Einführung in die Programmierung

Christian Zahler

Unter "Programmieren" versteht man die Erstellung von Software.

Was ist eigentlich ein Programm?

Ein Programm ist eine Folge von Befehlen an die CPU (central processing unit, zu deutsch Zentralprozessor, z.B. der intel 80486), d.h.

- es gibt einen 1. Befehl
- es gibt einen letzten Befehl
- für jeden Befehl außer dem letzten gibt es einen nachfolgenden Befehl

Jedes Programm muss in einer Sprache verfasst sein, die der Prozessor versteht. Diese Sprache muss den Befehlssatz des Prozessors unterstützen.

1.1 Übersicht über Programmiersprachen

Eine Programmiersprache ist also eine Sprache, in der Programme abgefasst sind. Prinzipiell hat jede Sprache drei Eigenschaften:

- **Syntax** ("Rechtschreibung", vom Computer überprüfbar)
- **Pragmatik** (Ziel)
- **Semantik** (Inhalt eines Satzes)

Die verschiedenen Programmiersprachen haben außerdem gemeinsame Elemente.

Elemente einer Programmiersprache

Deklaration: Womit soll gearbeitet werden? Einführung von lesbaren Begriffen

Kommentar: Welcher Sinn steckt hinter einem Befehl?

Zuweisung: Mit welchen (Variablen-)Werten soll ein Programm arbeiten?

Beispiele

BASIC: `LET X = 2`

PASCAL: `X := 2`

LOGO: `MAKE "X 2`

Verzweigung: Wie soll auf verschiedene Argumente verschieden reagiert werden?

BASIC, PASCAL etc.: `IF – THEN`

Schleife: Wie oft soll etwas durchgeführt werden?

Beispiele

`WHILE, UNTIL, FOR`, ... (gilt für die unterschiedlichsten Programmiersprachen)

Ein-/Ausgabe: Welche Daten sollen woher genommen werden und wohin sollen die Ergebnisse geschrieben werden?

Beispiele

BASIC: `PRINT`

PASCAL: `WriteLn`

C: `scanf, printf`, ...

Unterprogramme: Welche Aktionen (Programmteile) werden sehr oft gebraucht?

Die ersten Programmiersprachen waren maschinennahe. Das Programmieren erfolgte "auf Bitebene" und war dementsprechend mühsam. Der reine Maschinencode wird auch als Programmiersprache der **1. Generation** bezeichnet.

Computer verstehen nur **binäre Informationen**, d.h. Informationen, die aus den Ziffern 0 und 1 bestehen. Programme in Maschinensprache bestehen daher nur aus einer Abfolge der Ziffern 0 und 1.

Später (um 1950) entwickelte man eine Möglichkeit, die Maschinenbefehle "sprachlich" zu formulieren: es entstanden die Programmiersprachen der **2. Generation**, auch **ASSEMBLER** genannt. Jeder Prozessor benötigt eigene Befehle, da Assembler nur eine "sprachliche" Formulierung von reinen Maschinenbefehlen ist. (d.h. jeder Prozessor "spricht" ein eigenes ASSEMBLER!) Ein ASSEMBLER-Statement entspricht also 1:1 einem Maschinenbefehl. ASSEMBLER-Programme sind daher sehr schnell. Diese Art der Programmierung verwendet man auch für den Zugriff auf die unterste Betriebssystemebene. Damit kann man Probleme lösen (z.B. Schnittstellen programmieren), für die Hochsprachen nicht geeignet sind.

Um dem Programmierer etwas das Leben zu erleichtern, kam man auf die Idee, den sehr abstrakten Binär-Befehlen (die im Hexadezimal-System dargestellt und eingegeben wurden) kurze, aber aussagekräftige Namen oder Bezeichnungen zu geben.

Beispiel

```
3F   MOV
7E   ADD
```

usw.

Ein typischer Assembler-Befehl sieht daher folgendermaßen aus:

```
JMP C0,34
```

Die Übersetzung des für den Menschen gedachten Prozessorbefehles (`MOV`, `ADD`, `JMP`, `SUB` usw.) in den für die Maschine verständlichen Code (`3F`, `7E`, `B3` usw.) wurde durch ein Übersetzungsprogramm durchgeführt.

Assembler stellt eine sehr hardwarenahe Programmiermethode dar. Jedem Prozessorbefehl wurde direkt ein Assemblerbefehl zugewiesen. Programmierer müssen also ausgezeichnet über den Prozessor selbst und dessen Möglichkeiten Bescheid wissen.

Dies hat den Vorteil, dass alle im Prozessor steckenden Möglichkeiten direkt, ohne Umweg über andere Befehle, ausgenutzt werden können. Das dadurch entstehende Programm wird also eine sehr schnelle Abarbeitung gewährleisten.

Ablauf der Entstehung eines Assembler-Programmes

Heute wird diese Sprache noch für Spezialanwendungen eingesetzt, um etwa einen sehr schnellen Programmcode zu erzeugen oder für technische Anwendungen (z.B. Steuerungen oder Regelungen). In der eigentlichen kommerziellen EDV ist Assembler allerdings fast bedeutungslos geworden.

Diese Prozessornähe, die einerseits einen schnellen Programmablauf gewährleistet, ist aber auch ein großer Nachteil:

- der Programmierer muss sich vor der Programmierung mit der internen Befehlsstruktur genau vertraut machen
- Programme können nicht einfach von einem Rechnersystem auf ein anderes übertragen werden – sie müssen neu umgeschrieben werden.

Diese Nachteile veranlasste Spezialisten schon sehr früh, Programmiersprachen zu entwickeln, wo der Programm-Source (das Quellprogramm) unabhängig von einer bestimmten Hardware geschrieben werden kann. Der Programmierer brauchte dann nur mehr die eigentliche

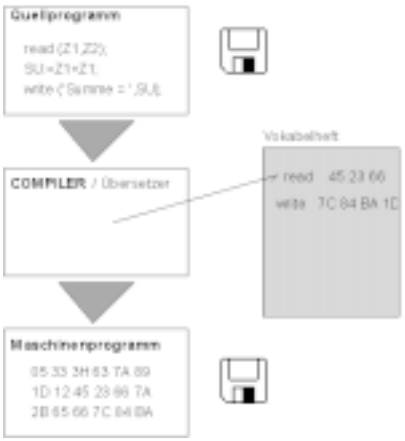
christian.zahler@telecom.at Christian Zahler

Programmiersprache lernen und sich nicht mehr über die Prozessoreinzelheiten den Kopf zerbrechen.

Diese Sprachen nennt man "Hochsprachen" oder auch "Sprachen der **3. Generation**". Die Befehlssyntax dieser Sprachen wird mit speziellen "Übersetzer-Programmen" in die Befehle der jeweiligen Prozessoren umgewandelt. Der "Wortschatz" der Programmiersprache richtet sich also nicht mehr nach den Prozessorbefehlen, sondern nach dem jeweiligen Anwendungsgebiet. Man unterscheidet zwei Arten von Übersetzerprogrammen: Compiler und Interpreter.

Gegenüber der Assembler-Sprache ist bei den höheren Programmiersprachen nur mehr der Übersetzer vom Prozessor abhängig.

Compiler: Programme werden als Ganzes übersetzt und als ausführbares .EXE-File gespeichert. Beispiele: C, COBOL, FORTRAN, ADA, PASCAL, ...



Ablauf der Entstehung eines Hochsprachen-Programmes

Ein Programm wird in einem Editor (meist) in einer höheren Programmiersprache eingegeben. (Ein Editor ist eine Art "Textverarbeitung für Programme".) Dann werden schrittweise Daten aus verschiedenen "Bibliotheken" hinzugefügt, wodurch der Programmcode immer länger wird. Der Compiler wandelt den ASCII-Code in eine binäre, für die Maschine lesbare Form um. Der Linker entscheidet schließlich, ob ein Programm als Background- oder Foreground-Programm abläuft; der Lader (EXEC) führt dann das Programm durch.

Interpreter: Programme werden zeilenweise abgearbeitet und ausgeführt; das Programm wird als "Textdatei" in der höheren Sprache abgespeichert. Code, der interpretiert wird, ist 20 bis 500mal langsamer als compilierter Code. Sprachen mit einem Interpreter-Übersetzungsprogramm sind gedacht als Experimentiersprachen für interaktives Arbeiten mit raschem Zugriff. Beispiele: BASIC, LISP, APL, PROLOG, LOGO,...

Wichtige Programmiersprachen der 3. Generation

FORTRAN: Abkürzung für "formula translator", entwickelt 1954 in den USA. Noch heute für naturwissenschaftliche Anwendungen in Gebrauch.

BASIC: Abkürzung für "beginners' all-purpose symbolic instruction code". Einsatz vor allem im kleinen Bereich günstig. Moderne Varianten besser. Entwickelt 1965 (USA).

COBOL: Abkürzung für "common business oriented language". Entwickelt 1959/60 vom amerikanischen Verteidigungsministerium. Gut lesbare Sprache für kommerzielle Anwendungen, sehr aufwendig. Textliche Manipulationen leicht möglich, Anwendung vor allem in der Wirtschaft.

ALGOL: Abkürzung für "algorithmic language". War bereits 1968 vorhanden, damals allerdings zu früh. Kommerzieller Flop.

PL/1: "Universalsprache", die die Vorteile und Möglichkeiten von ALGOL, FORTRAN und COBOL vereinigt. Durch diese Vielfalt ist die Sprache aber schwer erlernbar.

ADA: Benannt nach Ada Lovelace (Tochter von Lord Byron), entwickelt 1977–1980 in Frankreich. Heute die vom US-Verteidigungsministerium verwendete Programmiersprache.

PASCAL: Neukonzeption von FORTRAN nach ALGOL-Muster. 1969 von Niklaus Wirth (Schweiz) entwickelt. Gute Trainingssprache; ab 1970 entwickelte die Firma BORLAND "TURBO-PASCAL", eine PASCAL-Variante, die sich im technischen Bereich sehr stark durchgesetzt hat.

MODULA-2: Weiterentwicklung von PASCAL. Falls eine TURBO-Version entwickelt wird, könnte MODULA durchaus die Nachfolge von PASCAL antreten.

C: siehe nachfolgende Kapitel!

PROLOG: Abkürzung für "Programming in logic". 1973 in Frankreich entwickelt. Neuer Trend, bei dem Zusammenhänge programmiert werden und daraus Schlüsse gezogen werden sollen. Beispiel:

Angaben

Fakten:

```
X (is son of) Y
Y (is son of) Z
```

Relationen:

```
R(X,Z) = (grandfather)
```

Lösung:

andere Familienverhältnisse.

Grenzen:

Menge der Fakten und Relationen; wie erhält man die entsprechenden Relationen?

LOGO: 1967 Beranek, Newman. Grafisch orientierte Einsteigersprache, die ebenfalls strukturiertes Programmieren verlangt.

LISP: Abkürzung für "list processing". Wurde am MIT 1959–63 entwickelt (siehe auch "Künstliche Intelligenz"). Wird heute z.B. für die Steuerung von AutoCAD verwendet (als AutoLISP).

VISUAL BASIC: Neue Programmiersprache speziell für die Programmierung von WINDOWS-Applikationen.

Heute genügt es allerdings kaum mehr, in einem reinen Compiler eine Programmübersetzung in Maschinencode durchführen zu können. Die heutigen Forderungen sind flexible Programmlösungen mit guter Bedienerführung, möglichst eingebunden in die WINDOWS-Bedienoberfläche und eine in weiten Bereichen frei gestaltbare Auswertung. Ein Programmierer benötigt dazu heute eine sogenannte **Entwicklungsumgebung**, die den Programmierer von vielen lästigen Routinearbeiten (z. B. Gestaltung von Bildschirmmasken) entlastet.

Eine Entwicklungsumgebung gibt es auch in Visual C++, einer objektorientierten Weiterentwicklung von C, die auch die Programmierung von Windows-Programmen gestattet.

DELPHI: visuelle Entwicklungsumgebung, objektorientierte Architektur, Com-

Arbeiten in Visual C++ 5.0

piler erzeugt *.EXE-Programme, die ohne Laufzeit-Bibliothek arbeiten. Delphi basiert auf Object Pascal und wurde in Delphi (!) geschrieben.

JAVA: Eine sehr interessante Entwicklung scheint dieses von Sun entwickelte Produkt zu sein, dessen Rechte von IBM aufgekauft wurden. Der spezielle Vorteil dieser objektorientierten, in der Syntax an C++ angelehnten Programmiersprache ist, dass Java-Anwendungen ("Applets") über Netzwerke (auch Internet!) übertragen werden können und auf vielen verschiedenen Clients ausgeführt werden können (betriebssystemunabhängig!)! JavaScript wird als "Makrosprache" ab 1996 von Internet-Clients wie Netscape verwendet.

In diesem Abschnitt der Programmiersprachen der 3. Generation wurde schon erwähnt, dass es heute kaum mehr genügt, mit einem reinen Compiler eine Programmübersetzung in den Maschinencode durchführen zu können. Die EDV-Anwender von heute fordern flexible Programmlösungen mit guter Bedienerführung, Bildschirmmasken, wenn möglich eingebunden in die graphische WINDOWS-Bedieneroberfläche und eine im weiten Bereich frei gestaltbare Auswertung. Die Daten sollen auch vielfach in andere Produkte, wie Tabellenkalkulation, Grafiken, Textverarbeitung usw., auf einfache Art transportierbar sein.

Wollte der Programmierer all diese Forderungen mit der herkömmlichen Programmierung realisieren, würde er sicherlich bald resignieren.

Der Markt verlangt also nicht nur Programmiersprachen-Übersetzer (Compiler), sondern ganze Entwicklungssysteme, die den Programmierer von vielen lästigen, immer gleichbleibenden Routinearbeiten – wie z.B. die Gestaltung von Bildschirmmasken und Ausgabereports – entlasten.

Weiters zeigt sich, dass die Realisierung von Software, die Daten auf Festplatten ablegt, Daten sortieren oder indizieren muss und auf möglichst rasche Art diese wieder finden soll, mit herkömmlichen Programmiersprachen nur schwer realisierbar ist. Der Trend geht, sowohl am PC als auch auf größeren Systemen, in Richtung einer **DATENBANK**, welche dem Programmierer als auch dem Anwender alle Routineschritte abnimmt.

Nachdem fast alle Anwendungen mit einer derartigen Datenmanipulation arbeiten und strukturierte (oft gekoppelt mit unstrukturierten) Daten auf der Festlatte verwalten müssen, eignet sich eine Datenbank für sehr viele Anwendungen ausgezeichnet.

Aus all diesen Gründen haben sich Produkte am Markt durchgesetzt, die dem Programmierer als auch den Anwender sehr brauchbare Werkzeuge liefern, um in sehr kurzer Zeit (die Zeitersparnis gegenüber herkömmlichen Programmiersprachen beträgt oft den Faktor 100 und höher) eine Anwendung zu erstellen, die den heutigen Bedürfnissen entspricht. Diese Systeme wurden als die **4. GENERATION** bezeichnet.

Ein derartiges System der 4. Generation wird folgende Merkmale aufweisen:

- **Integrierte Datenbankverwaltung**: d.h. Indexaufbau und Verwaltung, integrierte Befehle für Sortierung, Datensuche, Datenverknüpfungen, . . .
- **Datums- und Zeitfunktionen**: Mit Datum und Uhrzeit kann gerechnet werden (ohne komplizierte Umrechnugsvorgänge)
- **Netzwerkbetrieb**: Zentrale Daten können vielen Benutzern zugänglich gemacht werden.
- **Leichte Datenabfrage**: mittels spezieller Befehle kann man vom Programm oder direkt (als Anwender) die Datenbestände abfragen, verknüpfen und auswerten. Zu dieser Datenabfrage haben sich gewisse Standards entwickelt, z.B.: SQL (structured query language).
- **Einfache Bildschirmmasken**: mittels Maskengenerator kann man, ohne Programmierkenntnisse, Masken für die Dateneingabe erzeugen. Diese haben nicht nur eine ansprechende Form, sondern können auch gleichzeitig die Eingabe kontrollieren (z.B. Datum,...)
- **Reports**: wiederum ohne Programmierkenntnisse sind Ausgabelisten (am Bildschirm oder Drucker) mit geeigneten Überschriften, Seitenanzeigen usw. erzeugbar.
- **Transaktionssteuerung**: Dieses Verfahren trägt ausgezeichnet zur Datensicherheit und Datenkonsistenz bei. Gewisse Eingabe- und Verarbeitungsschritte bilden eine Transaktion. Kommt es während dieser Transaktion zu einem Fehler (Systemzusammenbruch, Fehlbedienung,...), so wird automatisch der Zustand vor der Transaktion wieder hergestellt.
- **Datenübertragung in andere Systeme**: die Datenbankstruktur ist genormt und daher leicht von einer Standardsoftware (z.B. Grafikprogrammen) lesbar.

Nicht nur am PC, auch im Zentralrechnerbereich erfreuen sich Datenbanken heute immer größerer Beliebtheit. In diesem Zusammenhang sei nochmals das bereits unter Windows kurz genannte **Client-Server-Verfahren** erwähnt. Dort liegt auf einem zentralen Rechner eine Datenbank, externe kleine Rechner fragen diese Datenbank ab und stellen den Kontakt zum Benutzer her.

Typische Vertreter der Programmiersprachen der 4. Generation am PC

dBASE (III+, IV)

dBASE war die erste weitverbreitete Datenbankanwendung am PC. Auf das Datenbankformat von dbase (.dbf) können viele andere Produkte zugreifen.

dBASE ist ein Interpreter, mit dem der Anwender direkt (ohne Programm) auf den Datenbestand Zugang hat. Über diesen direkten Modus kann man jedoch auch mit eigens erstellten Programmen arbeiten.

dBASE bietet alle Eigenschaften und Merkmale, die auf der vorigen Seite dargestellt wurden.

dBASE selbst ist nur als Interpreter verfügbar (es können keine eigenständigen .EXE-Dateien erzeugt werden) und läuft nur unter DOS..

CLIPPER

Mit Clipper kann man mit weitestgehend identen Befehlen wie in dBASE Programme erzeugen, die direkt vom Betriebssystem aus lauffähig sind (.EXE-Dateien) und auf die gleiche Datenstruktur wie dBASE zugreifen (.dbf-Format).

SQL

(Structured Query Language) Mit den meisten professionellen Datenbanken (Progress, Gupta, Oracle, Paradox usw.) wird meist eine SQL mitgeliefert, eine Sprache, mit der Datenbank-Abfragen strukturiert programmiert werden können.

2 Professionelle Entwicklung von Software-Projekten

Einen Hauptbereich der Informatik stellt die Entwicklung neuer Software dar. Wenn man von "Privatprogrammierern" absieht, so benötigt ein kommerzielles Softwareprojekt fast immer ein größeres Team, um eine professionelle Entwicklung zu garantieren.

Bei der Realisierung größerer Software-Projekte versucht man, den Herstellungsprozeß in definierte Phasen zu untergliedern, um eine Funktions-, Termin- und Kostenkontrolle zu ermöglichen.

Ein solches **Phasenmodell** könnte aus folgenden Phasen aufgebaut sein:

Phase 1: **Problemanalyse**. Auftraggeber und Hersteller untersuchen das mit EDV zu lösende Problem unter Einbeziehung der vorhandenen Rahmenbedingungen (vorhandene Organisationsstruktur, Hardware etc.). Ergebnis ist ein Lastenheft, in dem genau steht, was die zu entwickelnde Software leisten soll.

Klasse	Anzahl Teammitglieder	Zeit (Jahre)	Programmzeilen	Beispiele
Klein	1	0,1 – 0,5	1000 – 2000	Kleine kommerzielle Anwendungen, Berechnungen
Mittel	2 – 5	1 – 2	10 000 – 50 000	Kleine Compiler, Assembler; 2D-CAD; Lagerverwaltung; Prozeßrechnerapplikation
Groß	5 – 20	2 – 3	50 000 – 100 000	Größere Compiler; kleinere Betriebssysteme; CAD/CAM-Systeme; Echtzeitanwendungen
Sehr groß	100 – 1000	4 – 5	rund 1 000 000	Datenbanksysteme; große Betriebssysteme; militärische Kontrollsysteme; komplexe CAD/CAM/CAE-Systeme
Extrem groß	2000 – 5000	5 – 10	106 – 107	Luftverkehrsüberwachung; militärische Raketenabwehrsysteme

Beispiele für Softwareprojekte

Phase 2: **Spezifikation**. Hier müssen die funktionalen Eigenschaften des Systems, seine Benutzerschnittstellen und Datenstrukturen festgelegt werden. Auch die Funktionstests und die Entwurfsmethoden für die folgenden Phasen werden festgelegt. Das Ergebnis ist ein Pflichtenheft (Spezifikationsdokument), das den Aufgabenkatalog für die weitere Entwicklung darstellt.

Phase 3: **Detailplanung** . Die Software wird weiter strukturiert und in kleinere Einheiten zerlegt.

Phase 4: **Codierung**. Hier werden die bei der Detailplanung definierten Funktionseinheiten in eine konkrete Programmiersprache umgesetzt. Hier sind auch die notwendigen Tests laufend durchzuführen. Wenn alle Komponenten fehlerfrei arbeiten, so können sie zu einem Ganzen zusammengefügt werden. Erfüllt das Gesamtsystem alle im Spezifikationsdokument gestellten Anforderungen, so kann die Gesamtdokumentation fertiggestellt werden.

Phase 5: **Übergabe und Systemeinführung**. Das fertige Programm wird an den Auftraggeber übergeben und installiert. Dazu gehört eine Einführung der zukünftigen Anwender in das neue System.

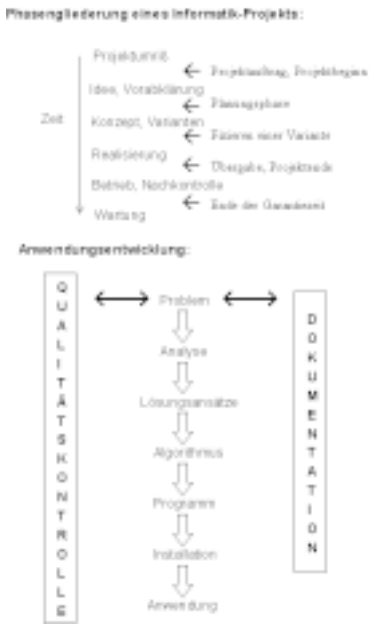
Phase 6: **Wartung und Betreuung**. Im Echtbetrieb wird es sicherlich immer wieder zu Komplikationen oder Situationen kommen, die den Kontakt zum Programmersteller notwendig machen. Programmabläufe können durch Fehlbedienung oder durch Eingabe falscher Daten gestört sein. Es können aber auch Anpassungen oder Erweiterungen von Programmen notwendig oder wünschenswert sein. Echte Hilfe dabei können oft nur jene Personen garantieren, die mit der Programmerstellung befasst waren. Der Anwender kann durch eigene **Wartungsverträge** diesen Situationen einigermaßen vorbeugen. **Achtung**: Es ist oft sehr schwierig, aus bestehenden Wartungsverträgen wieder auszusteigen! (Die Vertragsbedingungen sorgfältig durchlesen!)

Als "Zwischenprodukt" jeder Phase wird ein Dokument verlangt (Spezifikationsdokument, Gesamtdokumentation usw.).

Zur Realisierung der Phasen 1 – 3 gibt es *Entwurfsmethoden*, darunter versteht man (meist grafische) Darstellungen, die die umgangssprachliche Formulierung des Projekts formalisieren, ohne dabei auf Einzelheiten der programmtechnischen Realisierung (etwa Programmiersprache) einzugehen.

Wenn manche Software von den Anwendern als unbrauchbar bezeichnet wird, so ist eine oder mehrere oben beschriebener Stufen gar nicht oder mangelhaft durchgeführt worden.

Der beste Schutz vor späteren unliebsamen Zwischenfällen ist jedoch der Einsatz einer Produktpalette, die eine entsprechende Marktfülle bietet und von weltweit tätigen Firmen unterstützt wird. Der Einsatz von "exotischer" Software sollte, wie schon erwähnt, gut überlegt sein und nur in bestimmten Fällen erfolgen.



Die Qualitätskontrolle erfolgt in der Wirtschaft z.B. durch Testen kleiner Module. Eine Dokumentation ist sehr wichtig für die Produkt-Weiterentwicklung und die Reproduzierbarkeit.

Ein Programm ist also im Prinzip ein Weg zur Lösung eines aufgetretenen Pro-

blems. Es besteht aus einer Sammlung von Vorschriften an die Maschine, einem **Algorithmus**. Dabei ist ein Algorithmus noch völlig losgelöst vom Computer, d.h. er kann nicht falsch sein, wohl aber nicht zielführend.

Unter einem Algorithmus versteht man eine (Rechen-)Vorschrift, die ein bestimmtes Problem in elementare Einzelschritte zerlegt und die Reihenfolge festlegt, in der diese Einzelschritte ausgeführt werden müssen. Dabei muss die Anzahl der Schritte bis zur Lösung des Problems endlich sein.

3 Erstellen von Algorithmen; graphische Hilfsmittel

Beispiel eines Algorithmus

Bubble-Sort (Sortieren von Zahlen durch paarweises Vergleichen):

1 Merk dir die erste Zahl der zu ordnenden Liste.

2 Vergleiche diese Zahl mit allen anderen Zahlen.

3 Falls du eine kleinere Zahl darunter findest, merk sie dir statt der ersten.

4 Wenn du alle Zahlen verglichen hast, schreib die Zahl, die du gerade weißt, an und streich sie aus der Liste.

5 Gehe zu 1.

Ein Algorithmus wird in einer (Programmier-)Sprache formuliert.

Beim Erstellen (und besonders bei der Dokumentation) von Algorithmen sind

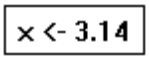
graphische Hilfsmittel unerlässlich. Dabei verwendet man meist das **NASSI-SHNEIDERMAN-Diagramm** (**Struktogramm**): flächenhafte Darstellung mittels "Strukturblocks".

Dabei gibt es folgende Darstellungen:

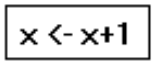
Begrenzung:

Operationen:

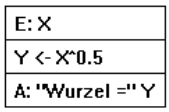
1. Beispiel: Zuweisung



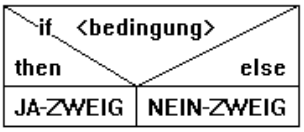
2. Beispiel: Zähler



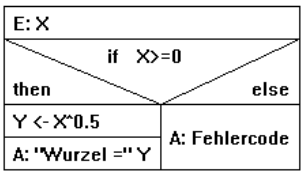
Ein- und Ausgabe: (Beispiel: Wurzelziehen)



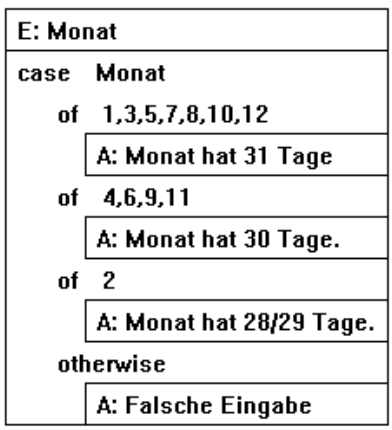
Verzweigung:



Beispiel: Wurzelziehen



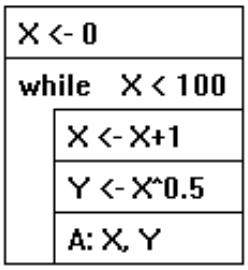
Mehrfache Verzweigung:



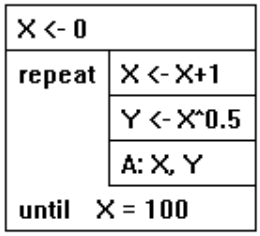
Wiederholungsstrukturen ("Schleifen"):

Beispiel: Wurzeltabelle für die Zahlen 1 bis 99

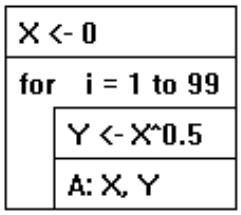
"kopfgesteuert"



"schwanzgesteuert"



"Zählschleife" mit bekannter Durchlaufzahl



Unterprogramme:

Kommentare können mit Pfeilen oder nur so am Rand ergänzt werden. Beim Struktogramm sind bei sog. "einseitigen Verzweigungen" auch gezielte "Leerblöcke" gestattet (sie können - um die Absicht des Vorhabens zu kennzeichnen - ein Zeichen wie % enthalten).

Bevor die erste Zeile codiert wird, sollte ein fertiges Struktogramm vorliegen!.

Programme zur Entwicklung von Struktogrammen: CASE-Tools, zum Beispiel EASYCASE.

CASE bedeutet "Computer Aided Software Engineering" (deutsch: Computerunterstützte Softwareerstellung). Vorteil: Nach der Entwicklung des Struktogramms kann automatisch (in verschiedenen Programmiersprachen!) kodiert werden.

4 Zahlensysteme

4.1 Das Dezimalsystem

Unser gewohnter Umgang mit Zahlen und unser Vorstellungssystem über die Größe einer Zahl basiert auf diesem Zahlensystem. Wir benutzen dieses Zahlensystem täglich und haben gelernt, damit zu rechnen, ohne dass wir uns über seinen eigentlichen Aufbau so richtig Gedanken machen.

Wir wollen uns einmal überlegen, was die Schreibweise einer bestimmten Zahl bedeutet:

$$3759 = 3000 = 3 * 1000 = 3 * 10^3$$
$$= 700 = 7 * 100 = 7 * 10^2$$
$$= 50 = 5 * 10 = 5 * 10^1$$
$$= 9 = 9 * 1 = 9 * 10^0$$
$$3759 = 3759 = 3759 = 3759$$

Wir sehen bei dieser unterschiedlichen Schreibweise der Zahl 3759, dass diese aus Ziffern (0 bis 9) bestehen, die entsprechend ihrer Stelle mit einem Basiswert multipliziert werden. Je weiter links eine Ziffer steht, desto größer ist ihre Wertigkeit (der 3er links ist also viel höherwertig als der 9er rechts). Die **BASIS** des Dezimalsystems ist daher **10**.

Dezimal ziffer	**3**	**7**	**5**	**9**
	1000er -stelle	100er- stelle	10er- stelle	1er- stelle
Stellen wert	$\cdot 10^3$ $\cdot 1000$	$\cdot 10^2$ $\cdot 100$	$\cdot 10^1$ $\cdot 10$	$\cdot 10^0$ $\cdot 1$
3759=	3·1000 +	7·100 +	5·10 +	9·1

4.2 Das Binär- oder Dualsystem (Zweiersystem)

Dual ziffer	**1**	**0**	**0**	**1**
	8er-stelle	4er-stelle	2er-stelle	1er-stelle
Stellen wert	$\cdot 2^3$ $\cdot 8$	$\cdot 2^2$ $\cdot 4$	$\cdot 2^1$ $\cdot 2$	$\cdot 2^0$ $\cdot 1$
9=	8 +	0 +	0 +	1 =

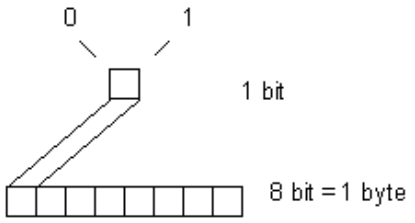
Als Ziffernvorrat pro Stelle kommt nur 0 und 1 vor, als Basis fungiert die Ziffer 2 (darum 2er-System). Die Wertigkeit einer Stelle im 2er-System ist demnach der Wertigkeit im 10er-System weit unterlegen.

Das heißt, man braucht im Binärsystem wesentlich mehr Stellen, um eine gleich große Zahl darzustellen, als im Dezimalsystem.

Beispiel: Die 3. Stelle im Dezimalsystem hat die Wertigkeit 100, im 2er-System nur 4.

Daraus ergibt sich, dass man zur Darstellung des gleichen Zahlenwertes im Dualsystem wesentlich mehr Stellen benötigt als im Zehnersystem. Es sind aber trotzdem alle Zahlenwerte darstellbar.

Die Informationen werden im Inneren des Computers als Ketten von binären Informationseinheiten = **Bit** (bit = engl. binary digit) gespeichert. Eine binäre Einheit ist die Entscheidung 0/1 oder Strom/kein Strom usw.



8 binäre Einheiten werden als 1 Byte (engl. "by eight") bezeichnet.

4.3 Das Hexadezimalsystem

Wir wissen nun, dass das zentrale Element der Datenspeicherung- und Verarbeitung eine 8stellige Binärzahl ist (welche einem Stromfluss oder einer Magnetisierung entspricht). Nachdem diese Zahlen nur sehr schwer merkbar und darstellbar sind, hat man nach einer Möglichkeit gesucht, diese langen Zahlen durch kürzere zu ersetzen. Als sehr optimal bot sich an, 4 Stellen einer Binärzahl (das ist ein Halbbyte) nur durch eine einzige Stelle eines anderen Zahlensystems darzustellen.

Unser gewohntes Dezimalsystem kam nicht in Frage, da zur Abdeckung von 4 Binärstellen bis zu 2 Dezimalstellen (0 – 15) notwendig sind.

Die Basis jenes neuen Zahlensystemes muss daher 16 sein (2^4). An jeder Stelle – die eine höhere Wertigkeit als das Dezimalsystem aufweist – können Ziffern zwischen 0 und 15 vorkommen. Da man jedoch keine einstelligen Ziffern für 10 bis 15 kennt, lauten die Ziffern des **HEXADEZIMALSYSTEMS**:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Binärzahl	Dezimal-zahl	Hexadezi-malzahl
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Beispiel

Hex ziffer	**C**	**3**	**0**	**A**
	4096er-stelle	256er-stelle	16er-stelle	1er-stelle
Stellen wert	$\cdot 16^3$ $\cdot 4096$	$\cdot 16^2$ $\cdot 256$	$\cdot 16^1$ $\cdot 16$	$\cdot 16^0$ $\cdot 1$
49930=	12.4096 +	3.256 +	0 +	10.1

Anhang: Internationale IBM-ASCII-Tabelle

```
32:      55: 7   78: N   101: e  124: ¦   147: ô  170: ¬  193: Á  216: Ø  239: n
33: !    56: 8   79: O   102: f  125: }   148: ö  171: ½  194: Â  217: Ù  240: =
34: "    57: 9   80: P   103: g  126: ~   149: ò  172: ¼  195: Ã  218: Ú  241: ±
35: #    58: :   81: Q   104: h  127:     150: û  173: ¡  196: Ä  219:    242: =
36: $    59: ;   82: R   105: i  128: Ç   151: ù  174: «  197: Å  220: _  243: =
37: %    60: <   83: S   106: j  129: ü   152: ÿ  175: »  198: Æ  221:    244: (
38: &    61: =   84: T   107: k  130: é   153: Ö  176:    199: Ç  222:    245: )
39: '    62: >   85: U   108: l  131: â   154: Ü  177:    200: È  223: ¯  246: ÷
40: (    63: ?   86: V   109: m  132: ä   155: ¢  178:    201: É  224: á  247: ¯
41: )    64: @   87: W   110: n  133: à   156: £  179: ³  202: Ê  225: ß  248: °
42: *    65: A   88: X   111: o  134: å   157: ¥  180: ´  203: Ë  226: Ã  249: ?
43: +    66: B   89: Y   112: p  135: ç   158: P  181: µ  204: Ì  227: p  250: ·
44: ,    67: C   90: Z   113: q  136: ê   159: ƒ  182: ¶  205: Í  228: Õ  251: v
45: -    68: D   91: [   114: r  137: ë   160: á  183: ·  206: Î  229: õ  252: n
46: .    69: E   92: \   115: s  138: è   161: í  184: ¸  207: Ï  230: µ  253: ²
47: /    70: F   93: ]   116: t  139: ï   162: ó  185: ¹  208: Ð  231: ô  254:
48: 0    71: G   94: ^   117: u  140: î   163: ú  186: º  209: Ñ  232: Ö  255:
49: 1    72: H   95: _   118: v  141: ì   164: ñ  187: »  210: Ò  233: È
50: 2    73: I   96: `   119: w  142: Ä   165: Ñ  188: ¼  211: Ó  234: Ù
51: 3    74: J   97: a   120: x  143: Å   166: ª  189: ½  212: Ô  235: ä
52: 4    75: K   98: b   121: y  144: É   167: º  190: ¾  213: Õ  236: 8
53: 5    76: L   99: c   122: z  145: æ   168: ¿  191: ¿  214: Ö  237: ö
54: 6    77: M   100: d  123: {  146: Æ   169: ¬  192: Ã  215: ×  238: å
```


Christian Zahler christian.zahler@telecom.at