme zum Zeichnen (z.B. Corel Draw)
  - Multimediaproduktion (z.B. Filme)
- CAD    Computer Aided Design
  - CAM    Computer Aided Manufacturing
  - CIM    Computer Integrated Manufacturing
  - CASE    Computer Aided Software Engineering
- } C-Techniken
- Modellbildungssysteme, Simulationen (z.B. Flugsimulator, Wetter- und Klimamodelle, Windkanalsimulation)
  - Auskunftssysteme, On-Line Datenbanken
  - Telekommunikation (→Netze, E-Mail, Electronic Commerce, E-Government, Mobile Computing)
  - Anlagensteuerung, Prozess-Steuerung, Robotik („menschenleere Fabrik“)
  - Computer unterstützte Lehrsysteme , Trainingssysteme, E-Learning  
CAI    Computer Aided Instruction  
CBT    Computer Based Training  
Lehr-Lern-Plattformen (CLIX an den PHen in BW seit WS 2002)
  - Künstliche Intelligenz KI  
Artificial Intelligence AI
    - Expertensysteme
    - Robotik („Intelligente Roboter“)
    - Sprachverstehen, maschinelle Übersetzung
    - Computer-Sehen, Mustererkennung
    - Maschinelles Beweisen, Computer Algebra
    - Spielprogramme (Schach)
    - Lernen, Intelligente Tutorielle Systeme  
ICAI    Intelligent Computer Aided Instruction

In der KI gibt es zwei vollständig verschiedene Ansätze:

- Symbolische Darstellung und Verarbeitung von Wissen, Regelbasierte Systeme  
Klassischer Ansatz, seit den 50er Jahren
- Neuronale Netze  
in Analogie zur Verarbeitung von Information durch ein Netzwerk von Nervenzellen im menschlichen Gehirn, unterhalb der symbolischen Ebene.

In Verbindung mit beiden Ansätzen kommt seit den 70er Jahren eine unscharfe, nicht zweiwertige Logik zum Einsatz, die sogenannte **Fuzzy Logic** von Lotfi Zadeh zum Einsatz. Sie soll es erlauben, die eher verwaschenen menschlichen Begriffe besser zu modellieren ( wie z.B. den Begriff ,ziemlich warm‘) als die strikte zweiwertige Logik es vermag. Fuzzy Logic steuert im Alltag Geräte wie Heizungen, Photoapparate, Waschmaschinen usw.

### 1.5.1 Tabellenkalkulation mit Geschäftsgraphik

Tabellenkalkulationsprogramme stellen sogenannte *elektronische Rechenblätter* zur Verfügung. Dort werden in Tabellenform Daten zusammen mit Rechenoperationen eingetragen, etwa die Haushaltsausgaben einer Familie, gruppiert nach Gebieten Wohnen, Essen, Kleider, Luxus usw. zusammen mit den Summenoperationen für die Teilgebiete. Bei Änderung der Daten wird das Tabellenblatt automatisch auf den neuesten Stand gebracht. Meist bietet ein Tabellenkalkulationsprogramm auch die Möglichkeit, die Daten in Form einer Geschäftsgraphik (Kreisdiagramm, Balkendiagramm) darzustellen. Diese Graphiken werden bei Änderung der Daten ebenfalls angepasst.

Ausgaben im Monat Mai					
	Wohnen	Essen	Kleider	Luxus	Gesamt
	1.250,00 DM	155,30 DM	160,40 DM	177,77 DM	
	124,00 DM	12,50 DM	255,80 DM	212,00 DM	
	35,00 DM	45,70 DM	270,70 DM	15,00 DM	
		30,20 DM	12,40 DM	170,60 DM	
			56,74 DM		
Teilsumme	1.409,00 DM	243,70 DM	756,04 DM	575,37 DM	2.984,11 DM

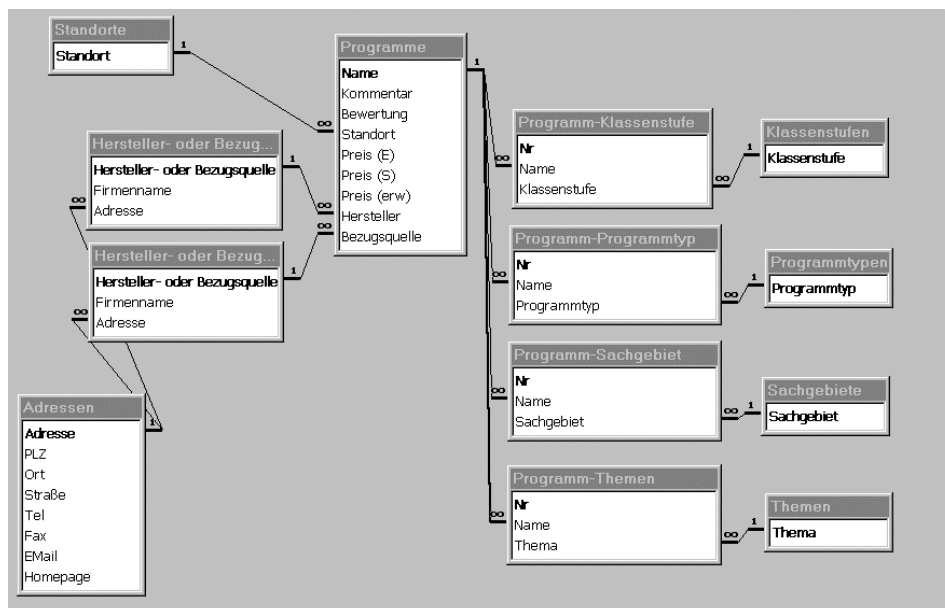
### 1.5.2 Datenbanksysteme

Einer (relationalen) Datenbank liegt eine Menge von Tabellen (ähnlich den Tabellen in Tabellenkalkulationsprogrammen) zugrunde, die untereinander verknüpft sein können (siehe Beispiel). Jede Tabelle ist eine Abfolge gleichartiger Datensätze. Standardbeispiel für einen solchen Datensatz sind die Daten eines Buches einer Bibliothek mit den Feldern Autor, Titel, Verlag, Erscheinungsjahr usw. Der Katalog der Universitätsbibliothek der Uni Freiburg z.B. liegt als Datenbank vor, die von den Computerterminals der PH-Bibliothek abgefragt werden kann (OLIX-System).

Ein Datenbanksystem enthält Hilfsmittel zur Erstellung, Verwaltung und Abfrage solcher Tabellen sowie zur Darstellung der Ergebnisse von Abfragen in gut lesbarer Form (Reports).

Auch mit Tabellenkalkulationsprogrammen lassen sich schon einfache Datenbankanwendungen realisieren. Richtet man z.B. in einem integrierten System (MS Office oder MS Works) eine Datenbank mit Adressen ein, so lassen sich damit Serienbriefe erstellen.

**Beispiel:** Datenbank zur Erfassung von Software zum Mathematikunterricht im Datenbanksystem MS Access. Schematische Darstellung der Verknüpfung der Tabellen.




## 2 Daten und ihre Codierung

### 2.1 Codierung im Binärsystem

Die Grundlage der Speicherung und der Verarbeitung von Daten im Rechner und auf Platten ist die Codierung im Binärsystem. Das heißt, alle Daten sind durch eine Folge von 0 und 1 dargestellt. 0 und 1 werden dabei durch zwei verschiedenartige physikalische Zustände, je nach Gerät, realisiert. Diese physikalische Umsetzung ist natürlich für einen Benutzer eines Gerätes nicht sichtbar und auch im allgemeinen nicht wichtig.

Beispiele für Binärdarstellungen:

Reihe von Lämpchen:      An oder aus:            Folge      **100101**

Elektrische Schalter:      Offen oder geschlossen.  
 Platten und Disketten:      Verschiedene Magnetisierungszustände.  
 Speicher im Rechner:      Verschiedene Spannungszustände (z.B. 0 V oder 5 V).

**Bits:** Die Menge an **Information**, die mit einer 0 oder 1 gespeichert werden kann heißt **1 Bit**.  
 Die Lämpchenreihe oben umfaßt 6 Bit.

**Bytes:** **8 Bits** werden zu einem **Byte** zusammengefasst. Der Speicher im Rechner, auf Festplatten und CD ROMs ist in Paketen zu je 8 Bits organisiert. Daher ist 1 Byte die Grundlage für die Codierung von Zeichen (wie Buchstaben) und Farben im PC.

Speichergrößen werden in Bytes angegeben.

Der Umfang von großen Speichern wird in größeren Einheiten angegeben:

1 Kilobyte	= 1024 Byte	= 1 KB	≈ 10 <sup>3</sup> Byte.
1 Megabyte	= 1024 Kilobyte	= 1 MB	≈ 10 <sup>6</sup> Byte.
1 Gigabyte	= 1024 Megabyte	= 1 GB.	≈ 10 <sup>9</sup> Byte.
1 Terabyte	= 1024 Gigabyte	= 1 TB.	≈ 10 <sup>12</sup> Byte.

#### 2.1.1 Speicherkapazitäten von Speichermedien: (2003)

Festplatten (in PCs):	ca. 60 GB bis 300 GB und mehr.	
CD ROMs (auch beschreibbar):	ca. 700 MB	(Preis ca.0,30 €)
DVD (Digital Video Disc):	4 - 4.7 GB (Ausbau bis zu 17 GB geplant)	
iomega Zip Disketten	100 - 750 MB	(Preis ca. 10-15 €)
Disketten 3½“ DD	740 KB	(nicht mehr gebräuchlich)
Disketten 3½“ HD	1.4 MB	(ca. 0.20 €)
Flash Memory Sticks (USB Sticks)	64 – 512 MB (bis 2GB)	(50 - 100 €) ( 2 GB 700€)
Interner Speicher PC (Arbeitsspeicher)	128 MB-512 MB	

#### Aufgaben:

Wie viele verschiedene Codes kann man mit 1 Byte, 2 Bytes bilden?

Ein Buchstabe wird üblicherweise durch 1 Byte codiert.

Wie viele Bytes braucht man für

- eine DIN A4 Seite reinen Text

- ein Manuskript mit 300 Seiten

Wie viele Seiten Text passen auf

- eine 3½“ HD Diskette?

- eine 1 GB Festplatte?

Ein Bild wird auf einem Punkteraster von 600 x 400 (1024 x 768) Punkten dargestellt (z.B. Bildschirm). Überlegen Sie Methoden, ein Bild binär zu verschlüsseln!

Wie viele Bytes braucht man bei Ihrer Methode

- für ein schwarz-weiß Bild?

- für ein Farbbild?

Da Bilddateien meist komprimiert werden, passen in der Praxis mehr Bilder auf ein Speichermedium, als hier berechnet wird.

### 2.1.2 Übertragungsraten von Daten (2003)

- **Modems:**

Üblich sind 14 400 Bit/s (sehr langsam, ungebräuchlich)  
28 800 Bit/s  
56 000 Bit/s

Mit Kompression sind höhere Übertragungsraten möglich.

- **ISDN:**

64 000 Bit/s (= 8 KB/s) pro Kanal

- **ADSL:**

Asymmetric Digital Subscriber Line (Telecom Deutschland TDSL). Asymmetrische Übertragung, also je nach Richtung unterschiedlich schnell. Zum Endanwender maximal 8 MBit/s, zur Vermittlungsstelle maximal 1 MBit/s.

- Übertragungsraten im Internet

- mit 14 400 Bit/s Modem (= 1.8 KB/s):  
1.4 KB/s, oft aber auch weniger,
- mit 56 000 Bit/s Modem (= 7 KB/s):  
4,5 KB/s, oft aber auch weniger.

- Große Datenleitungen (oft Glasfaser):

100 Mbit/s bis 600 Mbit/s (Verbindungen der großen Rechenzentren untereinander)  
Rekord: 3,2-Terabit/s (WorldCom Inc. und Siemens/Optisphere Networks, April 2001)

## 2.2 Zahlen im Dezimal-, Dual-, und Hexadezimalsystem

### 2.2.1 Dualsystem

In Analogie zum dezimalen Zahlensystem mit den Ziffern 0, 1, 2, ..., 9 können Zahlen auch im **Dualsystem** (Binärsystem) mit den Ziffern 0 und 1 dargestellt werden. An die Stelle der Potenzen  $10^0, 10^1, 10^2, 10^3, \dots$  der Basis 10 treten die Potenzen  $2^0, 2^1, 2^2, 2^3, \dots$  der Basis 2.

Formal ausgedrückt:

Jede natürliche Zahl  $z$  läßt sich im Dualsystem als Summe von Zweierpotenzen mit Koeffizienten 0 oder 1 darstellen, d.h.

$$z = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \dots + a_1 \cdot 2^1 + a_0 \cdot 2^0, \quad a_i \in \{0, 1\} \text{ für alle } i.$$

Diese Darstellung ist eindeutig.

Beispiel:

$$\begin{aligned} 27 &= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 \end{aligned}$$

Darstellung in einer Stellenwerttafel

$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$											dezimal
128	64	32	16	8	4	2	1		128	64	32	16	8	4	2	1		
								=										=
		1	0	0	1	0	1	=			32			+4		+1		= 37
1	0	0	1	1	0	0	0	=	128			+16	+8					= 152
1	1	1	1	1	1	1	1	=										=
								=			32	+16	+8				+1	= 57
								=										= 127
								=										= 63
								=										= 0
								=										= 24
								=										= 48
								=										= 12

Die letzten drei Beispiele zeigen, dass im Dualsystem die Multiplikation und Division mit 2 einfach durch Verschieben nach links bzw. nach rechts um eine Stelle geschieht, analog zur Multiplikation und Division mit 10 im Dezimalsystem.

Man verwendet einen Index, um zu unterscheiden, in welchem Zahlensystem eine Zahl interpretiert werden soll, z.B.

$$\begin{aligned} 27_{10} &= 11011_2 = 11011_B \\ 100_{10} &= 1100100_2 \end{aligned}$$

### 2.2.2 Hexadezimalsystem

Zahlen im Dualsystem sind wegen der vielen Ziffern sehr schwierig zu lesen, die Umrechnung von Hand ist langwierig. Daher ist in der Informatik auch das **Hexadezimalsystem** (16er-System) sehr verbreitet. Die Zahlen sind wie im Dezimalsystem gut lesbar, und die Umsetzung ins Dualsystem ist dazu außerordentlich einfach.

Die Basis des Hexadezimalsystems ist 16, man benötigt daher 16 Ziffern. Man verwendet dazu die Ziffern von 0 bis 9 und für die fehlenden „Ziffern“ 10, 11, ... die Buchstaben A, B, ...:

Hex-Ziffer	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Wert	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Formal kann man analog zum Dualsystem feststellen:

Jede natürliche Zahl  $z$  läßt sich im Hexadezimalsystem als Summe von 16er-Potenzen mit Koeffizienten 0, 1, 2, ..., 9, A, B, ..., F darstellen, d.h.

$$z = a_n \cdot 16^n + a_{n-1} \cdot 16^{n-1} + a_{n-2} \cdot 16^{n-2} + \dots + a_1 \cdot 16^1 + a_0 \cdot 16^0, \quad a_i \in \{0, 1, 2, \dots, 9, 10, 11, \dots, 15\} \text{ für alle } i.$$

Diese Darstellung ist eindeutig.

Man verwendet zur Notation im Hexadezimalsystem den Index  $H$  oder  $16$ , oder man stellt der Zahldarstellung ein H voran:

$$27_{10} = 1 \cdot 16^1 + 11 \cdot 16^0 = 1B_{16} = 1B_H = H1B$$

Darstellung in einer Stellenwerttafel

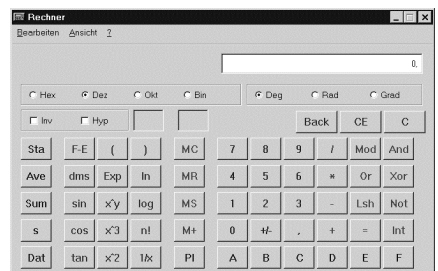
$16^4$	$16^3$	$16^2$	$16^1$	$16^0$								dezimal
65 536	4 096	256	16	1		65 536	4 096	256	16	1		
					=						=	
3	0	A	0	E	=	3·65 536		+10·256		+14·1	=	199 182
0	0	F	A	1	=			15·256	+10·16	+1·1	=	4 001
F	F	F	F	F	=	15·65 536	+15·4 096	+15·256	+15·16	+15·1	=	1 048 575
			3	9	=						=	
					=						=	127
					=						=	63
					=						=	0
					=						=	100
					=						=	1600
					=						=	

### 2.2.3 Umrechnung von Dualsystem ins Hexadezimalsystem und umgekehrt.

Die einfache Umrechnung ins Dualsystem und umgekehrt ergibt sich daraus, dass man die Ziffern der Hexadezimaldarstellung unmittelbar aus der Dualdarstellung ablesen kann.

Hex-Ziffer	Dezimaler Wert	Dualdarstellung
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadezimalzahlen mit dem Taschenrechner:



Die meisten Taschenrechner (auch z.B. der unter Windows / „Zubehör“ verfügbare Rechner, s.Bild) besitzen einfache Tasten zur Umrechnung zwischen den Zahlssystemen. Man wählt etwa die Darstellung „hexadezimal“, gibt eine Hexadezimalzahl ein und schaltet um auf die Darstellung „dezimal“. Der Rechner zeigt dann die eingeebene Zahl in Dezimaldarstellung an. Taschenrechner können auch direkt im Hexadezimalsystem rechnen.

Dualsystem  $1001 \ 1100 \ 0101 \ 1011_2 = \dots$ schrecklich, das hinschreiben !!!!

Hexadezimalsystem  $9 \ C \ 5 \ B_{16} = 9 \cdot 16^3 + 12 \cdot 16^2 + 5 \cdot 16^1 + 11 \cdot 16^0$   
 $= 9 \cdot 4\ 096 + 12 \cdot 256 + 5 \cdot 16 + 11 \cdot 1$   
 $= 36\ 864 + 3\ 072 + 80 + 11$   
 $= 40\ 027_{10}$

Kurz  $1001 \ 1100 \ 0101 \ 1011_2 = 9C \ 5B_{16} = 40\ 027_{10}.$

Für jede Hexadezimalziffer benötigt man 4 Bit, mit einem Byte codiert man also zwei Hexadezimalziffern. Die höchste mit einem Byte darstellbare natürliche Zahl ist also

$$1111 \ 1111_2 = FF_{16} = 15 \cdot 16 + 15 = 16^2 - 1 = 255_{10}$$

## 2.2.4 Rechnen im Dual- und Hexadezimalsystem

Im Dual- und Hexadezimalsystem können alle schriftlichen Rechenverfahren genauso durchgeführt werden wie im Dezimalsystem. Der Überschreitung von 10 im Dezimalsystem entspricht im Dualsystem die Überschreitung von 2 und im Hexadezimalsystem die Überschreitung von 16. Beim schriftlichen Rechnen verfährt man mit den Überträgen genauso wie im Dezimalsystem. Je ein Beispiel für die Addition soll das verdeutlichen.

Dezimalsystem:

$$\begin{array}{r} 7\ 2\ 5\ 4 \\ 5\ 4\ 6\ 8 \\ \text{Übertrag} + \frac{1}{1\ 2\ 7\ 2\ 2} \end{array}$$

Dualsystem

$$\begin{array}{r} 1\ 1\ 1\ 1\ 15_{10} \\ 1\ 1\ 0\ 6_{10} \\ \text{Übertrag} + \frac{1\ 1\ 1}{1\ 0\ 1\ 0\ 1} \end{array}$$

Hexadezimalsystem

$$\begin{array}{r} F\ 3\ C\ A\ 5\ 9\ 985\ 65_{10} \\ 1\ 7\ D\ 8\ 7\ 97\ 671_{10} \\ \text{Übertrag} + \frac{1\ 1\ 1}{1\ 0\ B\ A\ 2\ C} \end{array}$$

Überprüfen Sie die Rechnungen von Hand und mit Hilfe eines Taschenrechners.

### Übungen:

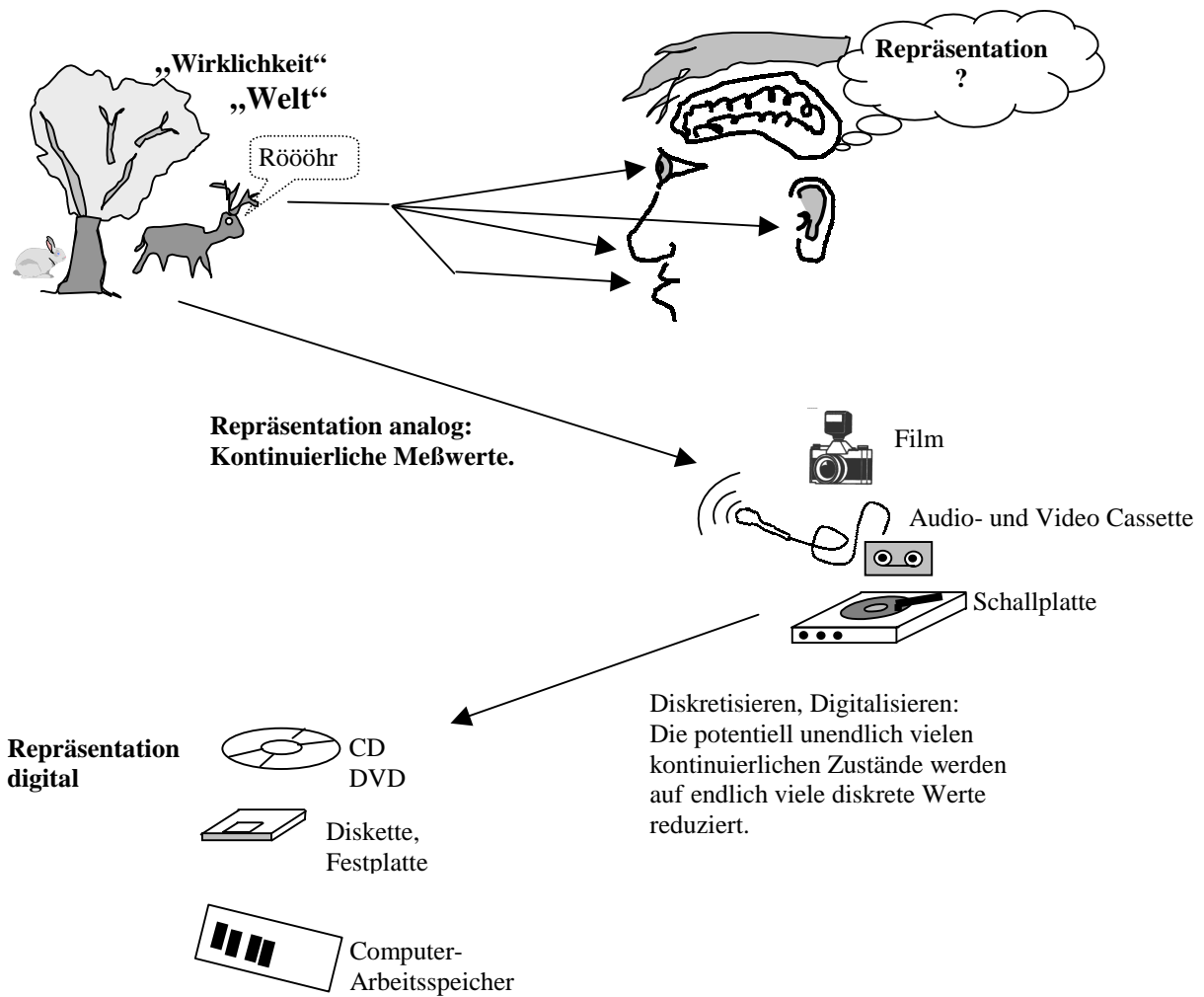
1. Addieren Sie jeweils 1 im jeweiligen Zahlensystem.

$$\begin{array}{l} 1234_{16} \quad BFDF_{16} \quad 1111_{16} \quad 9FFF_{16} \quad FFFF_{16} \quad 999F_{16} \\ 1000_2 \quad 1101_2 \quad 1111_2 \quad 101111_2 \end{array}$$

2. Berechnen Sie im jeweiligen System die Summen und überprüfen Sie das Ergebnis durch Umwandlung und Rückumwandlung ins Dezimalsystem (oder mit Hilfe eines Taschenrechners).

$$\begin{array}{l} 10_{16} + AD_{16} \quad FF_{16} + 9B_{16} \quad 1FFF_{16} + AD9B_{16} \quad 1234_{16} + A34_{16} \\ 1001_2 + 1111_2 \quad 1111_2 + 111_2 \quad 1000_2 + 1010_2 \end{array}$$

## 2.3 Das Abbild der Welt im Computer



Um mit dem Computer Daten aus der realen Welt verarbeiten zu können, werden diese Daten zunächst digitalisiert. Das bedeutet, dass aus den kontinuierlichen Messwerten, die z.B. ein Photoapparat erfasst und auf einem Film festhält oder ein Mikrophon aufnimmt und auf einer Audio-Kassette speichert, eine Folge von endlich vielen diskreten Werten erzeugt wird, die dann nochmals in eine Codierung im Dualsystem umgesetzt werden. Beispielsweise wird die Schallstärke einer Schallquelle zunächst in endlich viele Intervalle aufgeteilt, denen jeweils ein Wert zugeordnet ist. Ein solcher Wert kann dann im Dualsystem repräsentiert werden. Die Intervalle müssen so fein gewählt werden, dass sich das ursprüngliche Signal wieder mit guter Näherung aus den diskreten Werten rekonstruiert werden kann.

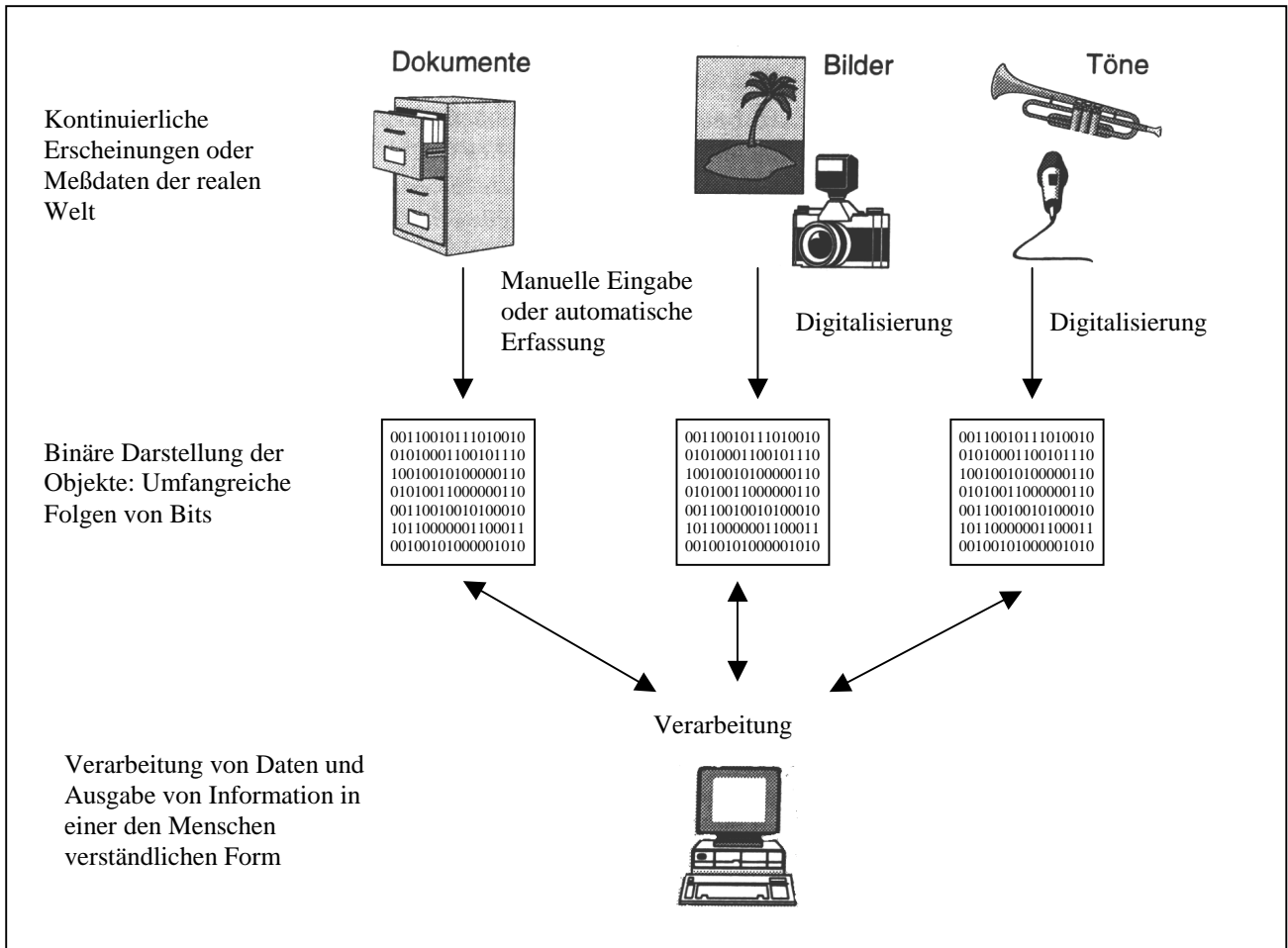
Vorteile der Digitalisierung:

- Die Dualcodierungen können schrittweise durch eindeutige Verfahren manipuliert werden.
- Die diskreten Werte sind weitgehend unempfindlich gegen Verfälschungen.

Nachteile der Digitalisierung

- Wenn die Intervalle, innerhalb derer die kontinuierlichen Werte zusammengefasst werden, zu grob sind, dann ergeben sich bei der Rekonstruktion der Ausgangsdaten Verfälschungen.
- Die Verarbeitung erfordert oft einen großen Rechenaufwand und ist daher sehr langsam (z.B. Videoclips im Computer).





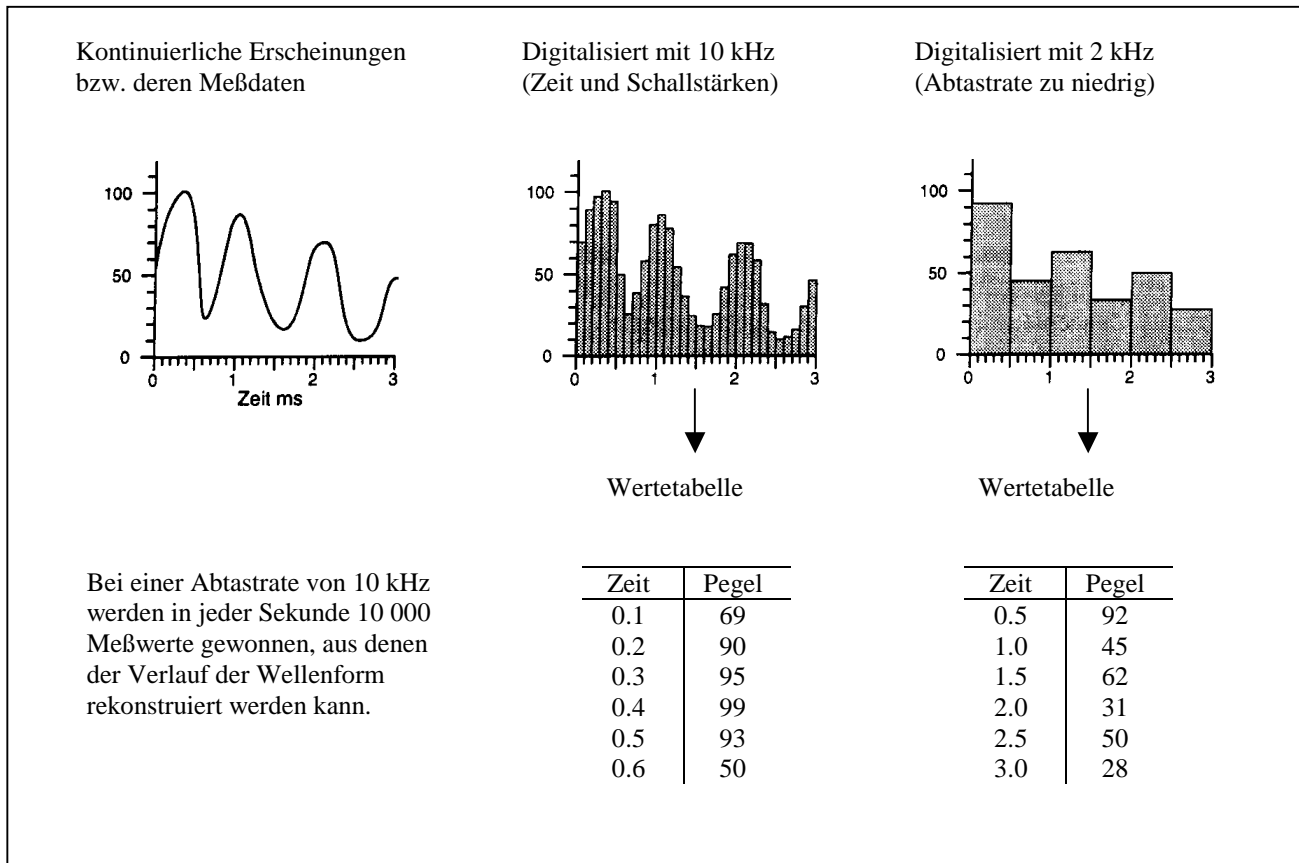
### Beispiel der Digitalisierung eines Audiosignals.

Im folgenden Beispiel könnte das Signal von einem Mikrophon stammen. Hier wird sowohl die Zeit als auch die Schallstärke diskretisiert.

Im ersten Diagramm zur Diskretisierung findet man feine Zeitintervalle der Länge 0,1 Millisekunden und feine Pegel-Intervalle (der Bereich von 0 bis 100 wird in 100 Intervalle aufgeteilt).

Im zweiten Diagramm zur Diskretisierung findet man grobe Zeitintervalle der Länge 0,5 Millisekunden und offenbar die gleichen feinen Pegel-Intervalle.

Es wäre auch möglich, bei gleicher Diskretisierung der Zeit, die Pegelintervalle zu vergrößern.



### Digitalisierung analoger Audio-Daten

Zwei Beispiele sollen eine Vorstellung geben, welche Zahlenwerte hier auftreten.

- **ISDN:**
  - Abtastrate 8000 mal/s
  - Digitalisierung des Schalldrucks 8 Bit Code (= 256 Werte)
  - Übertragungsrate/Kanal 64 000 Bit/s
- **Audio CD:**
  - Abtastrate 44 100 mal/s
  - Digitalisierung des Schalldrucks 16 Bit Code (=  $2^{16}$  Werte = 65536 Werte)
  - Übertragungsrate/Kanal 705 600 Bit/s

### Dateiformate von Grafikdateien und Audiodateien

Sollen Grafik- oder Audio-Informationen gespeichert werden, so werden jeweils zwei prinzipiell verschiedene Arten der Codierung.

1. Reale analoge Daten werden digitalisiert und gespeichert, so wie dies soeben beschrieben wurde. Bei *Grafiken* werden Informationen über die einzelnen Bildpunkte (Pixel) festgehalten, man spricht daher von Pixelgrafiken. Je nach darauf noch angewandtem Kompressionsverfahren spricht man dann von BMP, TIF, GIF, JPG ....-Dateien. Bei *Audiodateien* werden Informationen über die einzelnen Abtastwerte festgehalten. Je nach darauf noch angewandtem Kompressionsverfahren erhält man WAV, MP3, ... -Dateien.
2. Wird der Erzeugungsprozess der Bild- oder Audiodaten auf einem technischen Gerät wie dem Computer oder einem Synthesizer aufgezeichnet, erhält man viel kleinere Dateien. *Grafiken* können z.B. mit einem sogenannten Zeichenprogramm (z.B. Corel Draw) erstellt werden. Sie werden dabei aus Grundobjekten wie Rechtecken, Kreisen, Linien verschiedener Farben und Dicken aufgebaut. Wird die Beschreibung dieses Prozesses gespeichert, erhält man sogenannte Vektorgrafik-Dateien, die als Kennung je nach Erzeugungs- bzw. Speicherungsart eine charakteristische Erweiterung erhalten. Beispiele sind die Formate WMF, CDR .... *Audiodateien* aus dem Synthesizer oder aus Computerprogrammen werden z.B. im sogenannten MIDI-Format gespeichert, wobei einfach Daten wie Tonhöhe, Dauer, Instrument, Anschlagsart usw. von Tönen gespeichert werden.

## 2.4 Datentypen

Alle Speicher (im Rechner RAM und ROM sowie der Speicher auf Platten, CDROMs und Disketten) kann man sich vorstellen als eine lineare Folge von Bits, zusammengefasst als Bytes oder Words (=2 Bytes), die gemeinsam als Block übertragen und verarbeitet werden. Jedes Byte hat dabei eine Adresse, die angibt, um das wievielte Byte in der Folge es sich handelt. Die technische Realisierung von Speicherung und Übertragung soll hier nicht weiter behandelt werden (→ Informatik II).

### 2.4.1 Speichermodell: Folge von Bytes

Adresse (dezimal)	Inhalt	
	0	A
1	3	C
2	F	F
3	3	9
4	C	2
5	4	E
.	.	.
.	.	.
.	.	.
.	.	.

Adressen werden oft auch hexadezimal angegeben.

Wie die Daten dabei interpretiert werden sollen, muß man natürlich festlegen. Im folgenden werden einige grundlegende Konzepte für eine solche Codierung beschrieben. In den Anfängen der Datenverarbeitung waren im wesentlichen Zahlen und eng damit zusammenhängende Strukturen die zu verarbeitenden Daten, heute spielen zusätzlich noch vielerlei andere Daten eine große Rolle. Die verschiedenen Typen von Daten werden als **Datentypen** bezeichnet.

### 2.4.2 Zahltypen

Statt die Folge von Bytes untereinander zu schreiben werden sie der besseren Lesbarkeit wegen jetzt nebeneinander gesetzt.

#### 2.4.2.1 Positive ganze Zahlen

Speicherplatz	Bereich der möglichen Werte	Name des Datentyps in TURBO Pascal
1 Byte	0 bis 255 ( $= 2^8 - 1$ )	Byte
2 Bytes	0 bis 65535 ( $= 2^{16} - 1$ )	Word

Beispiele:

1	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---

1 Byte, Wert dezimal:  
Wert hexadezimal:

1	0	0	0	0	0	0	0	1	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

2 Byte, Wert dezimal:  
Wert hexadezimal:

Mit dieser Darstellung von Zahlen kann man leicht Addieren, Multiplizieren und Dividieren .

#### 2.4.2.2 Ganze Zahlen

Speicherplatz	Bereich der möglichen Werte	Name des Datentyps in TURBO Pascal
1 Byte	-128 bis 127 ( $-2^7$ bis $2^7 - 1$ )	Shortint
2 Bytes	- 32768 bis 32767 ( $-2^{15}$ bis $2^{15} - 1$ )	Integer

Dabei soll jeweils das erste Bit zur Darstellung des Vorzeichens dienen: 0 bedeutet „+“, 1 bedeutet „-“. Diese Codierung soll noch etwas genauer betrachtet werden. Um die Zahlen klein zu halten, legen wir statt 8 oder 16 Bit nur einen Speicherplatz von 4 Bit zugrunde. Die angegebene Darstellung heißt **Zweierkomplement-Darstellung**.

Code	$2^3$	$2^2$	$2^1$	$2^0$	Tatsächliche Bedeutung
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

Einfachste mögliche Bedeutung wäre	Zur tatsächlichen Bedeutung
	0
	1
	2
	3
	4
	5
	6
	7
	16 - 8
	16 - 7
	16 - 6
	16 - 5
	16 - 4
	16 - 3
	16 - 2
	16 - 1

Diese zunächst seltsam anmutende Darstellung negativer Zahlen bringt einen großen Vorteil bei der Subtraktion von Zahlen:

1. Die Umwandlung von x in -x ist im Dualsystem ganz einfach:

0101 → 1010 + 1 = 1011      Alle Nullen in Einsen und umgekehrt verwandeln und 1 addieren  
 +5 → -5

1011 → 0100 + 1 = 0101      Alle Nullen in Einsen und umgekehrt verwandeln und 1 addieren  
 -5 → +5

2. Die Subtraktion kann so in eine Addition überführt werden:

Zunächst im Dezimalsystem. Statt 6 - 5 rechnet man 6 + (16 - 5) und „vergißt danach den Überlauf 16“:

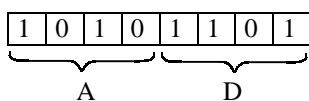
$$6 - 5 \equiv 6 + (16 - 5) \equiv 6 + 11 \equiv 17 \equiv 1$$

Im Dualsystem:

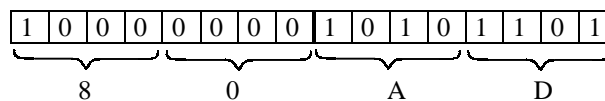
$$0110 - 0101 \equiv 0110 + 1011 \equiv 10001 \equiv 0001$$

Nebenrechnung	0110
	+1011
	-----

Beispiele:



1 Byte, Wert hexadezimal: AD  
 Wert dezimal: 173  
 Interpretation:  $-(2^8 - 173) = -(256 - 173) = -90$



2 Byte, Wert hexadezimal: 80AD  
 Wert dezimal: 32914  
 Interpretation:  $-(2^{16} - 32914) = -(65536 - 32914) = -32595$

**2.4.2.3 Gleitkommazahlen**

(Fließkommazahlen, floating point numbers, Real-Zahlen,)

**Dezimaldarstellung**

-251,781 = -2.51781 · 10<sup>+2</sup>      normierte Darstellung,  
 0,000251 = 2.51 · 10<sup>-3</sup>      auf Taschenrechnern oft im Modus ‚Sci‘ (scientific) verwandt

Will man eine Zahl in normierter Weise speichern, dann genügt es zu speichern:

- das Vorzeichen der Zahl
- die Ziffernfolge (die sogenannte Mantisse)
- den Exponenten mit seinem Vorzeichen

**Dualdarstellung**

Wie im Dezimalsystem können Brüche auch im Dualsystem dargestellt werden, die Nenner sind dabei Zweierpotenzen statt Zehnerpotenzen. Einige Beispiele sollen genügen.

Dualbrüche

2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>-1</sup>	2 <sup>-2</sup>	2 <sup>-3</sup>	2 <sup>-4</sup>	bedeutet dezimal								
8	4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	8	4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	
	1	0	1	0	1				4	+0	+1	+0	$+\frac{1}{4}$			= 5.25
			1	0	1	0	1				1	+0	$+\frac{1}{4}$	+0	$+\frac{1}{16}$	= 1.3125
1	0	1	0	1				8	+0	+2	+0	$+\frac{1}{2}$				= 10.5

Diese Beispiele zeigen auch, dass im Dualsystem die Multiplikation und Division mit Zweierpotenzen einfach durch verschieben des ‚Dualkommata‘ geschieht.

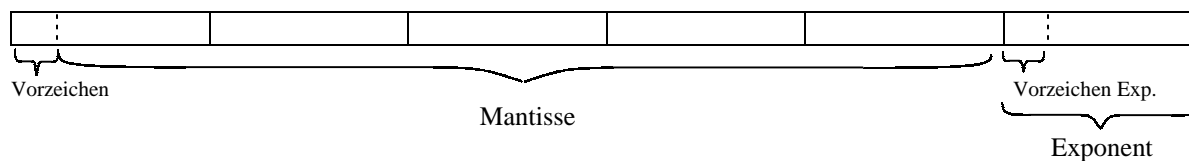
Die normierte Darstellung sieht genauso aus wie im Dezimalsystem:

-1010,1 = -1.0101 · 2<sup>+3</sup>      normierte Darstellung

Will man eine Dualzahl in normierter Weise speichern, dann genügt es zu speichern:

- das Vorzeichen der Zahl
- die Ziffernfolge (die Mantisse, ohne die führende 1, die stets an der Stelle stehen muß)
- den Exponenten zur Basis 2 mit seinem Vorzeichen

Damit könnte eine in 6 Bytes gespeicherte Gleitkommazahl etwa folgendermaßen aussehen:



**Gleitkommazahlen in Turbo-Pascal (nur Beispiele)**

Speicherplatz	Bereich der möglichen Werte	Genauigkeit	Name des Datentyps in TURBO Pascal
6 Bytes	$\pm 2.9 \cdot 10^{-39}$ bis $1.7 \cdot 10^{38}$	11 – 12 Stellen	Real
8 Bytes	$\pm 5.0 \cdot 10^{-324}$ bis $1.7 \cdot 10^{308}$	15 – 16 Stellen	Double

- Gleitkommazahlen sind im allgemeinen nicht genau, auch wenn sie genaue dezimale Zahlen darstellen (z.B. 0.1<sub>10</sub> wird im Dualsystem periodisch : 0.0001100110011....., kann also nicht exakt mit endlich vielen Stellen dargestellt werden).
- Verschiedene Zahltypen werden in völlig verschiedener Weise gespeichert und sind daher nicht ohne weiteres miteinander verträglich.

### 2.4.3 Datentyp Boolean (Wahrheitswert)

Für die beiden Wahrheitswerte *Wahr (True)* und *Falsch (False)* genügt eigentlich ein Bit, meist verwendet man aber wegen der einfacheren Adressierung ein Byte. Man kann etwa *Wahr* mit 1 und *Falsch* mit 0 codieren. Für die Wahrheitswerte sind die Booleschen Operationen AND, OR, NOT sowie eventuell weitere Operationen definiert.

Wahrheitswerte treten als Werte bei der Auswertung von Ausdrücken wie  $x > 3$ ,  $i=5$ ,  $(x+2)^4/z < m$  auf. AND, OR, NOT können benutzt werden, um aus solchen elementaren Ausdrücken komplexere Ausdrücke zusammensetzen, wie z.B.

$(x > 3) \text{ AND } (x < 10) \text{ AND NOT } (x=5)$ .

Die Operationen werden AND, OR, NOT werden durch die folgenden Tafeln definiert:

AND	Wahr	Falsch
Wahr	Wahr	Falsch
Falsch	Falsch	Falsch

OR	Wahr	Falsch
Wahr	Wahr	Wahr
Falsch	Wahr	Falsch

NOT	Wahr
Wahr	Falsch
Falsch	Wahr

### 2.4.4 Datentypen für Zeichen und Zeichenketten

Zeichen für „normalen“ Text werden oft durch 1 Byte nach der folgenden Tabelle codiert (ASCII Tabelle, für American Standard Code for Information Interchange). Eine erweiterte Tabelle bis Codennummer 255 für nationale Sonderzeichen wie *ü, ö, ß* usw. ist in Gebrauch. Diese erweiterten Tabellen unterscheiden sich oft in der Codierung mancher Zeichen mit Codennummer >127. Neben dem älteren IBM-ASCII findet man heute meist den sogenannten ANSI-ASCII-Code.

Die Zeichen mit den Codes von 0 bis 31 werden als Steuerzeichen bezeichnet und zur Steuerung von Geräten (z.B. Drucker) benutzt. Die Namen der Steuerzeichen gehen auf die Zeit der Fernschreiber zurück.

Eine Codierung eines sehr viel umfangreicheren Zeichensatzes stellt der *Unicode* dar, bei dem ein Zeichen durch 2 Bytes dargestellt wird. In vielen neueren Anwendungen werden Texte mit diesem Code dargestellt (z.B. MS Word).

Dec	Hex	Char	Description	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	(null)	32	20		64	40	@	96	60	`
1	1	SOH	(start of heading)	33	21	!	65	41	A	97	61	a
2	2	STX	(start of text)	34	22	"	66	42	B	98	62	b
3	3	ETX	(end of text)	35	23	#	67	43	C	99	63	c
4	4	EOT	(end of transmission)	36	24	\$	68	44	D	100	64	d
5	5	ENQ	(enquiry)	37	25	%	69	45	E	101	65	e
6	6	ACK	(acknowledge)	38	26	&	70	46	F	102	66	f
7	7	BEL	(bell)	39	27	'	71	47	G	103	67	g
8	8	BS	(backspace)	40	28	(	72	48	H	104	68	h
9	9	TAB	(horizontal tab)	41	29	)	73	49	I	105	69	i
10	A	LF	(line feed, new line)	42	2A	*	74	4A	J	106	6A	j
11	B	VT	(vertical tab)	43	2B	+	75	4B	K	107	6B	k
12	C	FF	(form feed, new page)	44	2C	,	76	4C	L	108	6C	l
13	D	CR	(carriage return)	45	2D	-	77	4D	M	109	6D	m
14	E	SO	(shift out)	46	2E	.	78	4E	N	110	6E	n
15	F	SI	(shift in)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	(data link escape)	48	30	0	80	50	P	112	70	p
17	11	DC1	(device control 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2	(device control 2)	50	32	2	82	52	R	114	72	r
19	13	DC3	(device control 3)	51	33	3	83	53	S	115	73	s
20	14	DC4	(device control 4)	52	34	4	84	54	T	116	74	t
21	15	NAK	(negative acknowledge)	53	35	5	85	55	U	117	75	u
22	16	SYN	(synchronous idle)	54	36	6	86	56	V	118	76	v
23	17	ETB	(end of trans. block)	55	37	7	87	57	W	119	77	w
24	18	CAN	(cancel)	56	38	8	88	58	X	120	78	x
25	19	EM	(end of medium)	57	39	9	89	59	Y	121	79	y
26	1A	SUB	(substitute)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	(escape)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	(file separator)	60	3C	<	92	5C	\	124	7C	
29	1D	GS	(group separator)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	(record separator)	62	3E	>	94	5E	^	126	7E	~
31	1F	US	(unit separator)	63	3F	?	95	5F	_	127	7F	DEL

ASCII Tabelle für die Codes von 0 bis 127<sub>10</sub> (0 bis 7F<sub>16</sub>)

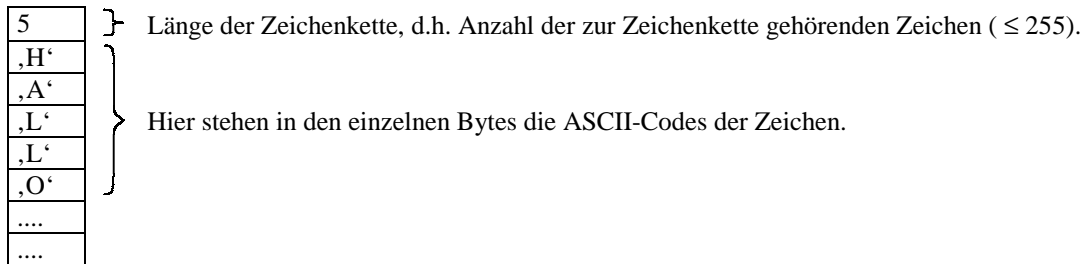
2.4.4.1 Zeichen (Character)

Speicherplatz	Bereich der möglichen Werte	Name des Datentyps in TURBO Pascal
1 Byte	Zeichen der ASCII-Tabelle	Char

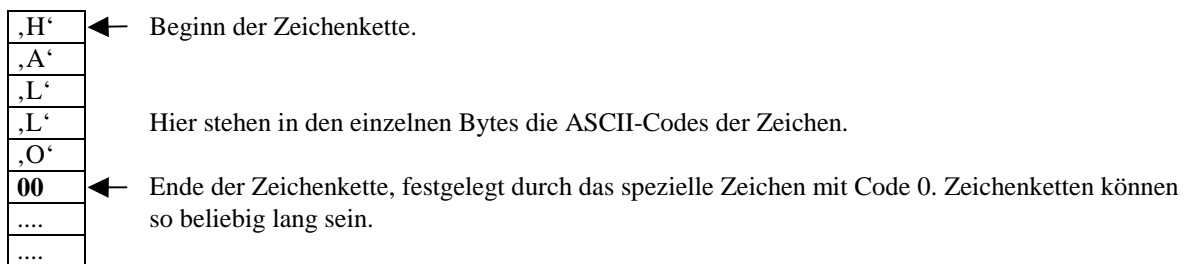
2.4.4.2 Zeichenketten (String)

Unformatierter Text kann als Kette aufeinanderfolgender Zeichen aufgefaßt werden. Eine solche Folge nennt man ‚Zeichenkette‘, engl. ‚string‘.

Es finden sich in den Programmiersprachen verschiedene Varianten der Codierung:



Das ist die Art der Speicherung von TURBO-Pascal (ältere Versionen).



Das ist die Art in der C die Zeichenketten speichert (mittlerweile auch in TURBO-Pascal als Alternative übernommen).

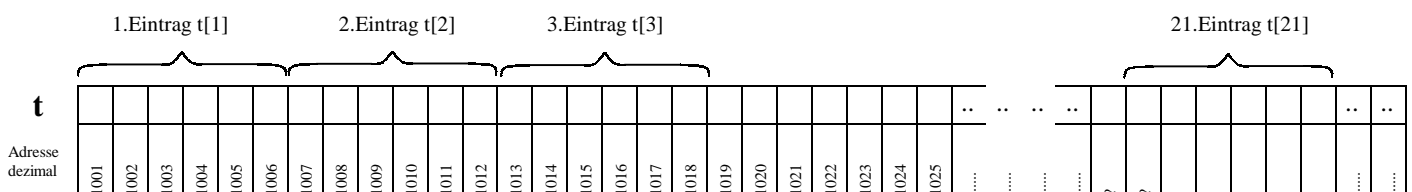
2.4.5 Codierung komplexer Strukturen

Will man nicht nur Zahlen und Zeichenketten verarbeiten, sondern auch komplexere Strukturen, so müssen auch diese im Speicher des Rechners repräsentiert werden. Zwei grundlegende Strukturen werden hier kurz erklärt. Man spricht auch von **zusammengesetzten Datentypen**.

2.4.5.1 Array (Reihung).

Beispiel:

Die Meßwerte der durchschnittlichen Tagestemperatur für jeden Tag eines Jahres sollen gespeichert werden. Man muß sich entscheiden: Von welchem Datentyp soll ein einzelner Meßwert sein? Da ein Meßwert sinnvollerweise eine Dezimalzahl sein sollte, wird man für Meßwerte einen Gleitkommatyp wählen, in Pascal den Typ Real. Die gesamte Tabelle wird dann aus 365 **gleichartigen Einträgen** bestehen, die alle den gleichen Speicherplatz, z.B. 6 Bytes brauchen. Ein solche Tabelle nennt man eine Reihung (füher Feld), engl. **Array**. Bezeichnet man das gesamte Array mit t, so soll t[i] den i-ten Eintrag bezeichnen; wobei i die Werte 1 bis 365 annehmen kann und als Index bezeichnet wird. In der Mathematik schreibt man statt dessen t<sub>i</sub>. Da alle Komponenten des Array die gleiche Bytezahl verbrauchen, ist es einfach, aus der Anfangsadresse des Array und dem Index i die Adresse der i-ten Komponente zu errechnen.



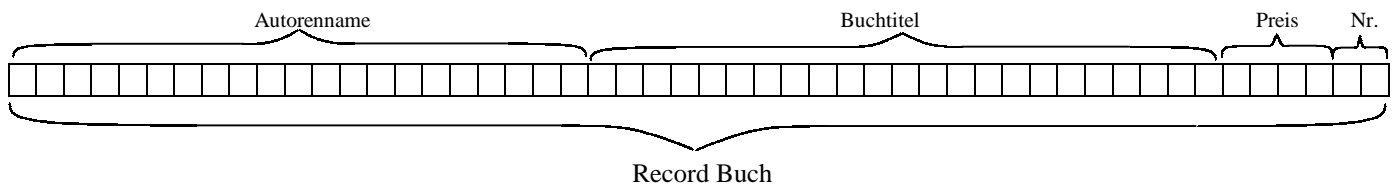
### 2.4.5.2 Record (Verbund, Datensatz)

Beispiel:

Die Daten für ein Bücher einer Bibliothek sollen erfasst werden, der Einfachheit halber nur etwa der Autorename, der Buchtitel, der Kaufpreis und die Inventarnummer. Hier sind die Einträge von **verschiedenem Typ**, der Autor und der Titel sind Zeichenketten, der Kaufpreis eine Gleitkommazahl und die Inventarnummer eine positive Ganzzahl oder auch eine Zeichenkette. Eine solche Struktur heißt Verbund oder Datensatz, engl. **Record**, mit den **Feldern** Autorename, Buchtitel, Kaufpreis und Inventarnummer. Beabsichtigt man, später die Daten vieler Bücher als Array zu speichern (→Datenbank), so ist klar, dass man versuchen wird, die Einträge für jedes Buch gleich lang zu machen. Man könnte etwa festlegen:

Feld	Datentyp
Autorename:	Zeichenkette mit maximal 20 Zeichen (21 Bytes)
Buchtitel	Zeichenkette mit maximal 20 Zeichen (21 Bytes)
Kaufpreis	Gleitkommazahl (Real) (6 Bytes)
Inventarnummer	Ganzzahl (Word) (2 Bytes)

} Gesamter Datensatz  
50 Bytes



Selbstverständlich können solche Records wieder zum Aufbau von Arrays verwandt werden.

### 2.4.6 Zusammenfassung

Für alle zu verarbeitenden Daten muss der verwandte Datentyp festgelegt sein.

Der Datentyp bestimmt

- den benötigten Speicherplatz
- die Interpretation der Datenbits
- den Bereich der möglichen Werte für die Daten
- die Operationen, die man mit den Daten durchführen kann

Man unterscheidet zwischen **einfachen Datentypen** (Zahltypen, Boolean, Character, String) und den **zusammengesetzten (strukturierten) Datentypen** (wie Array, Record und andere).

Jede Programmiersprache stellt eine Reihe von Datentypen zusammen mit geeigneten Operationen darauf zur Verfügung. Um die oben skizzierten technischen Realisierungen muss sich der Programmierer im allgemeinen nicht kümmern. Lediglich die Bereiche der möglichen Werte muss er den Sprachbeschreibungen entnehmen.



## 2.5 Codieren und Wiedergewinnen von Information: Beispiele

Hier ist die Ausgabe einer ASCII Textdatei **test.txt** auf dem Bildschirm. Die Datei stand auf einer Festplatte. Sie enthält keinerlei Formatierungen und kann unter DOS mit dem Befehl *type test.txt* auf dem Bildschirm angezeigt werden oder mit Notepad unter Windows in einem Bildschirmfenster betrachtet werden. Diese Systeme interpretieren den binären Code als ASCII Code und stellen die Datei entsprechend dar.

Bildschirm:

```
C:\>type test.txt
Hallo, das ist ein Text!
Und das die 2.Zeile.
```

Jetzt wird die Datei mit einem Textbetrachter<sup>1</sup> als Datei von Bytes in Hexadezimal-Darstellung angesehen. Rechts daneben werden die Deutungen als ASCII Zeichen zusätzlich gezeigt.

Text-Betrachter:	C:\...\WS02-03\test.txt	Sp 0	111 Byte	100%
00000	48 61 6C 6C 6F 2C 20 64	61 73 20 69 73 74 20 65	Hallo, das ist e	
00010	69 6E 20 54 65 78 74 21	0D 0A 55 6E 64 20 64 61	in Text! Und da	
00020	73 20 64 69 65 20 32 2E	5A 65 69 6C 65 2E 0D 0A	s die 2.Zeile.	
00030	20 20 20 20 20 20 20 20	20 20 20 20 20 20 20 20		
00040	20 20 20 20 20 20 20 20	20 20 20 20 20 20 20 20		
00050	20 20 20 20 20 20 20 20	20 20 20 20 20 20 20 20		

Die Hexadezimalzahlen in der ersten Spalte geben an, welche Nummer das 1.Byte in dieser Zeile hat, gerechnet vom Beginn der Datei an (relative Adresse).

- Prüfen Sie mit Hilfe einer ASCII Tabelle, ob die rechts gezeigte Interpretation des Hexadezimalcodes als ASCII Zeichen richtig ist.
- Was bedeutet wohl das Zeichen mit dem Code 20<sub>H</sub>?
- Was könnten die Zeichen mit dem Code 0D<sub>H</sub> und 0A<sub>H</sub> bedeuten?

Derselbe Text mit Formatierungen im Programm WORD FÜR WINDOWS:

Der Ausdruck von test1.doc auf einem Drucker:

**Hallo, das ist ein Text!**  
Und das die 2.Zeile.

Ein Ausschnitt der Datei als Datei von Bytes in Hexadezimal-Darstellung:

Text-Betrachter:	C:\...\WS02-03\test1.doc	Sp 0	6.656 Byte	42%
00B60	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....	
00B80	48 61 6C 6C 6F 2C 20 64	61 73 20 69 73 74 20 65	Hallo, das ist e	
00B90	69 6E 20 54 65 78 74 21	0D 55 6E 64 20 64 61 73	in Text! Und das	
00BA0	20 64 69 65 20 32 2E 5A	65 69 6C 65 2E 0D 07 00	die 2.Zeile. ..	
00BB0	A1 01 00 96 01 97 00 D0	CF 11 E0 A1 B1 1A E1 00	í'û'ù.--•_í_•ß.	
00BC0	FE FF 00 00 03 0A 00 00	00 00 00 00 00 00 00 00	___~.....	
00BD0	00 00 00 00 00 00 00 00	01 00 00 00 E0 85 9F F2	.....'...._af_	
00BE0	F9 4F 68 10 AB 91 08 00	2B 27 B3 D9 30 00 00 00	•Oh•½æ..+' +0...	
00BF0	D0 02 00 00 0E 00 00 00	07 00 00 00 98 00 00 00	-^.....ÿ...	
00C00	05 00 53 00 75 00 6D 00	6D 00 61 00 72 00 79 00	~.S.u.m.m.a.r.y.	
00C10	49 00 6E 00 66 00 6F 00	72 00 6D 00 61 00 74 00	I.n.f.o.r.m.a.t.	
00C20	69 00 6F 00 6E 00 00 00	00 00 00 00 00 00 00 00	i.o.n.....	
1Hilfe	2K.Umbr 3	4ASCII 5	6	7Suchen 8Betra. 9
			10Quit	

Die hier auftretenden unverständlichen Codes zeigen, dass die Formatierungsangaben in Word nicht ohne weiteres zu verstehen sind.

<sup>1</sup> hier aus dem Norton Commander; ein entsprechendes Shareware Programm ist der „Servant Salamander“, [www.altap.cz](http://www.altap.cz)

## 2.6 Datentypen in höheren Programmiersprachen am Beispiel von TURBO-PASCAL

In sogenannten problemorientierten Programmiersprachen (anwendungsorientierten Programmiersprachen), mit deren Hilfe fast alle größeren Programme erstellt werden, muß sich der Programmierer nicht selbst um die detaillierte Organisation des Speichers für die zu verarbeiteten Daten kümmern. Es genügt in den meisten Fällen zu wissen, welche Art von Daten verarbeitet werden sollen, aus welchen Bereichen diese Daten sein dürfen und welche Operationen für diese Daten definiert sind. Solche Operationen sind für Zahlentypen z.B. die üblichen Rechen- und Vergleichsoperationen (+, -, /, \*, <, >, =) und für Zeichenketten (Strings) die Verkettung und Teilstringoperationen.

Ein Beispiel in PASCAL soll zeigen, wie ein Pascal-Programm aufgebaut ist und wie der Übersetzer (→Compiler, nächstes Kapitel) die Speicherverwaltung für die Daten übernimmt. Ansonsten tut das Programm nichts sinnvolles.

```
Program tunix;
uses crt;
```

```
Var
n,m: integer;
x : real;
st : string;
ww : boolean;
```

```
Const
```

```
max=10;
```

```
begin
```

```
clrscr;
```

```
n:=5;
```

```
n:=n+2;
```

```
m:= 2*n+1;
```

```
x:=3.2;
```

```
x:=n;
```

```
st:='Hallo';
```

```
write('st eingeben :');
```

```
readln(st);
```

```
write('x eingeben :');
```

```
readln(x);
```

```
writeln(st+'das wars' );
```

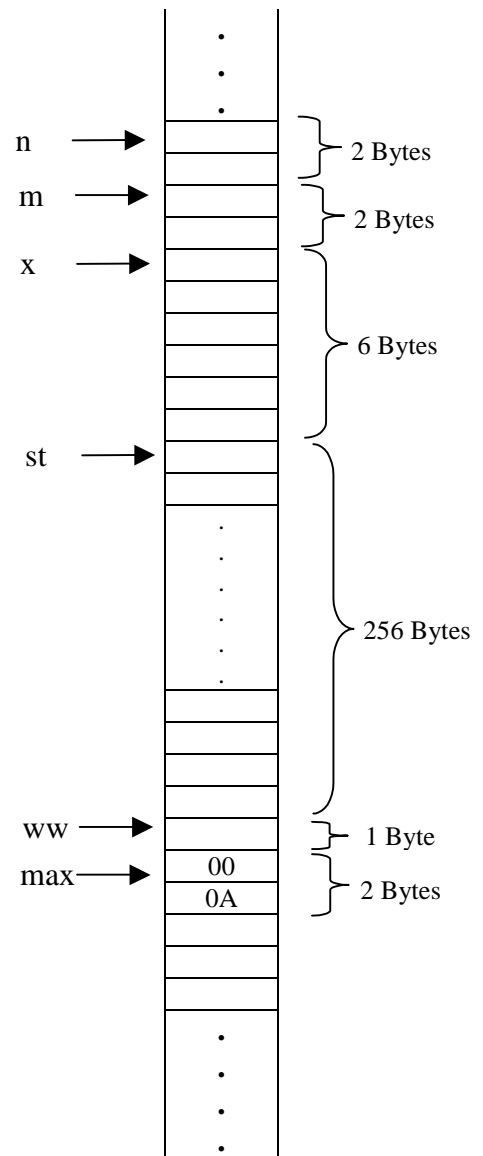
```
writeln('Das ist x :' , x:6:2);
```

```
ww:=true;
```

```
ww:= m=n;
```

```
end.
```

**Speicher** (schematisch)  
zu Beginn des Hauptprogramms



Variablendeklaration

reserviert Speicherplatz

Konstantendeklaration

reserviert Speicherplatz **und**  
setzt Werte fest

Beginn des  
Hauptprogramms

Wertzuweisung

Weitere  
Wertzuweisungen

Typenumwandlung  
automatisch

Ausgabe

Eingabe

formatierte  
Ausgabe

Ende des  
Hauptprogramms

Unzulässig: `st:=x;` zulässig nur explizite Typenumwandlung mit Umwandelungsprozedur `str(x:6:2,st)`  
`n:=x;` zulässig nur explizite Typenumwandlung mit Umwandelungsfunktion `trunc`: `n:=trunc(x)`  
`ww:=n;`

Zulässig, aber sinnlos: `n:=n+2` als *erste* Anweisung im Programm, da dann n noch keinen Wert hat.

In manchen Programmiersprachen (z.B. BASIC) sind alle Variablen mit dem Wert 0 initialisiert. In PASCAL oder C ist dies nicht der Fall, der Inhalt der entsprechenden Speicherplätze ist undefiniert, d.h. man kann nicht sagen, welche Werte dort stehen.

### 3 Vom Problem zum Programm: Übersicht (vereinfacht)

#### (1) Problem und Aufgaben eingrenzen

Was soll mit welchen Daten getan werden?

Eingangs- und Ausgangsdaten ( E, A )

Funktionale Beschreibung ( V )

EVA – Prinzip ( HIPO Methode Hierarchische Input Process Output Methode )

#### (2) Spezifikation ( Abstraktion von (1) )

Datenstruktur festlegen

Eingangs- und Ausgangsdaten spezifizieren

Vorbedingungen für Eingangsdaten

Nachbedingungen für Ausgangsdaten

Funktionalen Zusammenhang angeben

Auf welchem Rechner mit welcher Programmiersprache implementieren?

Benutzerschnittstelle beschreiben

#### (3) Entwurf der Algorithmen und der Programmstruktur

Entwurfsmethoden: Top-Down oder Bottom-Up

Strukturierte Programmierung

Objekt-orientierte Programmierung ( OOP ), für große Programmpakete mit Computerunterstützung (z.B. UML)

Benutzer Sprachtyp

Maschinennah (Assembler)

Prozedural, Imperativ (Pascal, C, Basic, Fortran,....)

Funktional (Lisp, Scheme, Logo)

Deklarativ (Prolog)

Objekt-orientiert (Smalltalk, Eiffel )

Imperativ- Objekt-orientiert (Turbo-Pascal ab Version 5.5, Delphi, C++, Java, Visual Basic,...)

#### (4) Codieren der Algorithmen in einer Programmiersprache ( = 'Programmieren' )

Programmiersprachen:

Formale Sprache mit eindeutiger Syntax (Grammatik → Syntax-Diagramme, EBNF-Grammatik) und

Semantik (Bedeutung)

#### (5) Compilieren oder Interpretieren des Programmcodes

Compiler: Übersetzen des gesamten Quellprogramms in Maschinencode für eine spezielle Maschine, das ohne Benutzung des Compilers dann ausgeführt werden kann.

Interpreter: Zeilenweise Übersetzung *und sofortige Ausführung* des Quellprogramms. Der Interpreter wird dazu benötigt.

#### (6) Verifizieren oder Testen

Verifizieren:

Beweisen, dass das Programm der Spezifikation genügt.

Schwierig bis unmöglich.

Testen:

An geeignet ausgewählten Testdaten prüfen, ob das Programm dafür der Spezifikation genügt.

Zeigt nur vorhandene Fehler, aber nie die Fehlerfreiheit.

Es gibt heute wohl keine großen fehlerfreien Programme (neuere Untersuchung zeigt: 1,5 Fehler pro 1000 Zeilen Quellcode für *freigegebene professionelle* Softwareprodukte , Inf.Spektrum 21, 1998 S.250) .

There is always one more bug!

#### (7) Dokumentieren

Beschreibung für den Benutzer

Beschreibung für die spätere Wartung und Weiterentwicklung

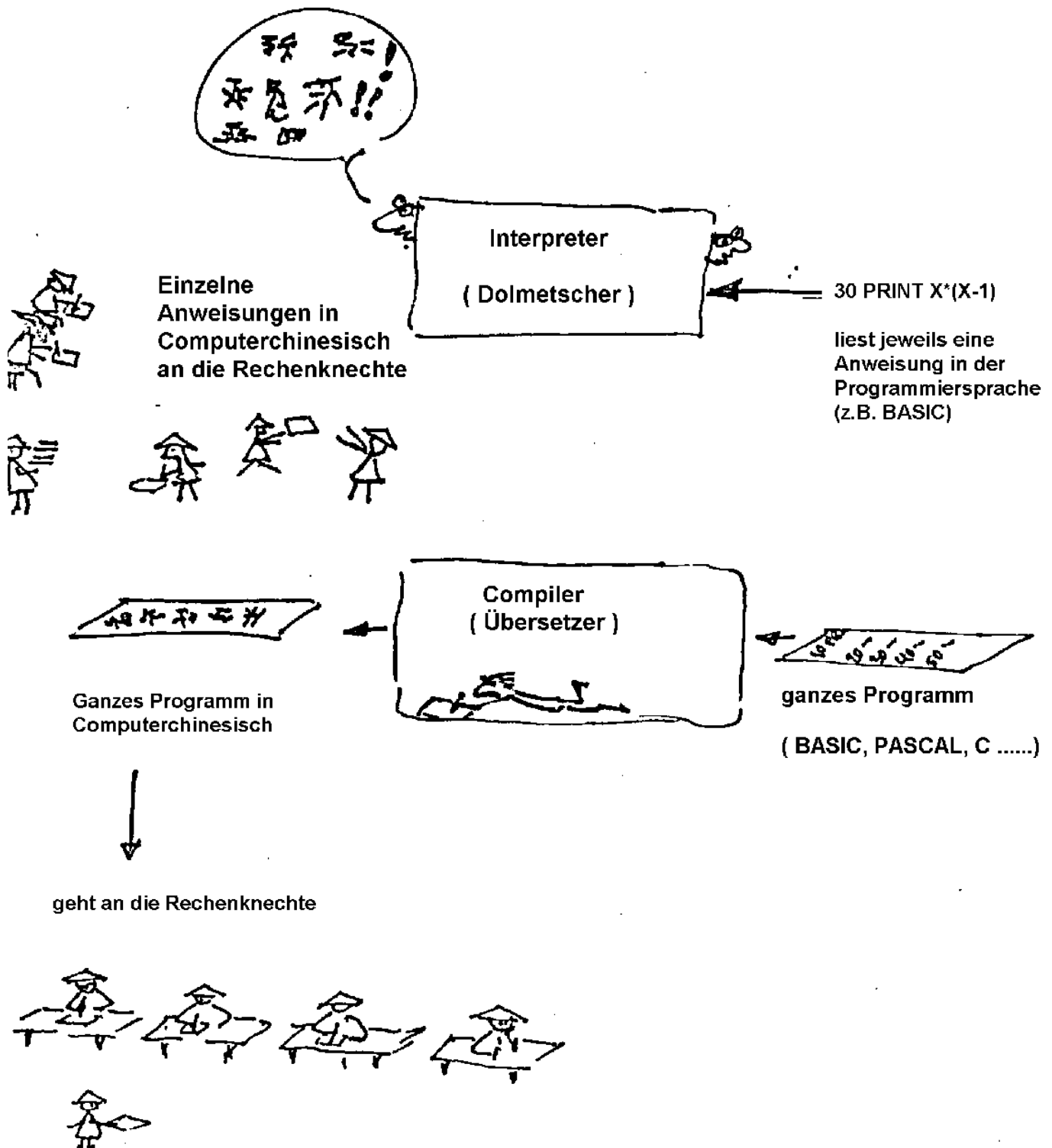
→ Strukturierte Programmierung soll selbst-dokumentierend sein; die Sprache UML kann als Dokumentation dienen.

#### (8) Installation und Wartung

Anpassung an Kundenwünsche, nachträgliche Fehlerbeseitigung und Verbesserung.

Zyklus wird heute meist mehrfach durchlaufen, Test und Entwicklung sind verzahnt.

### 3.1 Interpreter - Compiler



**Compiler** übersetzen in einem Arbeitsgang das gesamte Quellprogramm in ausführbaren Maschinencode und werden danach nicht mehr gebraucht. Da nicht jedesmal der Quellcode neu gelesen und übersetzt werden muß, ist die Programmausführung schneller als beim Interpreter.

**Interpreter** lesen ein Programm zeilenweise und übersetzen diese sofort in Maschinencode. Sie müssen während der gesamten Ausführung zur Verfügung stehen. Die Programmausführung ist langsamer als bei Compilern. Es können allerdings syntaktisch fehlerhafte Programme solange ausgeführt werden, bis der Fehler auftritt. Dies ist bei Compilern meist nicht der Fall.

Heute sind die meisten Sprachen Compilersprachen. Interpretersprachen sind z.B. Java Script, LOGO, LISP und die Autorensprache Toolbook. Die ‚Internet-Sprache‘ JAVA steht dazwischen: Übersetzung durch Compiler in einen hardwareunabhängigen Zwischencode, der von einem im Browser enthaltenen Interpreter interpretiert und ausgeführt wird.

## 3.2 Syntax von Programmiersprachen

### Syntax und Semantik für die natürlichen Sprachen.

Für die natürlichen Sprachen wie Deutsch oder Französisch sind Syntax und Semantik nicht einfach. Zum Leidwesen der Informatiker und Linguisten, die an maschineller Übersetzung arbeiten, kann für natürliche Sprachen die Syntax nicht von der Semantik getrennt werden. Nicht jedem formal korrekten Satz kann eine eindeutige Bedeutung zugewiesen werden. Man betrachte dazu z.B.:

'Der Wind belügt den Geschäftsführer der Angst',  
 'Mimi war in München verliebt',  
 'Er nahm den Kuchen vom Teller und aß ihn',  
 'Der Diebstahl einer Putzfrau ist nicht über die Haftpflichtversicherung gedeckt',  
 'Wenn der Tierarzt dem Hund einen Zahn zieht, muß die Arzthelferin die Schnauze halten'.  
 Natürliche Sprachen sollen hier nicht weiter untersucht werden.

### Syntax für formale Sprachen.

Formale Sprachen sind Sprachen, deren Syntax durch einen formalen Kalkül gegeben wird.

Alle Programmiersprachen sind formale Sprachen mit meist recht einfacher Syntax. Zwei in der Praxis gebräuchlichen Darstellungen werden hier an Beispielen eingeführt.

In der Theorie der formalen Sprachen bezeichnet man als ‚Sprache‘ jede Menge von Zeichenketten über einer gegebenen Symbolmenge, eine Grammatik dafür ein Regelsystem, das genau diese Zeichenketten formal zu erzeugen gestattet.

### 3.2.1 Erweiterte Backus-Naur-Form (EBNF)

Die erweiterte Backus-Naur-Form einer Grammatik ist eine Darstellung der Grammatikregeln. Viele Programmiersprachen werden in der EBNF definiert. Um eine Grammatik für eine Sprache aufzuschreiben, benötigt man Zeichen, die nicht selbst der Sprache angehören. Diese werden Metazeichen genannt, im Gegensatz zu den Zeichen der Sprache, die als Objektzeichen bezeichnet werden.

#### Metazeichen der EBNF sind:

- ◆ Das Definitionszeichen =,
- ◆ das Alternativzeichen |,
- ◆ die Anführungszeichen “ und ” für Objektsymbole,
- ◆ die Wiederholungsklammern { und } ,
- ◆ die Optionsklammern [ und ] ,
- ◆ die Gruppenklammern ( und ) ,
- ◆ der Punkt . am Ende einer Regel.

Häufig werden Objekt- und Metasprache nicht durch die Anführungszeichen sondern durch eine andere Schriftart unterschieden, z.B. durch Fett- oder Kursivschrift. Dies verbessert die Lesbarkeit.

Beispiele für Grammatiken (die zugrunde liegenden Symbolmengen werden nicht explizit genannt):

1.Beispiel (mit Anführungszeichen):

$$s = k'0'\{k'0'\}.$$

$$k = '1'\{'1'\}.$$

1.Beispiel (übersichtlicher ohne Anführungszeichen):

$$s = k\mathbf{0}\{k\mathbf{0}\}.$$

$$k = \mathbf{1}\{\mathbf{1}\}.$$

2.Beispiel:

*beschreibg* =“Das Programm arbeitet nach dem ” *prinzip* “der” *adj* 1.Hälfte 2.Hälfte.  
*prinzip* =“Prinzip” | “Verfahren” | “Algorithmus” | “System”.  
*adj* =“iterierten” | “rezidierten” | “substantivierten”.  
 1.Hälfte =“Rekursions” | “Iterations” | “Varianz” | “Diversifikations”.  
 2.Hälfte =“analyse” | “elimination” | “substitution” | “integration”.  
 .

3.Beispiel (ohne Anführungszeichen)

$$s = [\mathbf{0}]k\{\mathbf{0}k\}$$

$$k = \mathbf{1}\{\mathbf{1}\}.$$

oder noch kürzer, aber vielleicht nicht mehr gut lesbar:  

$$s = [\mathbf{0}]\mathbf{1}\{\mathbf{1}\}\{\mathbf{0}\mathbf{1}\{\mathbf{1}\}\}$$

Geben Sie für Beispiel 1 die verkürzte Form an

4.Beispiel

$s = palindr.$

$palindr = \square | 0 | 1 | 0 palindr 0 | 1 palindr 1.$

5.Beispiel

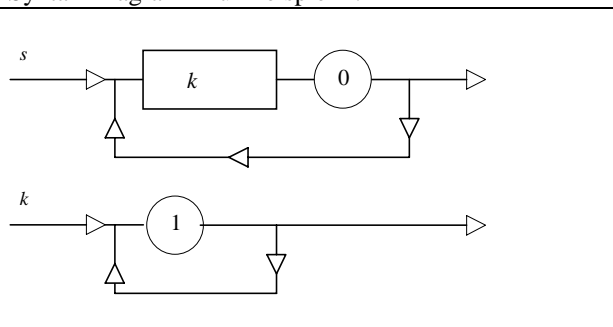
$s = gleichviel.$

$gleichviel = \square | gleichviel gleichviel | 0 gleichviel 1 | 1 gleichviel 0.$

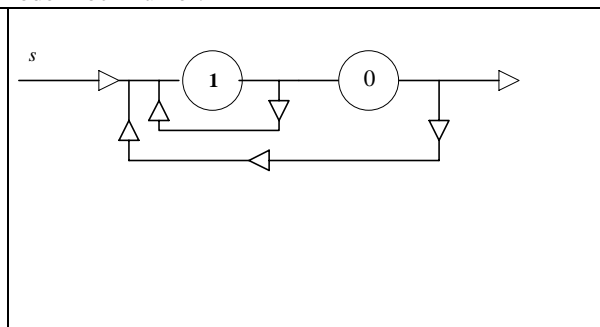
**3.2.2 Syntax-Diagramme**

Noch einfacher zu lesen als die EBNF sind Syntax-Diagramme. Ihr Nachteil liegt nur im großen Platzbedarf für die Darstellung. Die EBNF einer Grammatik läßt sich ganz einfach in Syntax-Diagramme umsetzen. In dieser Form werden Programmiersprachen in neuerer Zeit häufig beschrieben. Ein Beispiel ist etwa Turbo-Pascal ab Version 4, während Turbo-Pascal 3 noch in EBNF vorlag. Statt durch eine formale Definition sollen Syntax-Diagramme an Hand der oben gegebenen Beispiele eingeführt werden.<sup>2</sup>

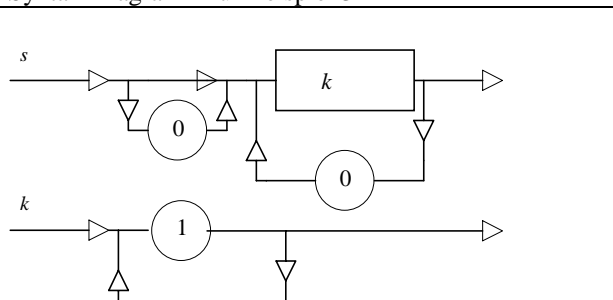
Syntax-Diagramm für Beispiel 1.



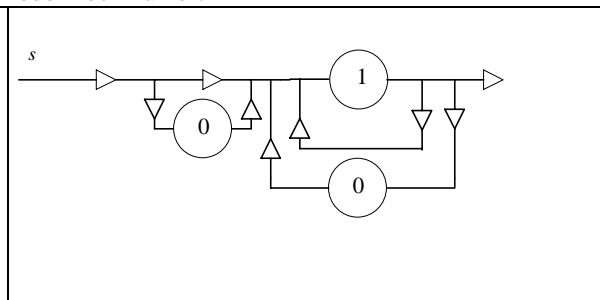
oder noch kürzer:



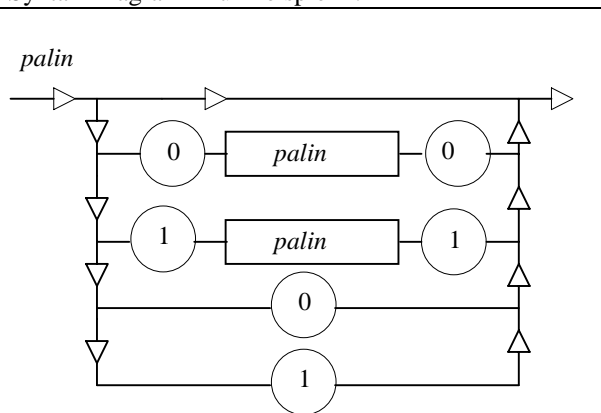
Syntax-Diagramm für Beispiel 3



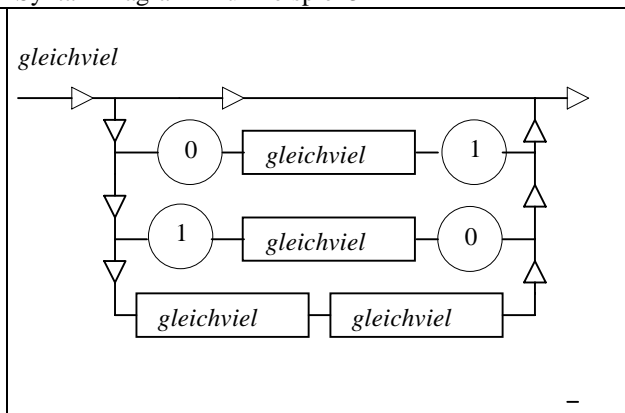
oder noch kürzer:



Syntax-Diagramm für Beispiel 4.



Syntax-Diagramm für Beispiel 5



<sup>2</sup>Für eine formale Definition und eine Übersetzungsregel von der EBNF in Syntax-Diagramme siehe Klaeren, Herbert, Informatik Teubner 1991. Einen Vergleich von EBNF und zugehörigem Syntax-Diagramm für ein sogenanntes 'Mini-Pascal' findet man dort ebenfalls.

**Bemerkungen: Scanning und Parsing.**

Die Syntax für Programmiersprachen muß eine **eindeutige Zerlegung** der zugelassenen Ausdrücke in ihre Teile gestatten. Diese Zerlegung dient dann dazu, den Ausdrücken eine **Bedeutung** zuzuweisen (**Semantik**). Man geht dabei von den einfachsten sprachlichen Teilen aus, ordnet ihnen eine Bedeutung zu und definiert dann, wie sich daraus die Bedeutung für die zusammengesetzten Ausdrücke ergibt. Entsprechend arbeiten auch **Compiler** und **Interpreter**. Die Zerlegung eines Programmtextes in seine Grundbestandteile (Tokens) heißt **Scanning**, der Prozeß, der analysiert, ob dieser Text syntaktisch zulässig ist und der die genannte Zerlegung findet, heißt **Parsing**.

**3.2.3 Syntax für Mini-Pascal<sup>3</sup>**

Die folgende EBNF und die Syntaxdiagramme beschreiben eine stark vereinfachte Version von Pascal, die hier Mini-Pascal genannt wird.

Die trivialen Regeln bzw. Diagramme für die Bezeichner (Identifier wie "constIdent", "varIdent" und "procIdent"....) lassen wir weg.

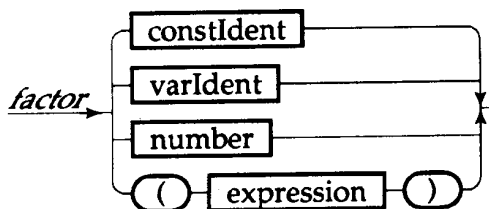
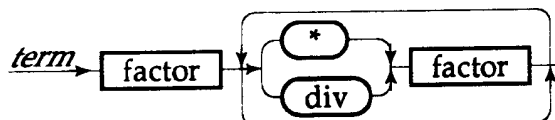
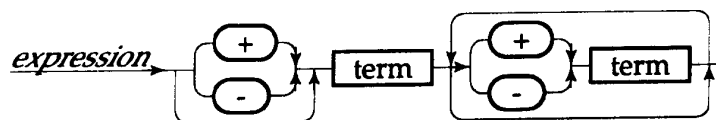
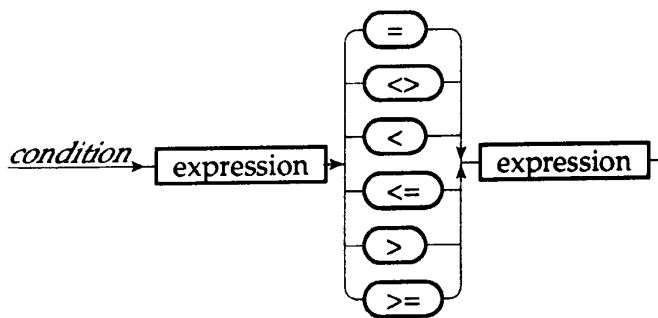
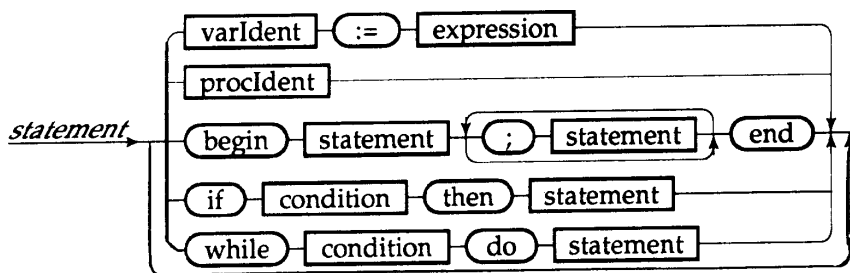
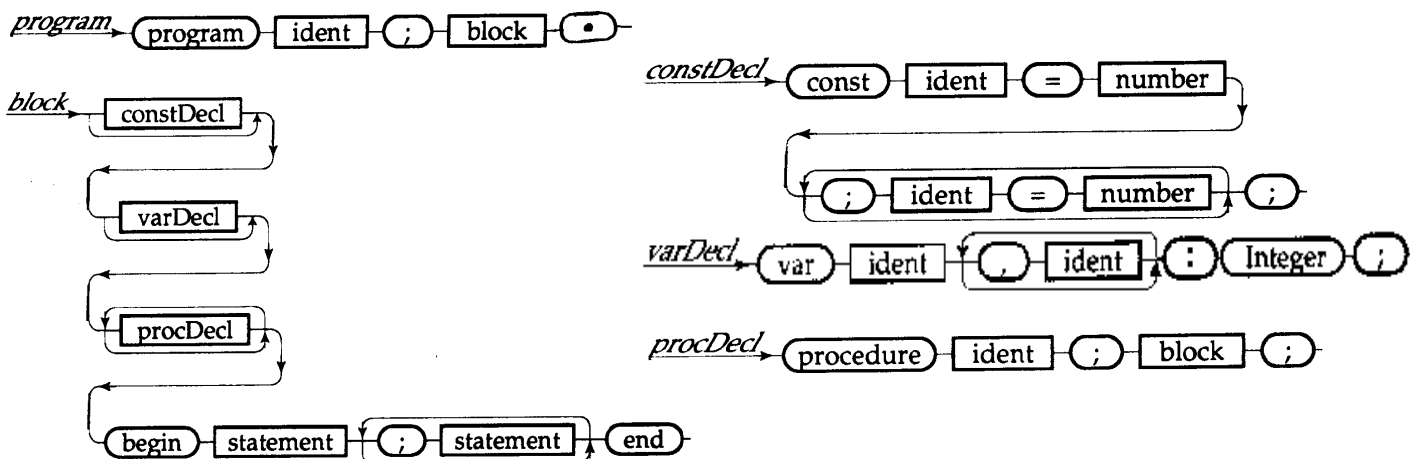
**Erweiterte Backus-Naur-Form (EBNF)**

```

program      =  "program" ident ";" block ";".
block        =  [constDecl] [varDecl] {procDecl}
               "begin" statement { ";" statement } "end".
constDecl    =  "const" ident "=" number
               { ";" ident "=" number } ";".
varDecl      =  "var" ident { "," ident } ":" Integer;".
procDecl     =  "procedure" ident ";" block ";".
statement    =  [varIdent ":=" expression |
               procIdent |
               "begin" statement { ";" statement } "end" |
               "if" condition "then" statement |
               "while" condition "do" statement].
condition    =  expression ("=" | "<" | "<=" | ">" | ">=")
               expression.
expression   =  ["+" | "-"] term { ("+" | "-") term }.
term         =  factor { ("*" | "div") factor }.
factor       =  constIdent | varIdent | number | "(" expression ")".
constIdent  =  ident.
procIdent   =  ident.
varIdent    =  ident.

```

Syntax-Diagramm





### 3.2.4 Aufgaben zur Syntax von Turbo-Pascal

In den folgenden Aufgaben soll die oben angegebene Syntax für Mini-Pascal angewandt werden.

Mini-Pascal ist gegenüber den Vollversionen von Pascal eine stark eingeschränkte Programmiersprache und soll nicht zum Programmieren, sondern zum Studium von Prinzipien dienen, die Programmiersprachen zugrunde liegen.

#### Mini-Pascal

- einziger Datentyp ist INTEGER
- "Identifizier" sollen z.B. sein zaehler, i, j, max, sortieren,.....  
"number" sollen z.B. sein 23, 15, 1007... (positiv, ganz)
- varIdent, constIdent usw. sind einfach "Identifizier".

1) Versuchen Sie, in Worten zu beschreiben, was man unter "faktor", "term", "expression" versteht.

2) Prüfen Sie, ob folgende Zeichenfolgen "expressions" sind:

200	-200	-max + 30
- -20	- (+20) - max	30*min*max
70*-max	70*(-max)+10*z	3*(20+i) div (10-z) div 3
3 div (((5-i)))	((3+z)	- 3*(-x)

3) Geben Sie ein möglichst einfaches Programm an.

4) Prüfen Sie, ob folgende Zeichenfolgen "statements" sind:

max:=30	sortieren
max:=30;	begin max:=30 end
begin i:=3 ; end	begin ; ; end
begin ; ; end;	if i=3
if i=3 then	if i=3 then i:=i+1
if i=3 then i:=i+1 ; j:=j+1	if i=3 then begin i:=i+1; j:=j+1 end
x+1:=3	3:=max

5) Prüfen Sie, ob folgende Zeichenfolgen "varDecl" sind:

var i : INTEGER;	var i : INTEGER
var i, j, max : INTEGER;	var i; j; max :INTEGER;

6) Schreiben Sie einige Beispiele für "constDecl" auf.

## 4 Algorithmen

### 4.1 Definition

Ein Algorithmus ist eine **endliche, eindeutige** Beschreibung eines **effektiven** Verfahrens zur Lösung einer **Klasse von Problemen**.

- ◆ **Endlichkeit:** Das Verfahren muß schrittweise in einem endlichen Text beschrieben werden. Formulierungen wie „usw.“, „1+2+3+...“ dürfen nicht vorkommen.
- ◆ **Eindeutigkeit:** An jeder Stelle des Verfahrens ist eindeutig bestimmt, wie das Verfahren weiter abläuft. Der Zufall wird ausgeschlossen, ebenso eine beliebige Wahl des weiteren Vorgehens. Statt von Eindeutigkeit spricht man auch von **Determiniertheit**. Bei jeder Wiederholung des Verfahrens mit den gleichen Eingangsdaten muß der Prozeß gleich ablaufen.
- ◆ **Effektivität:** Das Verfahren kann im Prinzip mechanisch (von einer Maschine ohne Intelligenz) ausgeführt werden. Alle Vorschriften müssen bis ins kleinste Detail festgelegt sein und nur einfachste Manipulationen von Zeichen erfordern. Anweisungen wie "Setze  $x:=3$  wenn in der Dezimaldarstellung von  $\pi$  unendlich viele Nullen vorkommen, sonst setze  $x:=1$ " dürfen nicht auftreten.

Bemerkungen:

Viele Autoren verlangen noch zusätzlich von einem Algorithmus die **Terminierung** von Algorithmen. Das soll bedeuten, dass das Verfahren bei allen Eingaben nach endlich vielen Schritten endet (terminiert), d.h. z.B. nicht in einer Endlosschleife „stecken bleibt“.

Es ist ohne Angabe einer konkreten Maschine nicht klar, was man unter „bis ins kleinste Detail“ verstehen soll. Deshalb läßt sich der Begriff Algorithmus am besten verstehen, wenn man zunächst eine konkrete Programmiersprache zugrunde legt. Daher sollen im folgende die Prinzipien zum Entwurf und zur Darstellung von Algorithmen am Beispiel von Pascal entwickelt werden.

### Zur Herkunft des Worten „Algorithmus“:

#### Abu Ja'far Muhammad ibn Musa al'Khwarizmi

Mohammed, Vater des Ja'far, Sohn des Moses, aus Khowarizmi

(persische Provinz südlich des Aralsees, heute Usbekistan)

Geboren: ca. 790, gestorben: ca. 850

Beruf: Bibliothekar des Kalifen in Bagdad

Wichtigste Bücher:

- Hisab **al-jabr** w'al muqabala → Algebra  
 Quadratische Gleichungen und  
 quadratische Ergänzung, geometrisch formuliert.  
 Ohne Symbole! Keine negativen Koeffizienten
- Ein Buch (ohne arabischen Titel) über das Rechnen im indischen Zahlensystem, Übersetzung im 12. Jh. ins Lateinische mit dem Titel:  
 Algoritmi de numero Indorum → Algorithmus

Die lateinische Übersetzung des Robert von Chester im 12. Jahrhundert beginnt mit den Worten „Dixit Algorithmi: laudes deo rectori nostro atque defensori dicamus dignas ...“

Die Schriften der Araber gelangten über das von den Arabern besetzte Spanien in die abendländische Kultur.



### 4.2 Sortierverfahren als Beispiele für Algorithmen

Sortiert wird ein Feld von 7 Zeichen (char). Die konkrete Umsetzung des Algorithmus erfolgt in Turbo-Pascal. Die Programme zeigen beim Ablauf auf dem Bildschirm nichts. Sie sind nur dazu gedacht, Sortieralgorithmen prinzipiell zu verstehen und die Vorgänge im Rechner mit Hilfe des Debuggers im sogenannten Watch-Fenster zu verfolgen.

#### 4.2.1 Das Bubblesort Verfahren in Pascal.

Diese Version des Verfahrens ist nicht optimal, da das Feld bei jedem "Durchgang" immer bis zum Ende durchlaufen wird. Wie könnte man das Verfahren ändern, so dass das Feld jedesmal nur soweit durchlaufen wird, wie unbedingt nötig?

```
program bubblesort;
uses crt;

var
  a:array[1..7] of char;
  i:integer;
  hilfshand:char;
  sortiert:boolean;

begin
  a[1]:='X';
  a[2]:='R';
  a[3]:='B';
  a[4]:='H';
  a[5]:='W';
  a[6]:='D';
  a[7]:='M';

  repeat
    sortiert:=true;
    for i:=1 to 6 do
      begin
        if a[i+1]<a[i] then
          begin {Vertauschen}
            sortiert:=false;
            hilfshand:=a[i];
            a[i]:=a[i+1];
            a[i+1]:=hilfshand;
          end
        end;
      until sortiert;
    end.
end.
```

#### 4.2.2 Das Verfahren „Sortieren durch Auswahl“ in Pascal.

```
program sortieren1;
var
  a:array[1..7] of char;
  i,k,l:integer;
  hilfshand:char;

begin {Hauptprogramm}
  a[1]:='X';
  a[2]:='R';
  a[3]:='B';
  a[4]:='H';
  a[5]:='W';
  a[6]:='D';
  a[7]:='M';

  for i:= 1 to 6 do
    begin
      k:=i; {Index k des kleinsten a[k] zwischen Stelle i und 7 finden }
      for l:=i+1 TO 7 do
        if a[l]<a[k] then k:=l;

        hilfshand:=a[i]; {Vertauschen}
        a[i]:=a[k];
        a[k]:=hilfshand;
      end
    end.
end.
```

## Das Verfahren „Sortieren durch Auswahl“ nochmals mit Prozeduren

```

program sortieren2;
var
  a:array[1..7] of char;
  i,k,l:integer;
  hilfshand:char;

  {-- Hilfsprozeduren -----}

procedure feldErzeugen;
begin
  a[1]:='X';
  a[2]:='R';
  a[3]:='B';
  a[4]:='H';
  a[5]:='W';
  a[6]:='D';
  a[7]:='M';
end;

procedure IndexVomKleinstenElementSuchen;
begin
  k:=i; {Index k des kleinsten a[k] zwischen Stelle i und 7 finden }
  for l:=i+1 to 7 do
    if a[l]<a[k] then k:=l;
end;
procedure WerteVertauschen;
begin
  hilfshand:=a[i];
  a[i]:=a[k];
  a[k]:=hilfshand;
end;

  {-- Hilfsprozeduren Ende -----}

begin {Hauptprogramm}
  feldErzeugen;
  for i:= 1 to 6 do
    begin
      IndexVomKleinstenElementSuchen;
      WerteVertauschen;
    end;
end.

```

### 4.2.3 Sortieren in einer funktionalen Programmiersprache:

(Deutsche Version von MSW-LOGO)

Es werden zwei Funktionen definiert:

```

pr einfügen :zahl :liste
  wenn :liste = [ ] [Rückgabe miterstem :zahl [ ] ]
  wennsonst :zahl < erstes :liste [ Rückgabe miterstem :zahl :liste] [ Rückgabe miterstem erstes :liste einfügen :zahl ohneerstes
:liste]
ende

```

```
pr sort :liste
  wenn :liste=[] [Rückgabe [ ] ]
  wennsonst (ohneerstes :liste) =[ ] [Rückgabe :liste] [Rückgabe einfügen erstes :liste sort ohneerstes :liste]
ende
```

Die Funktion **einfügen** fügt eine Zahl an der "richtigen" Stelle in eine Liste ein.

Beispiel: einfügen **5** [2 3 7 8 12 4 2] = [2 3 **5** 7 8 12 4 2]

Die Funktion **sort** benutzt diese Einfügefunktion, um eine Liste zu sortieren.

Aufruf:

```
druckezeile sort [7 5 4 8 9 12 3 55 32 12 2 3 2 5 1 88 6 5 4 3 7 9 0 2 4 ]
```

Ausgabe:

```
0 1 2 2 2 3 3 3 4 4 4 4 5 5 5 6 7 7 8 9 9 12 12 32 55 88
```

### 4.3 Struktur von Algorithmen

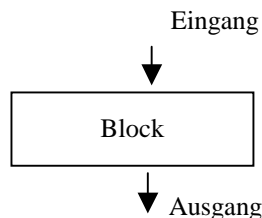
Algorithmen können aus wenigen elementaren Anweisungen - Ein- und Ausgabeanweisungen sowie Wertzuweisungen an Variablen - mit Hilfe weniger Steuerstrukturen aufgebaut werden. Diese grundlegenden Steuerstrukturen finden sich in allen algorithmenorientierten (d.h. imperativen, prozeduralen) Programmiersprachen in ähnlicher Form.

Beschränkt man sich beim Entwurf eines Algorithmus' auf diese wenigen Grundelemente, so bereitet die Übersetzung in eine Programmiersprache keine großen Schwierigkeiten mehr.

Statt „Steuerstruktur“ wird häufig auch der Begriff „**Kontrollstruktur**“ verwandt. Dies erklärt sich aus der englischen Bezeichnung „control structure“, control = Steuerung.

#### Blockorientierter Entwurf

- Algorithmen werden aus Blöcken aufgebaut.
- Ein Block beschreibt eine Anweisung; diese kann einfach oder komplex sein.
- Einfache Blöcke stehen für elementare Anweisungen (Ein- und Ausgabeanweisungen, Wertzuweisungen...) oder selbstdefinierte Prozeduren (die selbst wieder in komplexer Weise zusammengesetzt sein können)
- Komplexe Blöcke werden aus einfacheren nach wenigen Regeln zusammengesetzt. Diese Regeln spiegeln die Steuerstrukturen wider, die man in allen prozeduralen Programmiersprachen findet.
- Jeder Block hat genau einen Eingang und einen Ausgang.
- Blöcke werden nur kantendeckend aneinander gesetzt.

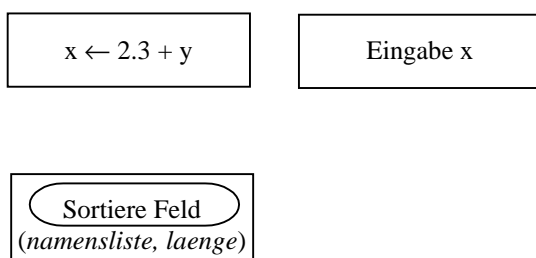


Im folgenden wird der Aufbau von Algorithmen aus Grundelementen nach diesen Prinzipien in vier Formen dargestellt:

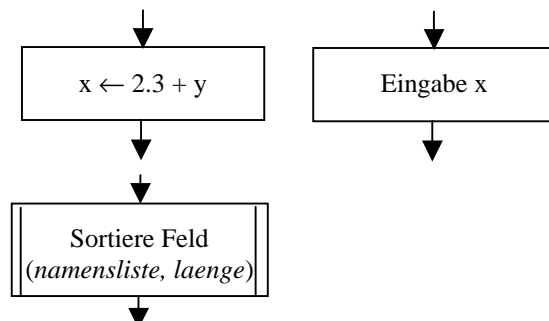
- Struktogramm (Nassi-Shneiderman-Diagramme → Prinzipien der strukturierten Programmierung)
- Programmablaufplan (Flußdiagramm nach DIN 66001, hier nur zur Erläuterung der inneren Funktion eines Blockes)
- Pseudocode (strenge umgangssprachliche Formulierung von Algorithmen nach den oben angegebenen Prinzipien)
- PASCAL (als eine konkrete Umsetzung in eine Programmiersprache)

#### 4.3.1 Einfache Blöcke (elementare Anweisungen, Prozeduranweisungen)

##### Struktogramm



##### Programmablaufplan



##### Pseudocode

Setze x auf den Wert 2.3+y

Eingabe x

Sortiere das Feld *namensliste*, *laenge*

##### Pascal

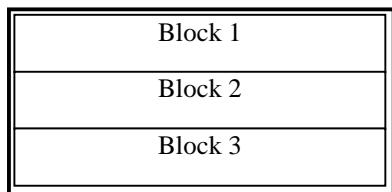
`x := 2.3 + y`

`readln(x)`

`Sortiere_Feld(namensliste, laenge)`

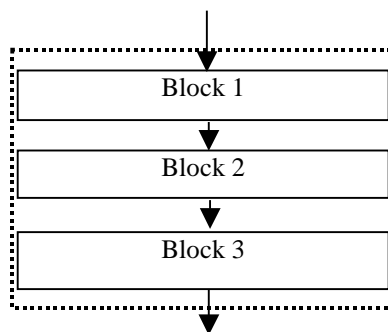
### 4.3.2 Sequenz (Verbundanweisung)

Struktogramm



Das ist **ein** Block

Programmablaufplan



Pseudocode

Anweisung 1  
Anweisung 2  
Anweisung 3

Pascal

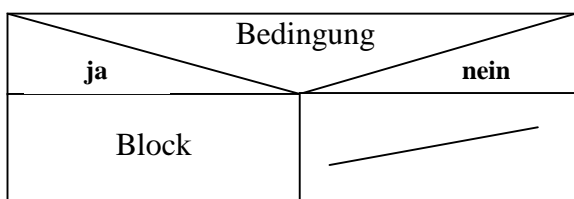
```
begin
  Anweisung 1 ;
  Anweisung 2 ;
  Anweisung 3 ;
end
```

} das ist **eine** Anweisung (Verbundanweisung)

### 4.3.3 Auswahl (Entscheidung, Fallunterscheidung)

#### 4.3.3.1 Einseitige Auswahl

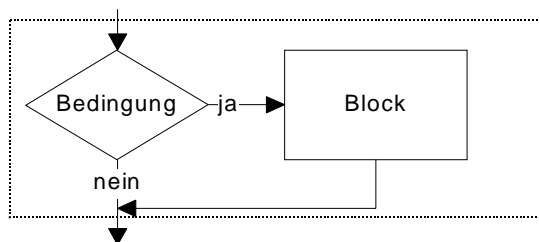
Struktogramm



Pseudocode

*Wenn* Bedingung *dann* Block

Programmablaufplan



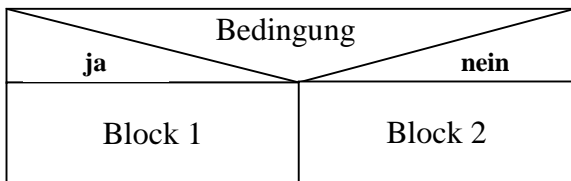
Pascal

```
if {Bedingung} then {Anweisung}
```

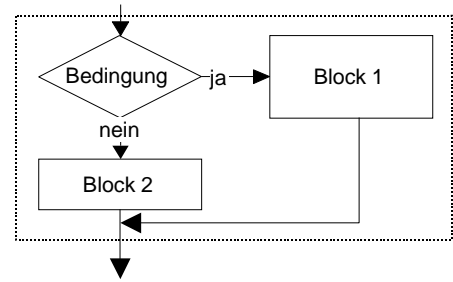
boolean expression      einfache oder Verbundanweisung

4.3.3.2 Zweiseitige Auswahl

Struktogramm



Programmablaufplan



Pseudocode

**Wenn** Bedingung **dann** Block 1  
**sonst** Block 2

Pascal

**if** Bedingung **then** Anweisung 1  
**else** Anweisung 2

oder in anderer Anordnung

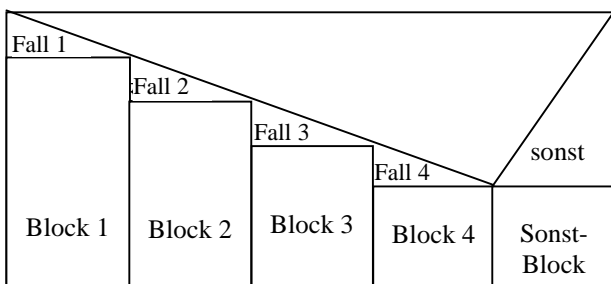
**Wenn** Bedingung **dann** Block 1  
**sonst** Block 2

**if** Bedingung **then** Anweisung 1  
**else** Anweisung 2

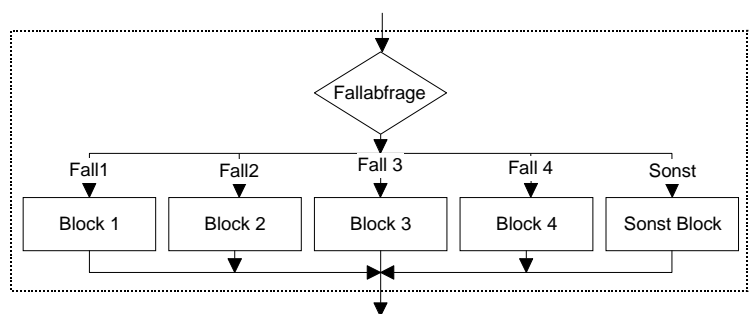
4.3.3.3 Mehrfache Auswahl (Fallunterscheidung)

1. Möglichkeit: Verschachteln von zweiseitigen Fallunterscheidungen.
2. Möglichkeit: Eigene Auswahlstruktur (case-Anweisung in PASCAL)

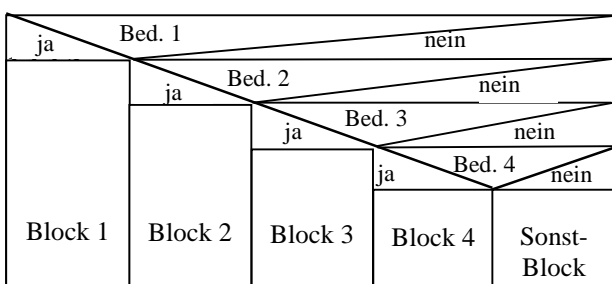
Struktogramm



Programmablaufplan



Diese Darstellung ist eine Abkürzung für folgende geschachtelte zweiseitige Fallunterscheidungen:



**if** Bedingung 1 **then** Anweisung 1  
**else if** Bedingung 2 **then** Anweisung 2  
**else if** Bedingung 3 **then** Anweisung 3  
**else if** Bedingung 4 **then** Anweisung 4  
**else** Sonst-Anweisung 2



## Pseudocode

**Falls**

Fall 1: Block 1  
 Fall 2: Block 2  
 Fall 3: Block 3  
 Fall 4: Block 4

**sonst:** Sonst-Block

## Pascal (konkretes Beispiel)

**case** zeichen **of**

```
'A' :   writeln('Das war ein A');
'B' :   writeln('Das war ein B');
'C..'Z': writeln('Das war kein A und kein B');
```

**else**

```
writeln('Das war kein Großbuchstabe')
```

**end**

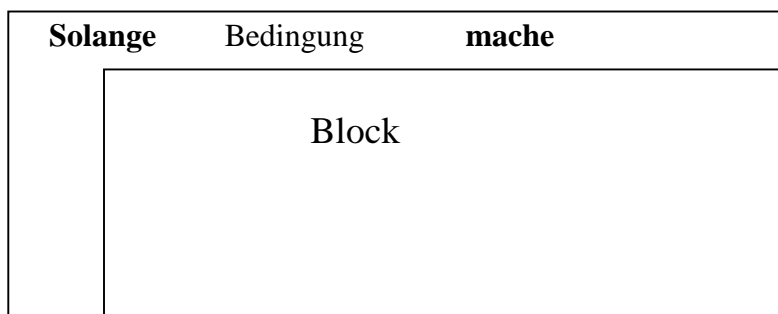
## Bemerkungen:

- der *else*-Teil darf fehlen
- es sind nicht alle Datentypen für die Fälle zugelassen

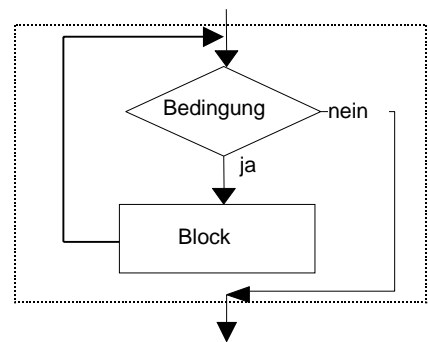
## 4.3.4 Wiederholungen, Schleifen

## 4.3.4.1 Kopfgesteuerte Schleife (abweisende Schleife)

## Struktogramm



## Programmablaufplan



## Pseudocode

**Solange** Laufbedingung **wiederhole**  
 Block von Anweisungen

## Beispiel 1

**var** i: integer;

```
i:=1;
while i<=100 do
  begin
    writeln(i);
    i:=i+1;
  end;
```

## Pascal

**while** Laufbedingung **do**  
 Anweisung

## Beispiel 2

**var** x: real;

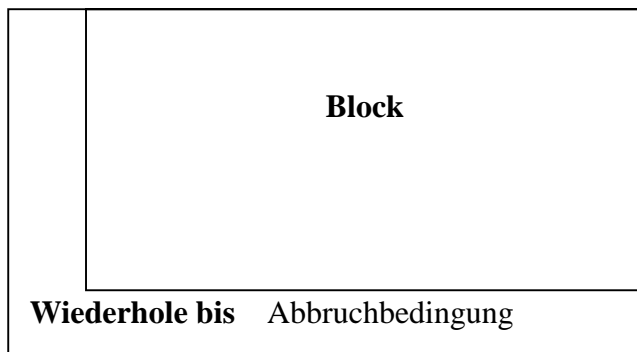
```
x:= - 1;
while x<=3 do
  begin
    writeln(x, ' ', x*x);
    x:=x+0.2;
  end;
```

**Beachte:**

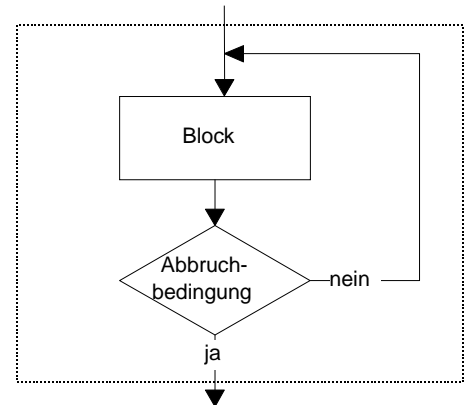
- Die Laufbedingung wird **vor** der ersten Ausführung des Blocks getestet → **Kopfsteuerung**
- Der Block **muß nicht ausgeführt werden**, wenn die Laufbedingung schon von vornherein falsch ist → **abweisende Schleife**

#### 4.3.4.2 Fußgesteuerte Schleife (nicht abweisende Schleife)

##### Struktogramm



##### Programmablaufplan



##### Pseudocode

```

Wiederhole
  Anweisung
  Anweisung
  .
  .
bis Abbruchbedingung
  
```

##### PASCAL

```

repeat
  Anweisung ;
  Anweisung ;
  .
  .
until Abbruchbedingung
  
```

##### Beispiel 1

```

var i: integer;

i:=1;
repeat
  writeln(i);
  i:=i+1;
until i>100;
  
```

##### Beispiel 2

```

var x: real;

x:= - 1;
repeat
  writeln(x, ' ', x*x);
  x:=x+0.2;
until x>3;
  
```

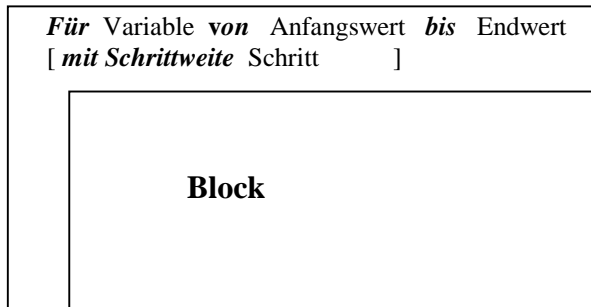
##### Beachte:

- Die Abbruchbedingung wird **nach** dem Durchlaufen des Blocks getestet → **Fußsteuerung**
- Der Block wird **mindestens einmal durchlaufen**, auch wenn die Abbruchbedingung schon von vorneherein falsch ist → **nicht abweisende Schleife**
- Abweisende Schleifensteuerung ist das allgemeinere Prinzip, auf das sich nicht abweisende Schleifen zurückführen lassen. Daher kann man bei Programmiersprachen auf letztere verzichten. Einer der frühen BASIC-Dialekte, das seinerzeit weit verbreitete GW-BASIC, kannte keine fußgesteuerten Schleifen.

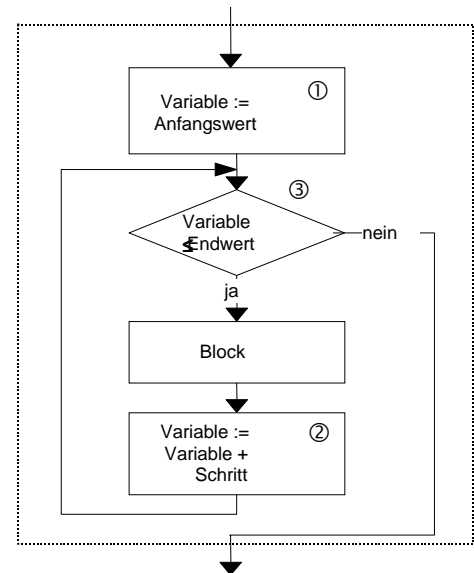
### 4.3.4.3 Zählschleife

Zählschleifen sind Abkürzungen von Schleifen, bei denen für die Initialisierung ①, das Weiterzählen ② und die Laufbedingung ③ eine feste Form vorgegeben ist. Zählschleifen werden benutzt, wenn die Durchlaufszahl schon zu Beginn bekannt ist. Zählschleifen sind in der Regel abweisende Schleifen.

#### Struktogramm



#### Programmablaufplan



#### Pseudocode

*Für* Variable *von* Anfangswert *bis* Endwert  
[ *mit Schrittweite* Schritt ]  
Anweisungen

#### Pascal

*For* i:= Anfangswert *to* Endwert *do*  
Anweisung

#### Beachte:

- In PASCAL sind keine Schrittweiten außer +1 (mit *to*) oder -1 (mit *downto* anstelle von *to*) möglich.
- Als Zählvariable sind nur Variable vom sogenannten ordinalen Datentyp zugelassen, also Variablen vom Typ *integer*, *word*, oder *character*, aber keine Variablen vom Typ *real*.
- In BASIC sind beliebige Schrittweiten in Zählschleifen vorgesehen, die Variablen dürfen auch vom Typ *real* sein:

```
for x = -1.2 to 3 step 0.2
  Anweisung
  Anweisung
  .
  .
next x
```

### 4.3.5 Steuerstrukturen in der Sprache C und JAVA

#### Verbundanweisung (Sequenz, „Block“)

```
{ Anweisung Anweisung .... Anweisung }
```

entspricht *begin ..... end* in PASCAL

#### Auswahl

##### Einseitige /zweiseitige Auswahl

```
if ( Ausdruck ) Anweisung [ else Anweisung ] ;
```

##### Mehrseitige Auswahl ( Fallunterscheidung)

```
switch ( Ausdruck ) { case Konstante : Anweisung Anweisung ...
                      case Konstante : Anweisung Anweisung ...
                      default : Anweisung Anweisung ... }
```

#### Wiederholungen, Schleifen

##### Abweisende Schleife

```
while ( Ausdruck ) Anweisung
```

```
while ( Ausdruck ) { Anweisung Anweisung ..... }
```

##### Nicht abweisende Schleife

```
do ( Anweisung ) while ( Ausdruck ) ;
```

mit Laufbedingung anstelle einer Abbruchbedingung bei der repeat- Anweisung in PASCAL

##### Zählschleife

```
for ( Ausdruck 1 ; Ausdruck 2 ; Ausdruck 3 ) Anweisung
```

sehr allgemeine Zählschleife

Die Zählschleife ist äquivalent zu:

```
Ausdruck 1
while ( Ausdruck 2 ) { Anweisung ; Ausdruck 3 }
```

##### Bemerkung:

Die Sprache C ist wesentlich flexibler als PASCAL und läßt sehr kurzen, aber auch schwerer lesbaren Programmcode zu. Ein Beispiel für eine Schleife, bei der gleichzeitig Wertzuweisungen und eine Eingabeanweisung ausgeführt werden:

```
while ( c = d = getchar() != 'A' ) .....
```

*getchar()* liest ein Zeichen von der Tastatur, der Wert wird den Variablen c und d zugeordnet und gleichzeitig geprüft, ob dieses Zeichen ein 'A' ist; falls dies nicht der Fall ist, wird die Schleifenanweisung weiter ausgeführt.

## 4.4 Auswahl von Turbo Pascal Anweisungen mit Beispielen

Das folgende Verzeichnis enthält einige grundlegende Anweisungen, Prozeduren und Funktionen von TURBO Pascal. Die Befehle sind nach Einsatzgebieten geordnet.

Viele Standard-Prozeduren und –Funktionen und die Ein/Ausgabeprozeduren sind in der Unit CRT enthalten. Programme müssen daher immer den Zusatz *uses CRT;* nach dem Programmkopf aufweisen.

### 4.4.1 Grundstrukturen zur Steuerung von Algorithmen und grundlegende Anweisungen

#### *begin.....end*

Klammerung mehrerer Anweisungen mit *begin* und *end* zu einer einzigen Verbundanweisung:

- *if num=3 then begin a:=1; c:=5; d:=9 end;*

#### *if ..... then ..... else*

Wenn der Boolesche Ausdruck *num=3* wahr ist, dann .....

- *if num=3 then a:=-1* Einseitige Auswahlstruktur.  
 - *if num=3 then a:=1 else a:=2* Zweiseitige Auswahlstruktur.

#### *case.....of.....else.....end*

Kontrolle der mehrseitigen Auswahlstruktur bzw. Fallabfrage:

- *case* Eingabe *of*  
   'a': unterprozedur1;                   Je nach dem Zeichen in Eingabe  
   'b': unterprozedur 2;               eine der Prozeduren aufrufen,  
*else* fehlermeldung;               sonst Fehleingabe bearbeiten.  
*end;*

#### *for ..... to .... do*

Zählschleife mit Zählervariable z, i, b

-*for z:=1 to 5 do* writeln               Fünf Leerzeilen ausgeben.  
 -*for i:='a' to 'c' do* writeln       Drei Leerzeilen ausgeben.  
 -*for b:=true to false do* writeln   Zwei Leerzeilen ausgeben.

#### *repeat ... until*

Steuerung einer nicht-abweisenden Schleife:

- *repeat ... until b*                   Wiederhole die zwischen *repeat* und *until* stehenden Anweisungen, **bis** die boolesche Variable b wahr wird.

#### *while ..... begin ..... end*

Steuerung einer abweisenden bzw. „Solange“-Schleife:

- *while b begin ... end*               **solange** b wahr ist, werden die Anweisungen zwischen *begin* und *end* ausgeführt.

#### **Wertzuweisung** *num:=3+6*

Ergebnis von 3+6 ermitteln und der Variablen num zuweisen:

- *num:=3+6* weist 9                   in die Variable num als neuen Wert zu.  
 - *num:=num+3+6*                   erhöht den derzeitigen Wert von num um 9.  
 - *st:='Fido'*                   weist den String ‚Fido‘ der Variablen st zu.

#### **goto 4**

Zum Label (Sprungmarke) 4 im Programmtext verzweigen:

-*goto 4* fährt an der mittels 4: markierten Stelle fort.  
 → nach allen Regeln des Anstandes verboten!!!! (strukturierte Programmierung, objektorientierte Programmierung!)

## 4.4.2 Arithmetische Funktionen

### abs(num)

Absolutwert von num ausgeben mit Ergebnistyp wie num:

- abs(-3) ergibt 3; Ergebnis vom *integer*-Typ.
- abs(-3.4444) ergibt 3.4444; Ergebnis vom *real*-Typ.

### frac( 2.445)

Nachkommastellen von 2.445 angeben mit *real*-Ergebnistyp

- n:=frac(2.445) weist 0.445 bzw. 2.445-int(2.445) in n zu.
- write(frac(-2.445) ) gibt -0.445 aus.

### int(2.778)

Ganzzahligen Teil von 2.778 mit *real*-Ergebnistyp angeben:

- writeln(int(2.778)) gibt 2.000 aus.
- g:=int(-2.778) weist -2.000 der *real*-Variable g zu

### sqr(-1 .5)

Quadrat von -1.5 angeben mit *real*-Ergebnistyp:

- write(sqr(-1.5)) gibt 2.25 aus, also (-1.5)\*(-1.5).

### sqrt( 16)

Quadratwurzel von 16 angeben mit *real*-Ergebnistyp:

- w:=sqrt(16) weist 4.00 der *real*-Variable w zu.

### sin(pi/2)

Sinus von pi/2 mit *real*-Ergebnistyp angeben:

- writeln(sin(pi/2)) gibt 1.00 als sin des rechten Winkels.

### cos(num)

Cosinus von num angeben mit *real*-Ergebnistyp:

- write(cos(pi/2)) gibt 0.00 aus.

### arctan ( num)

Arcustangens in Grad angeben mit *real*-Ergebnistyp. Vorsicht: Die Tangensfunktion ist in Turbo-Pascal nicht enthalten.

### exp( 1 )

Exponent e hoch 1 mit *real*-Ergebnistyp ermitteln:

- write(exp(1)) gibt 2.7182818285 aus.

### ln(2.7182818285)

Natürlicher Logarithmus mit *real*-Ergebnistyp angeben:

- write(ln(2.7182818285)) gibt 1 aus.

## 4.4.3 Stringverarbeitung

### copy('Wegweiser',5,3)

Aus String 'Wegweiser' ab 5. Stelle 3 Zeichen entnehmen:

- write(copy('Wegweiser' ,5,3)) ergibt ,eis' als Teilstring.
- write(copy('Wegwei5er' ,5,100)) ergibt ,eiser'
- write(copy('Turbo',100,2)) ergibt einen Leerstring.

Funktion

### concat( 'Tu' , 'r' , 'bo' ) oder auch 'Tu'+ 'r'+ 'bo'

Durch Komma getrennte Teilstrings zu einem String verketteten:

- write(concat('Tu' , 'r' , , 'bo')) ergibt 'Turbo'.
- write(concat('Tu' ,ST) mit ST:= 'rbo' ergibt 'Turbo'.
- write('Tu'+ 'r'+ 'bo' ) ergibt ebenfalls 'Turbo'.

Funktion

**delete(st,4,3)**

In Stringvariable st ab der 4. Position 3 Stellen löschen:

- delete(st,4,3) mit st:='Wegweiser' ergibt ,Wegser' in st..
- delete(st,3,20) mit st:='Wegweiser' ergibt ,We' in st.

Prozedur

**insert('ist', st,6)**

Den String ,ist' in den Wert der Stringvariable st ab 6. Position einfügen:

- insert('ist', st,6) mit st:='Turbogut' gibt 'Turboistgut'.
- insert('ist', st,100) ergibt ,Turbogutist'.

Prozedur

**length('Wegweiser')**

Die Länge eines Strings als *integer*-Wert angeben:

- write(length('Wegweiser')) ergibt 9.
- l:=length('Turbo') weist 5 in *integer*-Variable l zu.

Funktion

**pos('eis', 'Wegweiser')**

Position von ,eis' im String ,Wegweiser' angeben:

- p:=pos('eis', 'Wegweiser') weist *integer*-Wert 5 in p zu.
- p:=pos( ,eiss' ,Wegweiser') weist 0 als Fehler in p zu.
- write(pos('T','Turbo')) gibt *integer*-Zahl 1 aus.

Funktion

**str(x,st)**

*integer*- oder *real*-Zahl x in String st umwandeln:

- str(x, st) ergibt '2000' in st , wenn x=2000 ist..
- str(x:8:2, st) ergibt '2000.55' in st wenn x=2000.553 ist (formatierte Umwandlung).

Prozedur

**val('99', num, fehl)**

- String '99' als *integer*- oder *real*-Zahl in num ablegen und eventuelle Fehlerposition fehl melden:
- val('99', num, fehl) ergibt 99 in num und 0 in fehl.
- val('9R', num, fehl) ergibt 2 in fehl als Fehlerposition.

Prozedur

**upcase('t')**

Umwandlung von Klein- in Großschreibung; ergibt hier 'T'. Die Eingabe muss vom Typ *char* sein

Funktion

#### 4.4.4 Typen-Umwandlungsfunktionen

**chr(66)**

Für ASCII-Ordnungszahl 66 das zugehörige Zeichen angeben, d.h. *integer*-Typ in *char*-Typ umwandeln bzw. transferieren:

- writeln(chr(66)) gibt das Zeichen ,B' am Bildschirm aus.
- c:=chr(53) weist Zeichen '5' der *char*-Variable c zu.

**ord('B')**

Ordnungsnummer von ,B' in dem zugehörigen Datentyp angeben, d.h. von Skalar-Typ in *integer*-Typ umwandeln (vgl. CHR):

- i:=ord('B') weist Nummer 66 in *integer*-Variable i zu.
- write(ord('5')) gibt Ordnungsnummer 53 aus ('5' ist vom Typ *char*).
- write(ord(5)) gibt Ordnungsnummer 5 aus ( 5 ist vom Typ *integer*).

**round(-3.9)**

*real*-Wert -3.9 zum *integer*-Wert -4 aufrunden (vgl. trunc)

**trunc(-3.9)**

*real*-Wert -3.9 zum *integer*-Wert -3 abschneiden (vgl. round).

#### 4.4.5 Standard- Prozeduren und Funktionen

**clrscr**

Bildschirm löschen und Cursor in Ausgangsposition links oben:

- clrscr                      Ausgangsposition (1,1).      clrscr für clearscreen.

**delay(100)**

Warteschleife erzeugen, die ca. 100 Millisekunden lang läuft

- delay(100)                      Argument in Klammern muß *integer* sein.

**sound(1000)**

Lautsprecher mit 1000 Hz einschalten

- sound(1000); delay(2000); nosound;      ergibt einen Piepston mit 1000 Hz und 2 Sekunden Dauer.

**nosound**

Schaltet den Lautsprecher ab.

**randomize**

Zufallsgenerator mit einer Zufallszahl starten.

**random**

Zufallszahl vom *real*-Typ zwischen 0 und 1 erzeugen:

- r:=random                      weist in *real*-Variable r z.B. 0.3451 zu.

**random( 100 )**

Zufallszahl vom *integer*-Typ zwischen 0 und 99 erzeugen:

- write(random(100))      gibt z.B. 49 als Zufallszahl aus.

**sizeof(st)**

Zahl der durch Variable st im Speicher belegten Bytes:

- write(sizeof(st))      gibt 8 aus (für st:string[7]).
- write(sizeof(r))      gibt 6 aus (für Vereinbarung r:*real*).

**succ(-7)**

Nachfolger zu -7 angeben (Umkehrung zu Funktion pred):

- write(succ(-7))      gibt -6 aus.

**pred(-7)**

Vorgänger von -7 (einfacher, abzählbarer Skalar-Typ) angeben:

- v:=pred(-7)      weist -8 der *integer*-Variable v zu.
- c:=pred('e')      weist ,d' der *char*-Variable c zu.

#### 4.4.6 Ein- und Ausgabeprozeduren und Funktionen für die Console

**gotoxy(4,20)**

Cursor zur 4. Zeile von oben, 20. Spalte von links bewegen:

- gotoxy(4,20)      Cursor zur Position (4,20) bewegen.
- gotoxy(1,1)      Cursor zur Home-Position links oben bewegen.

**keypressed**

Boolesche Funktion mit Ergebnis *true* für ‚Taste gedrückt‘:

- repeat until keypressed      ergibt *true*, sobald Tastendruck erfolgt.



**readkey**

Tastatureingabe ohne „Echo“ auf dem Bildschirm und ohne „ENTER“ einlesen.

- `ch:=readkey`                      Wartet, bis eine Taste auf der Tastatur betätigt wird und speichert das Ergebnis dann in der Variablen `ch`, die vom Typ *char* sein muß.
- `readkey`                             In TURBO-PASCAL 6 ist auch ein Funktionsaufruf ohne Zuweisung des Ergebnisses auf eine Variable möglich. Mit `readkey` kann man den Programmablauf anhalten bis eine Taste gedrückt wird.

**readln(ein)**

Tastatureingabe wie READ, aber mit „RETURN“ am Ende.

- `readln(ein)`                        Cursor nach Eingabe am Anfang der Folgezeile.

**write(aus)**

Ausgabe des Inhalts der Variablen `aus` am Bildschirm:

- `write('Summe: ', aus)`                Ausgabe von Konstante und Variable.
- `write('Summe: ',aus:10:2)`           Variableninhalt bei der Ausgabe „formatieren“:  
Ausgabefeld 10 Zeichen breit, bei *real*-Werten 2 Nachkommastellen.

**writeln(aus)**

Wie `write(aus)`, aber mit „RETURN“ am Ende der Ausgabe:

- `writeln(aus)`                        Cursor steht dann am Anfang der Folgezeile.
- `writeln`                                Leerzeile ausgeben.
- `writeln(LST, aus)`                   Ausgabe von `aus` auf den Drucker leiten mit LST (Lister) als logische Geräteeinheit.

**Bemerkungen:**

Die Prozeduren aus der Unit CRT beziehen sich auf die Ein/Ausgabe von Daten über Tastatur und Bildschirm. Diese Prozeduren stehen in DELPHI für Windows in dieser Form nicht mehr zur Verfügung, da dort die zu Grunde liegende „Philosophie“ eine ganz andere in als TURBO PASCAL für DOS ist:

- Mehrere Prozesse können parallel ausgeführt werden (Multitasking),
- Benutzerschnittstelle mit einer Fülle graphischer Elemente wie Fenster, Schaltflächen und Auswahlfelder,
- Ereignissteuerung.

Prozeduren wie `writeln` oder `readln` für den Bildschirm oder die Tastatur machen in diesem Zusammenhang keinen Sinn mehr. Die interne Verarbeitung (etwa Verarbeitung von Strings oder die Ausführung mathematischer Berechnungen) wird jedoch noch weitgehend in der gleichen Weise wie in TURBO PASCAL für DOS erfolgen. Fertigkeiten in der Programmierung solcher Verfahren sind daher auch in den „modernerer“ Sprachen noch immer gefragt und nötig, und sie lassen sich auch auf andere Sprachen wie C oder Java übertragen.

Wegen des einfacheren Rahmens für die Programmierumgebung von TURBO PASCAL für DOS empfiehlt sich diese Sprache noch immer zur Einführung in die Programmierung.

In der Liste der oben angegebenen grundlegenden Prozeduren und Funktionen fehlen gänzlich:

- Anweisungen für Zeigerstrukturen (Dynamische Variablen)
- Ein/Ausgabeweisungen für Dateien (Files)

## 5 Prozeduren und Funktionen

### 5.1 Einfachste Prozeduren und Funktionen

#### 5.1.1 Prozeduren als Abkürzung für Blöcke von Anweisungen

**Deklaration:**

```
procedure beep;
begin
  sound(1000); { Lautsprecher ein mit 1000 Hz }
  delay(500);  { Pause 500 ms }
  nosound;    { Lautsprecher aus }
end;
```

**Aufruf:**

```
beep;
```

#### 5.1.2 Funktionen als Abkürzung für Rechenausdrücke

**Deklaration:**

```
function kubik(x : real) : real;
begin
  kubik:=x*x*x; { Funktionswert zurückgeben }
end;
```

**Aufruf:**

```
k := 3*kubik(4);
s := 3*kubik(2+u);
```

#### 5.1.3 Funktionen, die nicht durch einen Rechenausdruck darstellbar sind

**Deklaration:**

```
function zehnhoch(n:integer) : real;
  var i : integer ;
  pot : real ;
begin
  pot := 1 ; { Funktionswert berechnen }
  IF n>=0 then FOR i:= 1 TO n DO pot := pot*10
  else FOR i:=-1 DOWNTON n DO pot := pot/10 ;
  zehnhoch := pot; { Funktionswert zurückgeben }
end;
```

**Aufruf:**

```
u := 3*zehnhoch(4*y-2);
IF zehnhoch(z-1) < 1000 then .....
```

## 5.2 Prozeduren und Funktionen mit Parametern

### 5.2.1 Deklaration und Aufruf: Formale und aktuelle Parameter

#### Deklaration:

```
function zehnhoch(n:integer) : real;
```

↑  
n ist formaler Parameter in der Deklaration

#### Aufruf:

```
u := 3*zehnhoch(4*y-2);
```

↑  
Der Ausdruck 4\*y-2 ist aktueller Parameter beim Aufruf

#### Deklaration:

```
procedure beep(frequ, dauer:integer);
begin
  sound(frequ); .{ Lautsprecher ein mit frequ Hz }
  delay(dauer); { Pause dauer ms }
  nosound;      { Lautsprecher aus }
end;
```

#### Aufruf:

```
beep(1000,500);
```

Der aktuelle Parameter ist der Parameter, der beim Aufruf der Funktion oder der Prozedur für den formalen Parameter aus der Deklaration eingesetzt wird.

Bei der Deklaration einer Prozedur müssen die Datentypen für die beim Aufruf übergebenen Daten im Prozedurkopf festgelegt werden.

Bei der Deklaration einer Funktion müssen die Datentypen für die beim Aufruf übergebenen Daten und der Ergebnistyp im Kopf der Funktionsdeklaration festgelegt werden.

### 5.2.2 Gültigkeitsbereich von Variablen: Lokale und globale Variablen

Im Hauptprogramm ist die Variable `temp` vom Typ `array [1..100] of real` deklariert, die ein Feld von Temperaturwerten repräsentieren soll. Die Anzahl der tatsächlich erfassten Werte sei in der Variablen `anzahl` gespeichert. Die folgende Prozedur soll den Mittelwert der Temperaturen berechnen und ausgeben.

```
procedure mittel_berechnen;
  var i      : integer;
      mittel : real;

begin
  mittel:=0;
  for i:=1 to anzahl do mittel := mittel + temp[i];
  mittel:=mittel/anzahl;
  writeln;
  writeln('Der Mittelwert beträgt ', mittel:10:2);
end;
```

Die Variablen `i` und `mittel`, die in der Prozedur deklariert sind, sind nur innerhalb der Prozedur `mittel_berechnen` gültig und heißen lokale Variablen. Die Variablen `temp` und `anzahl` aus dem Hauptprogramm heißen globale Variablen. Ist `i` auch im Hauptprogramm (global) benutzt worden, so wird dieses globale `i` nicht durch das lokale `i` aus der Prozedur verändert.

Auch die formalen Parameter im Kopf einer Prozedur- oder Funktionsdeklaration sind nur lokal (d.h. in der Funktion oder Prozedur) gültig.

#### Globale Variablen:

Variablen, die im Hauptprogramm deklariert werden. Sie sind im ganzen Programm gültig.

**Lokale Variablen:**

Variablen, die im Kopf oder im Deklarationsteil einer Prozedur oder Funktion deklariert werden. Sie sind nur in der sprechenden Prozedur oder Funktion gültig. Veränderungen ihrer Werte haben keinen Einfluß auf Variablen gleichen Namens im übergeordneten Programm.

**5.3 Arten der Parameterübergabe: Wert- und Variablenparameter**

Bei der Übergabe von Parametern an Prozeduren und Funktionen unterscheidet man die folgenden beiden Arten :

**Wertparameter, Call by Value:**

Nur ein fester **Wert** wird an die Prozedur übergeben. Die Prozedur richtet dafür eine eigene, lokale Variable ein und arbeitet mit diesem Wert. Rückwirkungen auf Variablen im aufrufenden Hauptprogramm sind so nicht möglich, insbesondere kann auch damit kein Ergebnis an das Hauptprogramm zurückgegeben werden.

**Variablenparameter, Referenzparameter, Call by reference:**

Eine **Variable** wird an die Prozedur übergeben. Die Prozedur benutzt diese Variable für ihre Arbeit. Der Wert dieser Variablen kann von der Prozedur verändert werden. Rückwirkungen auf das aufrufende Hauptprogramm sind so möglich, insbesondere können so Ergebnisse an das Hauptprogramm zurückgegeben werden.

Die unterschiedliche Art der Übergabe wird durch den Zusatz *var* im Prozedurkopf gekennzeichnet.

Beispiel:

```

program beisp;
  var u,v : integer;

  procedure erhoehe(x:integer ; var y:integer);
  begin
    x := x+1 ;   y := y+1 ;
    write( x:4 ) ; writeln( y:4 ) ;
  end;

begin
  u := 5 ;   v := 5 ;
  writeln( u:4 , v:4 );
  erhoehe(u,v);
  writeln( u:4 , v:4 );
end.

```

Man beachte die Ausgaben des Programms !

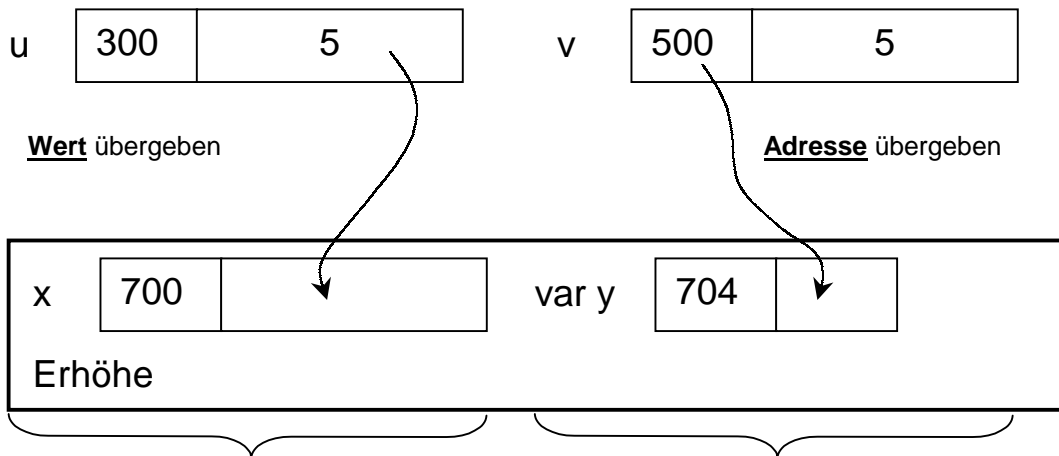
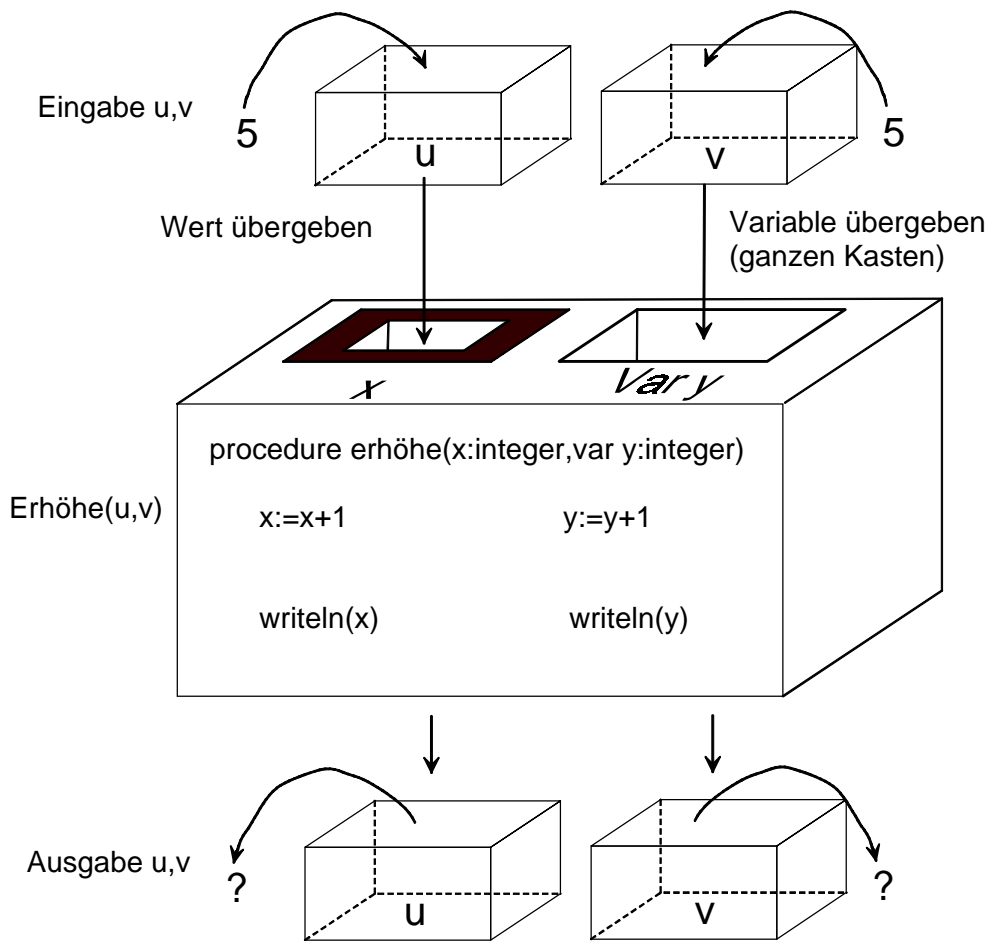
**5.4 Verwendung der Parameter****Eingangs-, Ausgangs-, Durchgangsparemeter**

Hinsichtlich der **Verwendung** der Parameter in einer Prozedur ist zu unterscheiden :

- **Eingangsparameter:** Nur zur Eingabe in die Prozedur.
- **Ausgangsparameter:** Nur zur Ausgabe aus der Prozedur an das Hauptprogramm.
- **Durchgangsparemeter:** Sowohl zur Eingabe in die Prozedur als auch zur Rückgabe von Werten aus der Prozedur.

Da eine Prozedur Wertparameter nicht so verändern kann, dass diese Änderung auch im Hauptprogramm wirksam bleibt, muß für Ausgangsparemeter und Durchgangsparemeter die Übergabe als Variablenparameter erfolgen.

**Veranschaulichung der verschiedenen Arten von Parameterübergabe an Prozeduren und deren Verhalten am Beispiel der Prozedur „Erhöhe“.**



Prozedur kann u **nicht** verändern

Prozedur **kann** v verändern, da sie „weiß“, wo der Wert von v steht

Wertparameter,  
Call by value

Referenzparameter,  
call by reference

## 5.5 Funktionen und Prozeduren im Überblick

### 5.5.1 Prozeduren, die eigentlich Funktionen sein sollten

Anstelle einer Funktion kann auch eine Prozedur definiert werden .

Beispiel :

```
function kubik(x : real) : real;
begin
  kubik := x*x*x ;
end;
```

Aufruf z.B.: u := kubik(4+y) ;

Als Prozedur:

```
procedure kubik(x : real ; var erg : real);
begin
  erg := x*x*x ;
end;
```

Aufruf z.B.: kubik(4+y , u) ;

Da als Ergebnistypen für Funktionen nur einfache Typen zugelassen sind (also keine *arrays* oder *RECORDs*), muß man eine Funktion mit komplexem Ergebnistyp als Prozedur schreiben.

### 5.5.2 Funktionen und Prozeduren mit Nebenwirkungen: Programmierstil

Wenn eine Funktion oder eine Prozedur den Wert einer globalen Variablen verändert, ohne dass diese Variable beim Aufruf explizit übergeben wird, spricht man von **Nebenwirkungen** .

Unsere ersten Beispielprogramme verwenden nur Prozeduren mit Nebenwirkungen.

Bei Nebenwirkungen besteht die Gefahr, dass Werte von Variablen unbeabsichtigt und unbemerkt verändert werden. Nennt man allerdings bei jeder Prozedur **alle** von ihr veränderten Variablen im Kopf explizit, dann entstehen unübersichtliche Programme.

Man sollte daher folgende Prinzipien befolgen :

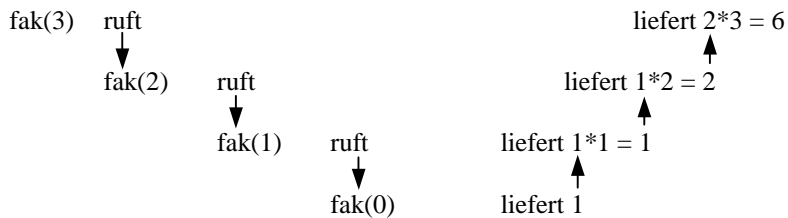
- unwichtige Variablen wie Zählvariablen in Schleifen usw. unbedingt als lokale Variablen in der Prozedur definieren,
- Variablen, deren Bedeutung während des gesamten Programms klar ist ( z.B. einer Tabelle in den ersten Beispielprogrammen) können als globale Variablen von Prozeduren verändert werden und müssen nicht an jede Prozedur explizit übergeben werden,
- werden Prozeduren häufig mit verschiedenen Werten aufgerufen, z.B. mit Koordinaten, so sind diese Werte explizit an die Prozedur zu übergeben.

## 5.6 Rekursive Funktionen und Prozeduren

Zur Erklärung ein Beispiel : Die Fakultätsfunktion .

```
function fak( n:integer ) : integer ;
begin
  IF n<=0  then fak := 1
            else fak := fak(n-1)* n ;
end.
```

Die Definition der Funktion fak ruft die Funktion fak selbst wieder auf, allerdings mit einem kleineren Argument. Ruft sich eine Funktion oder Prozedur selbst auf so spricht man von einer rekursiven Definition. Bei einer rekursiven Definition muß gewährleistet sein, dass diese Aufrufe nicht endlos wiederholt werden, sondern bei irgendeinem Wert aufhören. Im Beispiel geschieht dies bei  $n \leq 0$  .



## 5.7 Beispielprogramm zum Testen einfacher Prozeduren und Funktionen

```

program Beispiel;
uses crt;

var r:real;
    n:integer;

procedure beep;
begin
  sound(1000);
  delay(500);
  nosound;
end;

function hochdrei(x:real):real;
{Berechnet x hoch 3}
begin
  hochdrei:=x*x*x;
end;

function zehnhoch(n:integer):real;
{Berechnet 10 hoch n}
var
  i:integer;
  pot:real;
begin
  pot:=1;
  if n>=0 then for i:=1 to n do pot:=pot*10
  else for i:=-1 downto n do pot:=pot/10;
  zehnhoch:=pot;
end;

begin {Hauptprogramm}
  clrscr;
  Writeln('Hochdrei:');
  write('Zahl r eingeben: '); readln(r);
  writeln('r hoch drei ist ',hochdrei(r):0:2);
  beep;
  Writeln('Zehnhoch:');
  write('Eine ganze Zahl n eingeben: '); readln(n);
  writeln('10 hoch n ist ',zehnhoch(n):0:2);
  beep;
end.

```

## 5.8 Prozeduren und Funktionen in Turbo Pascal: Aufgaben zur String - Bearbeitung

Die folgenden Definitionen von Prozeduren und Funktionen zur String-Bearbeitung sind aus dem Handbuch von Turbo-Pascal entnommen. Auf diese Art sind dort alle Funktionen dokumentiert.

### Length

Deklaration **function** Length(S: string): Integer;

Funktion: Liefert die dynamische Länge eines Strings zurück.

Erläuterung Das Funktionsergebnis entspricht der Anzahl der in s enthaltenen Zeichen.

Beispiel

```
var s: string;
begin
  readln(s);
  writeln(s);
  writeln('Länge von s = ',Length(s));
end;
```

### Copy

Deklaration **function** Copy(s: string; Index, Count: Integer): string;

Funktion: Liefert einen Teil eines Strings zurück.

Erläuterung *Copy* liefert *Count* Zeichen ab der durch *Index* angegebenen Position des strings *s* zurück. *s* ist ein string-Ausdruck, *Count* und *Index* sind Integer-Ausdrücke. Wenn *Index* größer als die momentane Länge von *s* ist, liefert *Copy* einen leeren String; wenn *s* kürzer als *Index* + *Count* ist, werden nur die Zeichen zurückgeliefert, die *s* ab der durch *Index* angegebenen Position enthält. Beispiel

Beispiel

```
var s: string;
begin
  s := 'ABCDEF';
  s := Copy(s, 2, 3); { s:= 'BCD' }
end;
```

### Pos

Deklaration **function** Pos(Substr: string; s: string): Integer;

Funktion: Sucht einen String nach einer Zeichenfolge ab.

Erläuterung *Pos* sucht den als *s* übergebenen String-Ausdruck nach dem ersten Vorkommen des String-Ausdrucks *Substr* ab. Wenn *Substr* in *s* vollständig enthalten ist, dann liefert *Pos* die Position des Zeichens in *s* zurück mit dem *Substr* beginnt. Wenn *Substr* nicht in *s* enthalten ist, dann hat *Pos* das Funktionsergebnis 0.

Beispiel

```
var s: string;

begin
  s := '  123.5';
  { Ersetze Leerzeichen durch Nullen }
  while Pos(' ', s) > 0 do
    s[Pos(' ', s)] := '0';
end;
```

### Delete

Deklaration **procedure** Delete(var s: string; Index, Count: Integer);

Funktion: Löscht einen Teil eines Strings.

Erläuterung *s* ist eine String-Variable, *Index* und *Count* sind Integer-Ausdrücke. *Delete* löscht ab der durch *Index* angegebenen Position *Count*-Zeichen aus dem String *s*. Wenn *Index* größer als die Länge von *s* ist, passiert



nichts; wenn  $s$  kürzer oder genauso lang wie  $Index+Count$  ist, läßt *Delete* nur die vor der Position  $Index$  stehenden Zeichen übrig.

Beispiel

```
var s: string;

begin
  s := 'Honest Abe Lincoln';
  Delete(s, 8, 4);
  Writeln(s);           { 'Honest Lincoln' }
end;
```

### Insert

Deklaration

**procedure** Insert(Source: string; var s: string; Index: Integer);

Funktion:

Fügt einen String in einen anderen String ein.

Erläuterung

*Source* ist ein String-Ausdruck, *Index* ein Integer-Ausdruck. *s* ist eine String-Variable, in die *Insert* den Inhalt von *Source* ab der durch *Index* angegebenen Zeichenposition einfügt. *Insert* begrenzt die Länge von *s* automatisch auf 255, überzählige Zeichen werden (ohne Fehlermeldung) abgeschnitten.

Beispiel

```
var s: string;
begin
  s := 'Honest Lincoln';
  Insert('Abe ', s, 8);           { 'Honest Abe Lincoln' }
end;
```

Alle diese Prozeduren und Funktionen befinden sich in der Unit *System*, die stets mit Pascal geladen wird.

### Aufgaben

1. Geben Sie bei allen Prozeduren und Funktionen an, um welche Art der Parameterübergabe es sich bei den verschiedenen Parametern handelt. Geben Sie an, welche Funktion die Parameter haben (Eingangs-, Ausgangs- oder Durchgangsparameter) und begründen Sie jeweils, warum diese Art der Parameterübergabe gewählt wurde.
2. Definieren Sie eine Prozedur *DeleteExtension*, die in einem String nach einem eventuell vorkommenden Punkt den ganzen Rest des Strings, einschließlich des Punktes, löscht. Kommt kein Punkt vor, dann soll der String unverändert bleiben.

Beispiel für die Verwendung dieser Prozedur:

```
Var dateiname: string;

begin
  dateiname := 'Klassenarb.doc';
  DeleteExtension(dateiname);
  Writeln(dateiname);           { 'Klassenarb' }
end;
```

3. Definieren Sie eine Funktion *FDeleteExtension*, die die gleichen Aufgaben wie die oben definierte Prozedur erfüllen kann und schreiben Sie ein Beispielprogramm dazu.

## 6 Datentypen

Der Datentyp einer Variablen legt fest

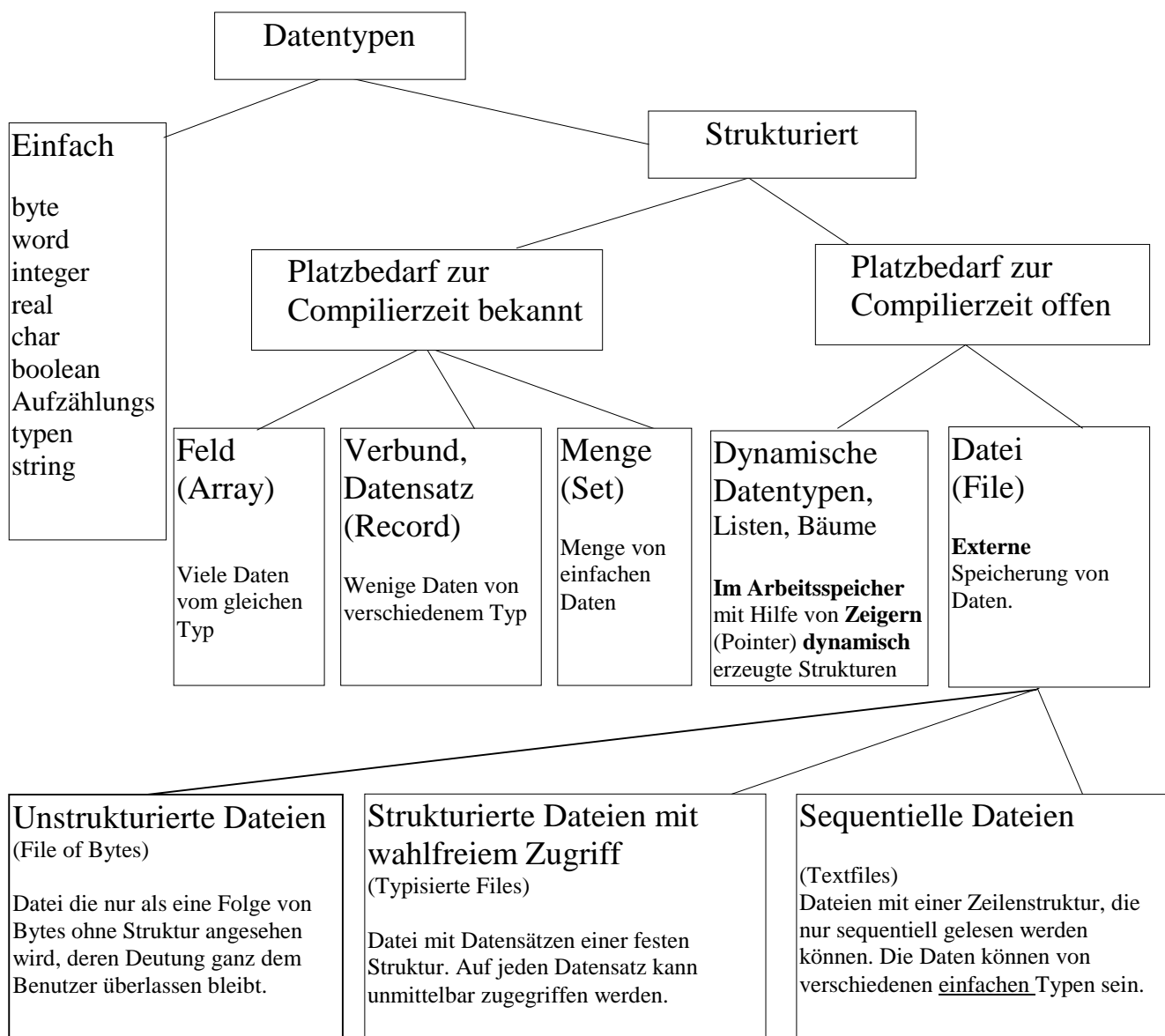
- den Wertebereich der Daten,
- die zugelassenen Operationen und Funktionen,
- den Speicherbedarf für die Daten.

Darum muß sich der Benutzer nicht unbedingt kümmern, wenn keine Optimierung des Speicherbedarfs nötig ist oder der Speicher zur Laufzeit explizit reserviert werden muß (bei dynamischen Variablen).

Man unterscheidet:

- einfache Datentypen,  
in PASCAL z.B. byte, word, integer, real, char, boolean, string,
- strukturierte (zusammengesetzte) Datentypen,  
in PASCAL z.B. array, record, set, file und dynamisch erzeugte Strukturen.

Die folgende Übersicht bezieht sich auf die Sprache Pascal. Man findet die genannten Datentypen in dieser oder ähnlicher Weise in allen imperativen Sprachen (C, Fortran, Ada,...). Nicht-imperative Sprachen benutzen eventuell andere oder zusätzliche weitere Datentypen ( LOGO und Prolog etwa den Datentyp Liste, vergl. das Beispiel zum Sortieren in 4.2.3).



In objektorientierten Sprachen kommen noch Objekte bzw. Klassen hinzu.

Objekte sind eine Verbindung der klassischen Datentypen mit Prozeduren zu einer Einheit. Sie können statisch oder dynamisch benutzt werden.

Die einfachen Datentypen wurden ausführlich in Kapitel 2.4 besprochen. Die Grundlagen für die beiden strukturierten Datentypen Array und Record wurden dort ebenfalls behandelt, daher sollen sie in diesem Kapitel nur nochmals kurz im Zusammenhang mit der Programmiersprache Pascal wiederholt werden. Der Datentyp Menge spielt eine untergeordnete Rolle und wird nicht weiter erläutert.

### Unterscheidung zwischen statischen und dynamischen Datentypen

Bislang sind wir beim Speichern von Daten davon ausgegangen, dass der Umfang der zu speichernden Daten zur Zeit der Programmerstellung und damit *vor* dem Übersetzen (Compilieren) des Programms bekannt ist. Daher konnte bei allen bisherigen Programmbeispielen der Speicherplatz für die zu speichernden Daten (d.h. der Speicherplatz für die benutzten Variablen) vom Compiler **zum Compilierzeitpunkt reserviert** werden. Während eines Programmablaufs stand dann nur dieser Speicherplatz zur Verfügung und konnte nicht vergrößert werden. Man spricht in diesem Zusammenhang von **statischen Variablen**.

Häufig kann man jedoch nicht vorhersehen, wieviel Speicherplatz für die Daten in einem Programm benötigt wird. Dies kann sich erst zum Zeitpunkt des Programmablaufs entscheiden, also zu einem Zeitpunkt, zu dem das Programm übersetzt ist und der Compiler nicht mehr zur Verfügung steht. Ein solcher Fall tritt auf, wenn etwa Bücher für eine Bibliothek aufgenommen werden sollen. In diesem Fall verwendet man für die Daten die **externe Speicherung** in einer **Datei**. Die Größe einer Datei muß zum Compilierzeitpunkt nicht bekannt sein, sie kann während eines Programmablaufs immer weiter vergrößert werden und ist nur durch die Kapazität des externen Speichermediums (Platte, Diskette, CD) beschränkt.

Ein anderer Fall liegt vor, wenn die zu bearbeitenden Daten nicht extern gespeichert werden sollen, sondern **im Rechner** zur Verfügung stehen müssen, ihr Umfang aber vor Beginn eines Programmablaufs sinnvollerweise nicht festgelegt werden kann. Ein typisches Beispiel für diesen Fall ist ein Programm, in dem ein Benutzer nach Belieben Fenster öffnen und in diese Fenster Texte, Bilder, Tabellen usw. einfügen kann (z.B. ein Textverarbeitungsprogramm oder die gesamte Windows-Oberfläche). Der Autor des Programms legt sinnvollerweise nicht Platz für 20 Textfenster mit Text bis zu 1 000 000 Zeichen pro Fenster fest, sonst würde ein Benutzer scheitern, der mit dem Programm nur ein Fenster mit einem Text mit 2 000 000 Zeichen öffnen möchte. Die Lösung des Problems liegt in der Verwendung **dynamischer Variablen**: Der **Speicherplatz** für dynamische Variablen **wird erst zum Zeitpunkt des Programmablaufs angefordert und reserviert**, wenn dieser Platz auch tatsächlich gebraucht wird. Der Speicherplatz für alle dynamischen Variablen zusammen ist damit nur beschränkt durch den Speicherplatz, der dem gesamten Programm zur Verfügung gestellt wird. Benötigt eine Variable weniger Platz, so steht für die übrigen mehr zur Verfügung.

Die Verwaltung des dynamischen Speicherplatzes obliegt dem Betriebssystem. Kommt es dabei, etwa wenn mehrere Programme gleichzeitig ablaufen und der Verwalter des Speicherplatzes einen Fehler begeht, zu Überschneidungen, dann kann sich ein Programm schon einmal mit einer Meldung „Allgemeine Schutzverletzung“ verabschieden.

## 6.1 Datentyp Array

### Definition:

Ein Array (Feld) ist eine Zusammenfassung einer **festen Anzahl gleichartiger Daten**, den Komponenten, zu **einer** neuen **zusammengesetzten Gesamtheit**. Die Komponenten lassen sich über einen gemeinsamen Bezeichner für das Array und einen Index ansprechen.

### 6.1.1 Eindimensionale Arrays (Pascal)

**Deklaration** eines eindimensionalen Array-Typs in Pascal:

```
array [ Indextyp ] of Komponententyp
```

Als Komponententyp ist (fast) jeder Typ zugelassen.

Als Indextyp sind nur Teilbereiche der sogenannten ordinalen Datentypen byte, word, integer, char zugelassen (ebenso selbstdefinierte ordinale Datentypen wie Montag, Dienstag, ....).

**Zugriff** auf die Komponenten:

```
array-Variable [ Index ]
```

Der *Index* darf dabei ein beliebiger Ausdruck vom Indextyp des Arrays sein.

### Beispiel:

```
var
  a: array [1..70] of real;
  x: real;
  u: integer;

for u:=1 to 35 do begin  x:= a[2*u-1];  writeln(x) end;
```

Bemerkung: Ist  $a$  eine *Variable* vom Arraytyp und  $t$  ein beliebiger Ausdruck des geeigneten Indextyps, dann ist  $a[t]$  eine *Variable* vom entsprechenden Komponententyp und kann als aktueller Parameter für Referenzparameter eingesetzt werden.

### 6.1.2 Mehrdimensionale Arrays (Pascal)

Im Grunde benötigt man in Pascal keine mehrdimensionalen Arrays: ein zweidimensionales Array ist nichts als ein eindimensionales Array von eindimensionalen Arrays; ein dreidimensionales Array ist nichts als ein eindimensionales Array von zweidimensionalen Arrays usw. Diese Auffassung hat den Vorteil, dass die einzelnen Teilfelder als Ganzes angesprochen werden können.

Statt allgemeiner Beschreibungen mag ein **Beispiel** dienen:

Das folgende Feld soll erfasst werden:

7	4	6	5	←	Zeile
2	5	12	33		
13	7	4	9		
	↑				
	Spalte				

Dieses Feld kann als eindimensionales Feld der Zeilen aufgefaßt werden:

```
var
a: array [1..3] of array [1..4] of integer;
```

Dann bedeuten

$a[1]$  die gesamte 1. Zeile (7, 4, 6, 5), also ein eindimensionales Array.  
 $a[1][3]$  die 3. Komponente der ersten Zeile, also die Zahl 6.  
 $a[3]$  die gesamte 3. Zeile (13, 7, 4, 9), also ein eindimensionales Array.  
 $a[3][2]$  die 2. Komponente der dritten Zeile, also die Zahl 7.

Kurzschreibweisen:

```
var
a: array [1..3, 1..4] of integer;
```

$a[1]$  die gesamte 1. Zeile, wie oben,  
 $a[1, 3]$  die 3. Komponente der ersten Zeile, also die Zahl 6

Zu Anwendungsbeispielen mit eindimensionalen Arrays siehe Kapitel 4.2.

## 6.2 Datentyp Record

### Definition:

Ein Record (Datensatz) ist eine Zusammenfassung einer *festen Anzahl* meist *verschiedenartiger Daten*, den Feldern, zu *einer* neuen *zusammengesetzten Gesamtheit*. Die Komponenten lassen sich über einen gemeinsamen Bezeichner für den Record und einen Feldbezeichner ansprechen.

**Deklaration** eines Record-Typs in Pascal:

```
record
  feldname1, feldname2 : feldtyp1;
  feldname3, ..... : feldtyp2;
  .
  .
end;
```

Als Feldtypen sind (fast) alle Typen zugelassen.

**Zugriff** auf die Felder eines Records:

*record-Variable.feldname*

Man ist geneigt, Records als Verallgemeinerung von Arrays anzusehen, da hier auch viele Daten zu einer Gesamtheit zusammengefasst und als Ganzes angesprochen werden können. Der Zugriff auf die verschiedenartigen Felder eines Records erfolgt aber in grundsätzlich anderer Weise als auf die gleichartigen Komponenten eines Arrays. Während man mit Hilfe von Variablen für die Indizes eines Arrays in Schleifen alle Komponenten des Arrays bearbeiten kann, sind die Feldnamen eines Records feste Bezeichner, für die keine Variablen zur Verfügung stehen, man kann also insbesondere keine Schleifen über die Feldnamen laufen lassen. Aus diesen Gründen haben Arrays oft eine sehr große Anzahl von Komponenten ( Tausende ), Records haben aber typischerweise eher wenige Felder ( 2 bis 10, manchmal mehr).

### Beispiel:

```
type
  TPerson = record
    vorname, name :string[12];
    alter : integer;
  end;
var
  schueler : TPerson;

  schueler.vorname := 'Hans-Peter';
  schueler.name := 'Becker';
  schueler.alter := 15;

  writeln( schueler.vorname + ' ' +schueler.name);
```

Der Zugriff auf die Felder eines Records kann in Pascal etwas übersichtlicher und kürzer gestaltet werden, wenn man die `with`-Anweisung benutzt, um den Bezeichner des Records nicht immer wiederholen zu müssen. Im Beispiel oben:

```
with schueler do
begin
  vorname := 'Hans-Peter';
  name := 'Becker';
  alter := 15;

  writeln( vorname + ' ' +name);
end;
```

Records spielen eine sehr große Rolle im Zusammenhang mit der Verarbeitung und Speicherung von Datensätzen mit Hilfe von Dateien (vergl. Kapitel 6.3) und für den Aufbau von Listen und Bäumen und für die objektorientierte Programmierung, da Objekte als Records mit Feldern für Prozeduren aufgefaßt werden können.

Zu weiteren Anwendungsbeispielen mit Records siehe Kapitel 6.3.1, Arbeit mit strukturierten Dateien.

## 6.3 Dateien

In Dateien werden Daten außerhalb des Arbeitsspeichers eines Rechners gespeichert. Die Speichermedien können Festplatten, Wechselplatten, Disketten, Magnetbänder, CDs oder andere Medien sein.

Man unterscheidet im wesentlichen zwei Zugriffsarten auf Speichermedien:

- |   |  |
|---|--|
| <b>Wahlfreier Zugriff</b> (Random Access)       | auf jede kleinste Dateneinheit (Byte) kann direkt über eine berechenbare Adresse zugegriffen werden. |
| <b>Sequentieller Zugriff</b> (Sequential Acces) | auf jede Dateneinheit kann nur durch Lesen der vorangehenden Dateneinheiten zugegriffen werden       |

Man kann die Zugriffsarten auf die externen Speichermedien unter zweierlei Aspekten betrachten:

- physikalisch
- logisch.



### 6.3.1 Beispielprogramme für einfachste Dateiverarbeitung

---

```

program Text_Datei1;           {Text-Datei schreiben}
uses crt;
var
  f:text;                       {Textfile-Variable zum Puffern}

begin
  clrscr;
  assign(f,'Testxt.txt');        {interne Puffer-Variable f mit Dateinamen verbinden }
  rewrite(f);                    {Textdatei zum neu Schreiben öffnen}

  writeln(f,'Das ist eine Zeile Text. ');
  writeln(f,'Das ist die zweite Zeile. ');
  writeln(f,'Jetzt kommt eine Real-Zahl ',3.1415);
  writeln(f,'Letzte Zeile!!!');

  close(f);                      {Datei schließen.}
end.

```

---

```

program Text_Datei2;           {Text-Datei lesen}
uses crt;

var
  f:text;                       {Textfile-Variable zum Puffern}
  st:string;                     {Variable, in die jeweils eine Zeile eingelesen wird}

begin
  clrscr;
  assign(f,'Testxt.txt');        {interne Puffer-Variable f mit Dateinamen verbinden }
  reset(f);                      {Textdatei zum Lesen öffnen}

  while not eof(f) do           {solange Dateiende nicht erreicht...}
  begin
    readln(f,st);                {eine Zeile aus der Datei lesen... }
    writeln(st);                  {und auf dem Bildschirm ausgeben. }
  end;

  close(f);                      {Datei schließen.}
end.

```

---

```

program Strukturierte_Datei1; {Feste strukturierte Datei schreiben }
uses crt;

type
  TPerson=record                 {neuer Record-Typ für Personen}
    vorname,name:string[12];
    alter:integer
  end;

var
  f:file of TPerson;             { Variable für strukturierte Datei}
  person:TPerson;                { Variable für eine Person }

begin
  clrscr;
  assign(f,'Person.dat');         {f mit Dateinamen verbinden }
  reset(f);                       {Datei zum Lesen öffnen}
  rewrite(f);                      {Datei zum neu Schreiben öffnen}

  person.vorname:='Hanna';
  person.name:='Blumentritt';

```

```
person.alter:=33;
write(f,person);

person.vorname:='Erwin';
person.name:='Wild';
person.alter:=24;
write(f,person);

person.vorname:='Marie-Louise';
person.name:='Leutheusser-Schnarrenberger';
person.alter:=47;
write(f,person);

person.vorname:='Eva';
person.name:='Kaupp';
person.alter:=22;
write(f,person);

close(f);           {Datei schließen.}
end.
```

---

```
program Strukturierte_Datei2; {Feste strukturierte Datei lesen}
uses crt;

type
  TPerson=record           {neuer Record-Typ für Personen}
    vorname,name:string[12];
    alter:integer
  end;
var
  f:file of TPerson;      { Variable für strukturierte Datei}
  person:TPerson;        { Variable für eine Person      }

begin
  clrscr;
  assign(f,'Person.dat'); {interne Puffer-Variable f mit Dateinamen verbinden }
  reset(f);              {Datei zum Lesen Öffnen}

  while not eof(f) do    {solange Dateiende nicht erreicht...}
  begin
    read(f,person);      {lies einen ganzen Datensatz aus der Datei...}
    with Person do       {und gib die Felder des Datensatzes aus}
    begin
      writeln(vorname,' ',name);
      writeln('Alter: ',alter);
      writeln
    end
  end;

  close(f);              {Datei schließen.}
end.
```

---

```
program Strukturierte_Datei3; {In fester strukturierter Datei Datensatz suchen}
uses crt;

type
  TPerson=record           {neuer Record-Typ für Personen}
    vorname,name:string[12];
    alter:integer
  end;
var
  f:file of TPerson;      { Variable für strukturierte Datei}
```



```

    person:TPerson;           { Variable für eine Person           }

begin
  clrscr;
  assign(f,'Person.dat');
  reset(f);

  seek(f,2);                 {suche den 2. Datensatz (der erste hat Nr.0)...}
  read(f,person);           {und lies den ganzen Datensatz aus der Datei...}
  with Person do            {und gib die Felder des Datensatzes aus.}
    begin
      writeln(vorname, ' ',name);
      writeln('Alter: ',alter);
      writeln
    end;

  close(f);
end.

```

---

```

program Strukturierte_Datei4; {Strukturierte Datei mit wählbarem Dateinamen schreiben}
uses crt;

```

```

type
  TPerson=record
    vorname,name:string[12];
    alter:integer
  end;

var
  f:file of TPerson;        { Variable für strukturierte Datei}
  person:TPerson;          { Variable für eine Person           }
  filename:string[12];     {Name für die Datei}
  ch:char;

begin
  clrscr;
  write('Dateiname: ');readln(filename);
  assign(f,filename);      {Variable f mit Dateinamen verbinden }
  rewrite(f);              {Datei zum neu Schreiben öffnen}

  writeln;

  repeat
    writeln;
    write('Vorname : ');readln(person.vorname);
    write('Nachname: ');readln(person.name);
    write('Alter   : ');readln(person.alter);
    write(f,person);
    writeln;
    write('Noch ein Eintrag? (J)');ch:=readkey;
  until not (ch in ['J','j']);

  close(f);
end.

```

```
program Strukturierte_Datei5; {Strukturierte Datei mit wählbarem Dateinamen lesen}
uses crt;

type
  TPerson=record
    vorname,name:string[12];
    alter:integer
  end;
var
  f:file of TPerson;           { Variable für strukturierte Datei}
  person:TPerson;             { Variable für eine Person      }
  filename:string[12];        { Name für die Datei}

begin
  clrscr;
  write('Dateiname: ');readln(filename);
  assign(f,filename);
  reset (f);

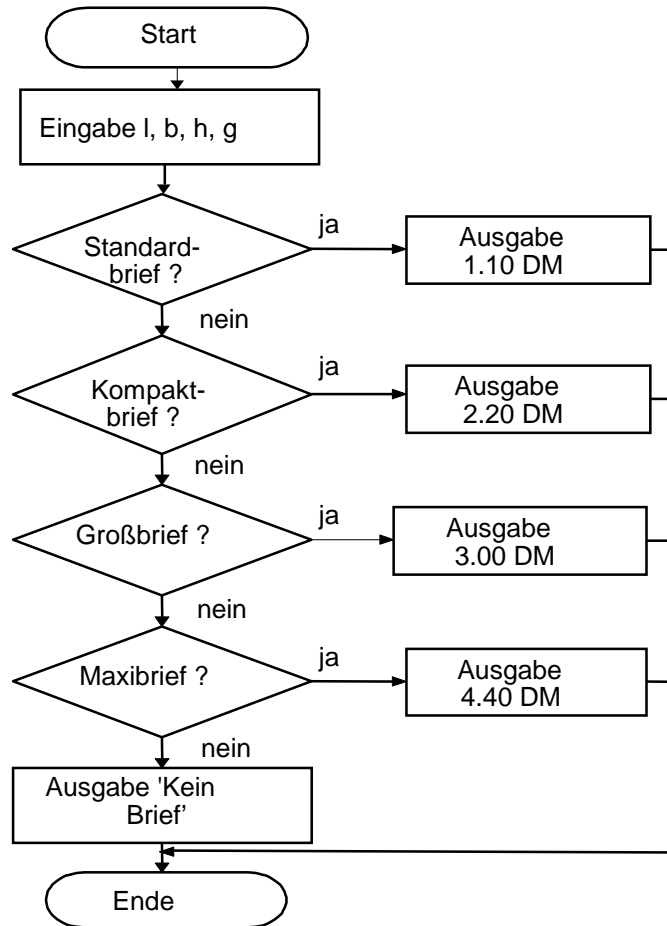
  writeln;

  while not eof(f) do          {solange Dateiende nicht erreicht...}
  begin
    read(f,person);           {lies einen ganzen Datensatz aus der Datei...}
    with Person do            {und gib die Felder des  Datensatzes aus}
    begin
      writeln(vorname,' ',name);
      writeln('Alter: ',alter);
      writeln
    end
  end;

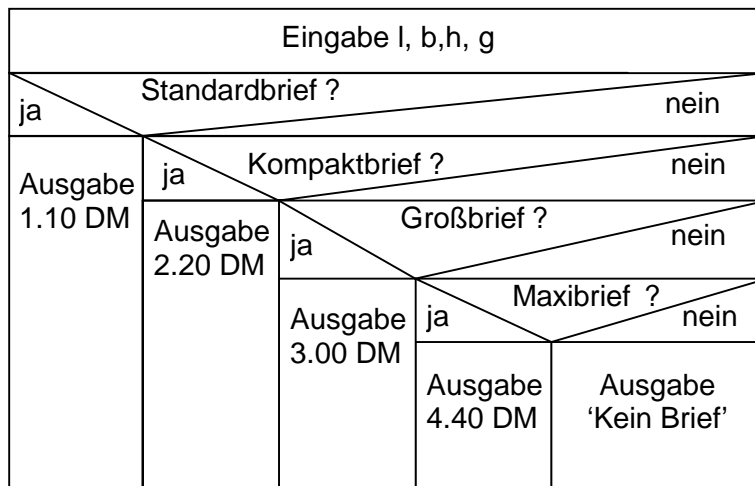
  close(f);
end.
```

## 7 Zusatz: Postgebühren für Briefe (von TURBO-PASCAL zu DELPHI)

### Programmablaufplan



### Struktogramm



<b>Standardbrief:</b> $(l \leq 235) \text{ AND } (l \geq 140) \text{ AND } (b \leq 125) \text{ AND } (b \geq 90) \text{ AND } (h \leq 5) \text{ AND } (l \geq 1.41 * b) \text{ AND } (g \leq 20)$	<b>Kompaktbrief:</b> $(l \leq 235) \text{ AND } (l \geq 100) \text{ AND } (b \leq 125) \text{ AND } (b \geq 70) \text{ AND } (h \leq 10) \text{ AND } (l \geq 1.41 * b) \text{ AND } (g \leq 50)$
<b>Großbrief:</b> $(l \leq 353) \text{ AND } (l \geq 100) \text{ AND } (b \leq 250) \text{ AND } (b \geq 70) \text{ AND } (h \leq 20) \text{ AND } (g \leq 500)$	<b>Maxibrief:</b> $(l \leq 353) \text{ AND } (l \geq 100) \text{ AND } (b \leq 250) \text{ AND } (b \geq 70) \text{ AND } (h \leq 50) \text{ AND } (g \leq 1000)$

## Das Programm in Turbo-Pascal:

```
program post;
uses crt;

var
  l,b,h,g:real;           {Daten für den Brief}

begin
  clrscr;                 {Bildschirm löschen}
  write('Länge   l in mm: '); readln(l);  {Eingabe l, b, h, g}
  write('Breite  b in mm: '); readln(b);
  write('Höhe   h in mm: '); readln(h);
  write('Gewicht g in mm: '); readln(g);

  if (l<=235) AND (l>=140) AND (b<=125) AND (b>=90) AND
     (h<=5) AND (l>=1.41*b) AND (g<=20)
  then writeln('Standardbrief, Porto 1,10 DM')
  else if (l<=235) AND (l>=100) AND (b<=125) AND (b>=70) AND
     (h<=10) AND (l>=1.41*b) AND (g<=50)
  then writeln('Kompaktbrief, Porto 2,20 DM')
  else if (l<=353) AND (l>=100) AND (b<=250) AND (b>=70) AND
     (h<=20) AND (g<=500)
  then writeln('Großbrief, Porto 3,00 DM')
  else if (l<=353) AND (l>=100) AND (b<=250) AND (b>=70) AND
     (h<=50) AND (g<=1000)
  then writeln('Maxibrief, Porto 4,40 DM')
  else writeln('Kein Brief, bitte am Schalter nachfragen');

  readln;
end.
```

Dies ist die einfachste Version eines möglichen Programms. Es muss für jede Berechnung neu gestartet werden. Die nächste Verbesserung bestünde in der Wiederholung der gesamten Eingabe und Berechnung bis ein Programmende gewünscht wird. Bei Programmierung in Delphi entfällt diese Notwendigkeit aber ganz, da dort die Programmsteuerung nicht mehr explizit von einer Prozedur vorgenommen wird, sondern implizit (gewissermaßen „automatisch“) durch die vom Betriebssystem (Windows) zur Verfügung gestellte Steuerung durch Ereignisse (Mausklicks auf Schaltflächen, Tastatureingaben, usw.).

Eine klarere Struktur des Programms ergibt sich, wenn die Berechnung des Portos in einer Funktion zusammengefasst wird, während Ein- und Ausgabe davon getrennt erfolgen. Das so modifizierte Programm lässt sich durch die Trennung der Benutzerschnittstelle von der eigentlichen Verarbeitung der Daten sehr leicht in *Delphi* übertragen, da die Funktion einfach übernommen werden kann. Die Eingabe der Daten erfolgt in einer Prozedur `eingabe`, während die Berechnung der Postgebühren (bzw. gleich des Ausgabestrings für die berechnete Gebühr) durch die Funktion `gebuehr` erfolgt. Noch klarer wäre die noch weiter gehende Trennung der Berechnung der Gebühr von der eigentlichen Ausgabe eines Textes.

```
program post_Funktion;
uses crt;

var
  lg,bg,hg,gg:real;           {Daten für den Brief}

  ch:char;

procedure eingabe(var l,b,h,g:real);
begin
  write('Länge l in mm: '); readln(l);
  write('Breite b in mm: '); readln(b);
  write('Höhe h in mm: '); readln(h);
  write('Gewicht g in mm: '); readln(g);
end;

function gebuehr(l,b,h,g:real):string;
begin
  if (l<=235) AND (l>=140) AND (b<=125) AND (b>=90) AND (h<=5) AND (l>=1.41*b) AND
    (g<=20)
  then gebuehr:='Standardbrief, Porto 1,10 DM'
  else if (l<=235) AND (l>=100) AND (b<=125) AND (b>=70) AND (h<=10) AND (l>=1.41*b)
AND
    (g<=50)
  then gebuehr:='Kompaktbrief, Porto 2,20 DM'
  else if (l<=353) AND (l>=100) AND (b<=250) AND (b>=70) AND (h<=20) AND
(g<=500)
  then gebuehr:='Großbrief, Porto 3,00DM'
  else if (l<=353) AND (l>=100) AND (b<=250) AND (b>=70) AND (h<=50) AND
    (g<=1000)
  then gebuehr:='Maxibrief, Porto 4,40DM'
  else gebuehr:='Kein Brief, am Schalter fragen';
end;

begin
  clrscr;
  eingabe(lg,bg,hg,gg);
  writeln(gebuehr(lg,bg,hg,gg));

  ch:=readkey;
end.
```

## Das Programm in Delphi

Formular:

Das Formular für die Benutzerschnittstelle wird in der graphischen Entwicklungsumgebung von Delphi erzeugt (Visual Programming). Gleichzeitig erstellt das System automatisch den zugehörigen Code.

Die Arbeit in der graphischen Entwicklungsumgebung entspricht weitgehend der Arbeit mit einem üblichen Graphiksystem.

Die Wiederholung der Berechnungen muss hier nicht durch eine Schleifenkonstruktion programmiert

werden, sondern wird durch die Ereignissteuerung von Delphi-Programmen in Windows überflüssig. Das Berechnungsformular bleibt bis zum Programmende ständig verfügbar.

### Programmcode:

```
unit Postgeb;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TFormular = class(TForm)
    Titel: TLabel;
    Eingabefeld_Laenge: TEdit;
    Label_Laenge: TLabel;
    mm1: TLabel;
    Label_Breite: TLabel;
    Eingabefeld_Breite: TEdit;
    mm2: TLabel;
    Label_Hoehe: TLabel;
    Eingabefeld_Hoehe: TEdit;
    mm3: TLabel;
    Label_Gewicht: TLabel;
    Eingabefeld_Gewicht: TEdit;
    g: TLabel;
    Ausgabe: TEdit;
    Button_Berechnung: TButton;
    Button_Ende: TButton;
    procedure Button_BerechnungClick(Sender: TObject);
    procedure Button_EndeClick(Sender: TObject);
  private
    function gebuehr(l,b,h,g:real):string;
  end;
```

Dieser Code wird **automatisch erzeugt**, wenn beim Aufbau des Formulars in der graphischen Entwicklungsumgebung die Komponenten aus der Werkzeugleiste gezogen und mit Eigenschaften versehen werden.

```

var
  Formular: TFormular;

implementation

{$R *.DFM}

function TFormular.gebuehr(l,b,h,g:real):string;{exakt aus Pascal übernommen}
begin
  if (l<=235) AND (l>=140) AND (b<=125) AND (b>=90) AND
    (h<=5) AND (l>=1.41*b) AND (g<=20)
  then gebuehr:='Standardbrief, Porto 1,10 DM'
  else if (l<=235) AND (l>=100) AND (b<=125) AND (b>=70) AND
    (h<=10) AND (l>=1.41*b) AND (g<=50)
  then gebuehr:='Kompaktbrief, Porto 2,20 DM'
  else if (l<=353) AND (l>=100) AND (b<=250) AND (b>=70) AND
    (h<=20) AND (g<=500)
  then gebuehr:='Großbrief, Porto 3,00 DM'
  else if (l<=353) AND (l>=100) AND (b<=250) AND (b>=70) AND
    (h<=50) AND (g<=1000)
  then gebuehr:='Maxibrief, Porto 4,40 DM'
  else gebuehr:='Kein Brief, bitte am Schalter nachfragen';
end;

procedure TFormular.Button_BerechnungClick(Sender: TObject);
var
  l,b,h,g:real;           {Daten für den Brief}
begin
  l:=StrToFloat(Eingabefeld_Laenge.Text);
  b:=StrToFloat(Eingabefeld_Breite.Text);
  h:=StrToFloat(Eingabefeld_Hoehe.Text);
  g:=StrToFloat(Eingabefeld_Gewicht.Text);
  Ausgabe.Text:=gebuehr(l,b,h,g);
end;

procedure TFormular.Button_EndeClick(Sender: TObject);
begin
  halt;
end;

end.   {Ende der unit Postgeb}

program Post;

uses
  Forms,
  Postgeb in 'Postgeb.pas' {Formular};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TFormular, Formular);
  Application.Run;
end.

```

Dieser Code wird **automatisch erzeugt**, wenn in der graphischen Entwicklungsumgebung ein neues Projekt geöffnet wird.

## 8 Literatur

### 8.1 Zur Einführung in die Informatik zu empfehlen

**GÜNTHER, K.**, Informatik I

Skriptum zur Vorlesung im WS 89/90, PH Freiburg.

Sehr gute kurze Zusammenfassung der grundlegenden Begriffe der Informatik. Bestens geeignet als Leitfaden zu vertieftem Studium einzelner Teilgebiete an Hand weiterführender Literatur.

**BONG, U.**, Informatik II

Skriptum zur Vorlesung im WS 89/90, PH Freiburg

Sehr gute kurze Zusammenfassung. Bestens geeignet als Leitfaden zu vertieftem Studium einzelner Teilgebiete an Hand weiterführender Literatur.

**GOLDSCHLAGER/LISTER**, Informatik

3.Auflage, Hanser 1990

Relativ neu, gut lesbar, kann mit einigen Auslassungen gut neben der Vorlesung benutzt werden

**RECHENBERG, P.**, Was ist Informatik

2.Aufl., Hanser, München, 1994,

Sehr gut lesbar, im Aufbau aber nicht ganz der Vorlesung entsprechend, da zuerst mit der technischen Informatik begonnen wird.

⇒ Gutes Begriffswörterbuch im Anhang.

⇒ Sehr lesenswerte kurze Kapitel über "Philosophie der Informatik".

**PRECHT, M., MEIER, N. KLEINLEIN, J.**, EDV Grundwissen

3.Auflage, Addison-Wesley, Bonn 1996

Sehr **aktuelle**, aber etwas kurze Beschreibung von gegenwärtig verfügbaren Geräten, Anwenderprogrammen, Betriebssystemen und gängigen Theorien. Eignet sich sehr gut zum Nachschlagen, weniger als Text für die Informatik. Gute, klare Abbildungen.

**GUMM, H.-P., SOMMER, M.**, Einführung in die Informatik

2.Auflage, Addison-Wesley, Bonn 1995

Sehr aktuelle, gut lesbare Einführung in die Informatik, die im wesentlichen alle neueren Entwicklungen berücksichtigt (Objektorientierte Programmierung, aktuelle Betriebssysteme, Netze). Nicht sehr theoretisch. Folgt im Aufbau im wesentlichen der Vorlesung.

**WHITE, RON**, So funktionieren Computer. Ein universeller Streifzug durch den Computer

Markt und Technik, Haar bei München 1996

Kein Informatikbuch, aber sehr detaillierte Beschreibungen und Illustrationen zum Aufbau und zur Wirkungsweise von Computern sowie der Peripheriegeräte und Kommunikationseinrichtungen (Netzwerke, Modems,...).

### Weitere Literatur zur Informatik: Ergänzend oder interessante Aspekte

**GOOS, G.** Vorlesungen über Informatik

Grundlegende Vorlesung zur Informatik für Informatiker. Eher zum Nachlesen als zum Studium der Vorlesung an der PH geeignet.

Band 1

Grundlagen und funktionales Programmieren : mit 35 Tab. - 1995. - XII, 393 S. : graph. Darst.; (dt.)

Band 2

Objektorientiertes Programmieren und Algorithmen : mit 12 Tabellen. - 1996. - XII, 360 S. : graph. Darst.; (dt.)

Springer, Berlin 2000



**BAUMANN, R.**, Informatik für die Sekundarstufe II, 2 Bände

Klett, Stuttgart, Band 1 1992, Band 2 1993

Band 1: Einfach lesbare Einführung in die Informatik für die Schule (Gymnasium) mit vielen Beispielen in Pascal. Auch als Einführung in die Pascal-Programmierung geeignet.

Band 2: Weiterführende Gebiete werden behandelt, z.B. Höhere Datenstrukturen, abstrakte Datentypen, Automatentheorie und formale Sprachen, Grenzen der Berechenbarkeit, Komplexitätstheorie. Dieser Band führt in sehr verständlicher Form in die o.g. Gebiete ein, ohne zu formal vorzugehen. Auch zum Nachlesen einzelner Kapitel geeignet, wenn einige Grundkenntnisse vorhanden sind.

**WIRTH, N.**, Algorithmen und Datenstrukturen

Teubner, Stuttgart 1983

Hervorragende klassische Einführung in das genannte Gebiet vom Vater der Sprache PASCAL, auch Jahre nach dem Erscheinen noch aktuell. Enthält z.B. sehr gute Darstellung und Analyse von Sortierverfahren. Nicht ganz einfach zu lesen.

**WARNKE, M.**, Informatik (für das Nebenfachstudium)

Oldenbourg Verlag, München 1989

Klassische Einführung mit Pascal, nicht zu abstrakt.

Scheint mir von allen für das **Hochschulstudium** gedachten Büchern am besten geeignet.

**KURZWEIL, R.**, Das Zeitalter der Künstlichen Intelligenz

Hanser, München. 1992

Schöne popularwissenschaftliche Darstellung der Geschichte der Computerentwicklung mit Schwerpunkt auf der KI. Viele Bilder aus der Geschichte der EDV. Einige grundlegende Originalbeiträge. Sehr gut zu lesen, auch auszugsweise, interessant und anregend. Steht allerdings ganz in der Tradition der amerikanischen optimistisch-technikgläubigen Tradition der KI-Päpste vom MIT.

**WEIZENBAUM, JOSEPH**, Die Macht der Computer und die Ohnmacht der Vernunft

Suhrkamp 1977

Eines der wenigen, von Informatikern verfassten Computer-kritischen Bücher.

Stark von der Situation während des kalten Krieges geprägt. Enthält außer der Kritik auch eine Darstellung der Grundlagen der Informatik (z.B. Turingmaschinen, Grenzen der Berechenbarkeit) und ist daher nicht so einfach zu lesen.

## 8.2 Vorlesungen zur Informatik im Internet

**PROF. DR.U.SCHROEDER**

Vorlesung von Prof. Dr.U.Schroeder an der PH-Ludwigsburg

Sehr gute Einführung in die Informatik für Studierende an einer Pädagogischen Hochschule. Sehr starke Betonung übergreifender Aspekte, gesellschaftliche Auswirkungen, Bezug zur Schule.

<http://www.ph-ludwigsburg.de/mathematik/lehre/inf1/folien/index.html>

**PROF. DR.U.SCHROEDER**

Skript Einführung in die Informatik Schroeder an der TU Darmstadt

Vorlesung zur Einführung für Informatik-Studierende

<http://www.pi.informatik.tu-darmstadt.de/inf1/Skript/einf/index.html>

**PROF. DR. DILGER**

Vorlesungsskripte zur künstlichen Intelligenz von der TU-Chemnitz

Guter Überblick über die Anwendungen der Künstlichen Intelligenz.

<http://www.tu-chemnitz.de/informatik/HomePages/KI/skripte.php?druck>

**BICS - INFORMATIK SEKUNDARSTUFE II**

Sammlung von Materialien zur Behandlung der Programmiersprache Delphi im Unterricht

<http://www.be.schule.de/bics/inf2/programmiersprachen/delphi/index.html>

Linkliste zum Thema "Kann eine Maschine denken?"

<http://ods.schule.de/bics/inf2/gesellschaft/>

### 8.3 Zur Didaktik der Informatik

**BAUMANN, R.**, Didaktik der Informatik

Klett, Stuttgart 1997

Umfangreiche Sammlung von grundlegenden Konzepten der Informatik und ihren Randgebieten, einschließlich der unterrichtlichen Umsetzung bis hin zu Projektvorschlägen und Kriterien für die Leistungsbeurteilung.

**GIERTH, U.**, Datenschutz im Unterricht

Dümmler 1988

**HUBWIESER, PETER**, Didaktik der Informatik

Springer, Berlin 2000

Gute aktuelle Didaktik der Informatik mit praktischen Beispielen zum Einsatz im Unterricht. Versucht wie alle Didaktiken der Informatik die Frage zu beantworten: Was sind grundlegende und überdauernde Konzepte der Informatik und was ist eher nur augenblicklich aktuelles und nützliches Wissen?

**MODROW, E.**, Zur Didaktik des Informatikunterrichts

Band 1 und 2, Dümmler 1991

Band 2: Gesellschaftliche Auswirkungen )

**CYRANEK, G., FORNECK, H., GOORHUIS, H. (HRSG.)**, Beiträge zur Didaktik der Informatik.

Frankfurt, Diesterweg 1990

Ganz gute Beiträge

**BICS - BERATUNGSSTELLE FÜR INFORMATIONSTECHNISCHE GRUNDBILDUNG UND COMPUTEREINSATZ IN SCHULEN**

Materialien zum Informatikunterricht. Beispiele für Projekte, Verbindung zu freier Software usw.

<http://ods.schule.de/bics/>

### 8.4 Gesellschaftliche Auswirkungen, kritische Auseinandersetzung mit der Informatik

**COY, W.**, Sichtweisen der Informatik

Vieweg, Braunschweig 1992

Behandelt auch Gefahren und Sozialverträglichkeit der Informatik

**GMEHLICH, R., RUST, H.**, Mehr als nur Programmieren

Vieweg, Braunschweig 1993

Einführung in die Informatik mit Betrachtung kritischer Situationen.

**LANGENHEDER, W., MÜLLER, G. (HRSG.)**, Informatik-cui bono

Springer Berlin 1992

Tagungsband einer Tagung am Institut für Informatik und Gesellschaft in Freiburg 1992. Beiträge verschiedener Autoren zur Auswirkung der Informatik in der Gesellschaft.

**WEIZENBAUM, J.**, Kurs auf den Eisberg

Pendo Verlag, Zürich 1984

Interview mit J.Weizenbaum über die Auswirkungen der Informationstechnologie. Kurz und einfach zu lesen.

**FRIEDRICH, J., HERRMANN, TH., PESCHEK, M., ROLF, A. (Hrsg.)**, Informatik und Gesellschaft

Spektrum Akad. Verlag, Heidelberg 1995

Behandelt die Auswirkungen, Gefahren und Chancen der Informatik in Bezug auf die Gesellschaft und Arbeitswelt, Themen, über die in den klassischen Informatikbüchern nur im Vorspann oder im Anhang einige Seiten zu finden sind.

## 8.5 Freeware und Shareware-Programme zur Vorlesung

### SERVANT SALAMANDER

Programm (englisch) entspricht dem „Norton Commander“. Organisation von Dateien und Verzeichnissen (Erstellen, Kopieren, Verschieben usw.) auch auf dem Netzwerk.

Betrachten und Editieren von Dateien in Hexadezimaldarstellung.

© 2001 ALTAP, Ltd., Quelle: <http://www.altap.cz/>

### WINHEX

Sehr schönes Programm, um einmal die Festplatte byteweise anzusehen und auch die logische Deutung der Einträge zu erkunden.

Quelle: <http://www.winhex.com/winhex/index-m.html>