

Einführung in die Informatik

<http://vfheinf.oncampus.de>

Stand 05.06.2016 11:26



Inhalt

Einführung in die Informatik	5
1 Einleitung	6
2 Einführung	7
2.1 Was ist Informatik?	7
2.2 Analog - Digital	8
2.3 Rechenanlagen und ihre Programmierung	8
3 Modellierung	13
3.1 Konzepte, Modelle, Modellbildung	13
3.1.1 Graphen	15
3.1.2 Petri-Netze	18
3.1.3 Entity-Relationship-Modell (ER-Modell)	22
3.1.4 Unified Modeling Language (UML)	24
3.1.5 Aufgaben zum Thema: Konzepte, Modelle, Modellbildung	27
3.2 Simulation	29
3.3 Software - Entwicklungsprozess	34
3.4 Fragen mit Musterlösungen: Modellierung	37
4 Information und Nachricht	43
4.1 Informelle Einführung der Begriffe	43
4.2 Digitale Nachrichten, Codes	45
4.3 Nachrichten- und Informationsverarbeitung	48
4.4 Aufgaben zum Thema: Information und Nachricht	51
4.5 Fragen mit Musterlösungen: Information und Nachricht	52
5 Zahlen und Zahlssysteme	55
5.1 Mathematischer Zahlbegriff	55
5.2 Die Ursprünge von Zahlensystemen	59
5.3 Stellenwertcodes und Konvertierung ganzer Zahlen	59
5.4 Darstellung negativer ganzer Zahlen	64
5.5 Addition und Subtraktion	67
5.6 Darstellung von Gleitpunktzahlen	70
5.7 Aufgaben zum Thema: Zahlen und Zahlensysteme	72
5.8 Fragen mit Musterlösungen: Zahlen und Zahlssysteme	73
6 Algorithmen und Datenstrukturen	76
6.1 Algorithmen	76
6.1.1 Theoretische Fragestellungen zu Algorithmen	84
6.1.2 Algorithmisierung	89
6.2 Datenstrukturen	94
6.2.1 Abstrakte Datentypen	95

6.2.1.1	Basisdatentypen	97
6.2.1.2	Andere abstrakte Datentypen	102
6.2.2	Dynamische Datenstrukturen	103
6.2.2.1	Listen	104
6.2.2.2	Bäume	112
6.3	Sortieren	119
6.4	Aufgaben zu Algorithmen und Datenstrukturen	125
7	Aufbau eines Rechnersystems	128
7.1	Begriffserklärung	128
7.2	Das Schichtenmodell eines Rechnersystems	128
7.3	Die Struktur der von Neumann-Maschine	130
7.4	Prozessorarchitekturen	132
7.5	Maschinenbefehle und Mikroprogrammierung	136
7.6	Ein-/Ausgabeorganisation	138
7.7	Multimedia-Peripherie	141
7.8	Bussysteme	145
7.9	Speichertechnologien	148
7.10	Leistungsgrößen und Leistungsbewertung	150
7.11	Konzepte der Parallelverarbeitung	153
7.11.1	SIMD-Architekturen	156
7.11.2	MIMD-Architekturen	158
7.12	Aufgaben zum Thema Rechnerhardware	160
7.13	Fragen mit Musterlösungen: Rechnerhardware	162
8	System- und Anwendungssoftware	166
8.1	Betriebssysteme	166
8.2	Basissysteme	171
8.3	Anwendungssysteme	179
8.4	Aufgaben zum Thema: System- und Anwendungssoftware	180
8.5	Fragen mit Musterlösungen: System- und Anwendungssoftware	181
9	Rechnernetze	186
9.1	Datenkommunikation	186
9.2	Aufgaben von Rechnernetzen	188
9.3	Ausdehnung von Rechnernetzen	189
9.4	Netzstrukturen und -architekturen	189
9.5	Internet und Dienste im Internet	197
9.6	Aufgaben zum Thema Rechnernetze	199
9.7	Fragen mit Musterlösungen: Rechnernetze	200

Anhang

I Abbildungsverzeichnis	203
II Tabellenverzeichnis	208
III Medienverzeichnis	209
IV Aufgabenverzeichnis	210
V Index	213

Einführung in die Informatik



Gliederung

Einführung in die Informatik

- 1 Einleitung
- 2 Einführung
- 3 Modellierung
- 4 Information und Nachricht
- 5 Zahlen und Zahlssysteme
- 6 Algorithmen und Datenstrukturen
- 7 Aufbau eines Rechnersystems
- 8 System- und Anwendungssoftware
- 9 Rechnernetze



1 Einleitung

Das Studienmodul beginnt mit einer Einführung in den Begriff der Informatik und, damit verbunden, von digitalen Rechenanlagen. Es folgt ein Kapitel über Modellierung. Hier werden mit Graphen, Entity - Relationship - Modellen und der Unified Modeling Language Ansätze zur Problemdarstellung und damit auch zu einer Problemlösung gegeben. Mit dem Begriff der Simulation wird ein zentrales Anliegen der Informatik vorgestellt. Das Kapitel 4 befasst sich mit der Unterscheidung von Information und Nachricht und führt den Begriff der Codierung ein. Im Kapitel 5 werden Zahlen und Zahlensysteme besprochen, insbesondere das Dualsystem. Ebenfalls zu den Grundlagen der Informatik gehört durchaus die Erläuterung der Algorithmen und Datenstrukturen.

Die Kapitel 7, 8 und 9 geben einen Überblick über Rechnerhardware, Rechnersoftware und Rechnernetze. In die einzelnen Themen wird ohne Bezug zu konkreten Produkten eingeführt. Im Vordergrund steht ihr jeweiliger prinzipieller Aufbau und ihre Wirkungsweise. Gerade bei der Schnelligkeit der heutigen Systeme hat das Grundlegende eine deutlich längere Gültigkeitsdauer.

2 Einführung



Gliederung

2 Einführung

2.1 Was ist Informatik?

2.2 Analog - Digital

2.3 Rechenanlagen und ihre Programmierung

Dieses Kapitel beschreibt sehr kurz, was Informatik eigentlich ist, und liefert unter Punkt 2.3 auch ein paar historische Informationen, die zeigen, in welchen historischen Kontext sie eingeordnet werden kann.

2.1 Was ist Informatik?

Informatik ist die Wissenschaft, Technik und Anwendung der maschinellen Verarbeitung und Übermittlung von Informationen, insbesondere mittels Digitalrechner (Computer).

Die Informatik umfasst:

- Theorie (wissenschaftliche, rein gedankliche Betrachtungsweise, Gegensatz: Praxis; der Begriff Theorie sollte nicht verwechselt werden mit dem Begriff Hypothese)
- Methodik (Lehre von den Methoden)
- Analyse (von Problemen) und Konstruktion (von Lösungen) Anwendungen
- Auswirkungen des Einsatzes

Gebiete der Informatik sind:

- Theoretische Informatik (z.B. Automaten und Formale Sprachen)
- Praktische Informatik (Algorithmen, Software und Anwendungen)
- Technische Informatik (Hardware)

Gelegentlich wird als weiteres, zusätzliches Gebiet die Angewandte Informatik genannt. Hier lassen sich die Medieninformatik und andere Anwendungsgebiete der Informatik einordnen. Das Wort Informatik ist 1968 in Europa (außer in Großbritannien) eingeführt worden. Kreiert wurde es zunächst in Frankreich als *informatique*. Im Angelsächsischen ist stattdessen der Begriff *computer science* gebräuchlich geworden. Zu dieser Zeit hat sich die Informatik auch als eigenständige wissenschaftliche Disziplin

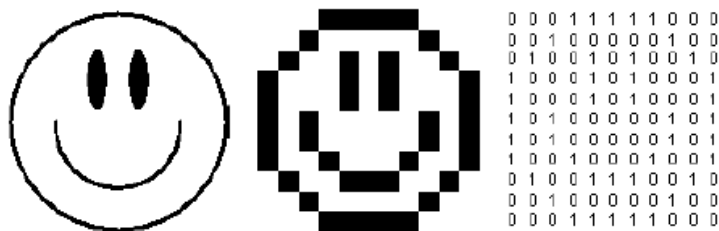
entwickelt. Ihre Wurzeln liegen einerseits in der Mathematik, andererseits in der Elektrotechnik und Nachrichtentechnik.

2.2 Analog - Digital

Die Informatik beschäftigt sich insbesondere mit der Verarbeitung und Übermittlung von Information mittels digitaler Rechenanlagen. In der Vergangenheit auch verwendete analoge Rechenanlagen sind heute nicht mehr im Einsatz.

Analog:	Kontinuierlich, stufenlos, unendlicher Wertebereich
Diskret:	Endlicher Wertebereich, unstetig
Digital:	Diskret, ziffernhalt (meist wird ein binäres Ziffernsystem verwendet)
Diskretisierung/ Rasterung:	Vorgang der Ersetzung eines unendlichen durch einen endlichen Wertebereich
Digitalisierung:	Umwandlung der diskreten Werte in eine digitale Darstellung

In untenstehender Abbildung ist ein Beispiel zur Digitalisierung aufgeführt.



Digitalisierung eines Bildes (analog - diskret - digital)

2.3 Rechenanlagen und ihre Programmierung

Historische Entwicklung von Rechenanlagen

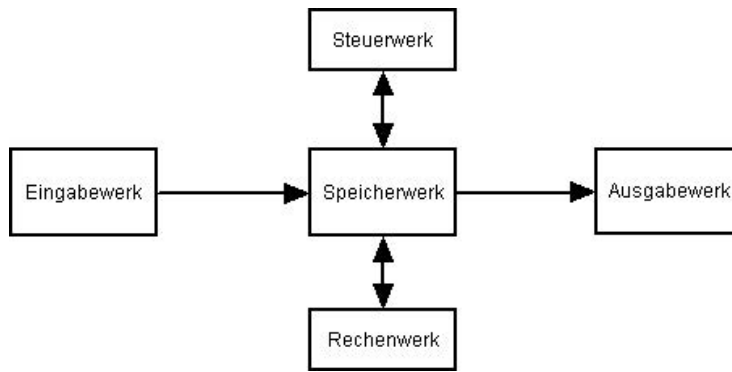
1700 v. Chr.	Papyros Rhind, Ägypten: Älteste schriftliche Rechenaufgaben.
--------------	--------------------------------------------------------------

300 v. Chr.	Euklidischer Algorithmus.
5 n. Chr.	Erfindung des Dezimalsystems in Indien.
820 n. Chr.	Al-Chowarizmi (etwa 780 bis 850), persischer Mathematiker und Astronom, Buch über Algebra.
1202	Leonardo von Pisa, genannt Fibonacci (etwa 1180 - 1240), italienischer Mathematiker, verfasst den liber abaci, die erste systematische Einführung in das dezimale Rechnen.
1524	Adam Riese (1492 - 1559) veröffentlicht ein Rechenbuch, in dem er die Rechengesetze des Dezimalsystems beschreibt. Seit dieser Zeit setzt sich das Dezimalsystem in Europa durch.
1623	Wilhelm Schickhard (1592 - 1635) konstruiert eine Maschine, die die vier Grundrechenarten ausführen kann.
1641	Blaise Pascal (1623 - 1662) konstruiert eine Maschine, mit der man sechsstellige Zahlen addieren kann.
1674	G. W. Leibniz (1646 - 1716) konstruiert eine Rechenmaschine mit Staffelwalzen für die vier Grundrechenarten. Er befasst sich auch mit dem dualen Zahlensystem.
1774	P. M. Hahn (1739 - 1790) entwickelt die erste zuverlässig arbeitende mechanische Rechenmaschine.
1805	J. M. Jacquard (1752 - 1834) entwickelt einen durch gelochte Holzplättchen programmgesteuerten Webstuhl.

1822	Charles Babbage (1791 - 1871) plant seine Analytical Engine (eine programmgesteuerte, mechanische Rechenmaschine).
1886	Hermann Hollerith (1860 - 1929) erfindet die Lochkarte.
1941	Konrad Zuse (1910 - 1995) entwickelt eine programmgesteuerte, elektromechanische Rechenmaschine, genannt Z3. Diese verwendet das duale Zahlensystem und die Gleitpunkt-Zahlendarstellung.
1944	H. H. Aiken (1900 - 1973) erbaut Mark I.
1946	J. P. Eckert und J. W. Mauchly entwickeln den ENIAC, den ersten vollelektronischen Rechner (18.000 Elektronenröhren).
1946	John von Neumann (1902 - 1957) schlägt das gespeicherte Programm vor. Er entwickelt ein prinzipielles Modell eines universellen Rechners.
1949	M. V. Wilkes baut EDSAC, den ersten universellen Rechner mit gespeichertem Programm.
ab 1950	Industrielle Rechnerproduktion (1. Rechnergeneration).

Prinzipielle Wirkungsweise einer Rechenanlage

Der prinzipielle Aufbau einer digitalen, programmgesteuerten Rechenanlage ist in untenstehender Abbildung als so genannter von Neumann-Rechner (1946) dargestellt.



Blockbild eines von Neumann-Rechners

Ein von Neumann - Rechner ist in dem Sinne universell, dass beliebige Programme von ihm abgearbeitet werden können. Erst ein Programm macht ihn arbeitsfähig. Programme und Daten werden über das Eingabewerk im Speicher abgelegt, Ergebnisse über das Ausgabewerk ausgegeben. Die Abarbeitung eines Programms wird über das Steuerwerk gesteuert, und die Operationsausführung erfolgt im Rechenwerk. Ergebnisse und Zwischenergebnisse werden im Speicher abgelegt. Programme und Daten werden als Folge von 0 und 1 in Einheiten fester Länge dargestellt. Ein Rechner "versteh" seine "Maschinensprache".

Abstraktionsgerade der Programmierung einer Rechenanlage

Maschinenprogramm (eines fiktiven 16-bit-Rechners):

0 1 1 0 0 0	0 0 0 1 1 0 0 1 0 0
0 1 0 0 0 0	0 0 0 1 1 0 0 1 0 1
0 1 1 1 0 0	0 0 0 1 1 0 0 1 1 0
Operationsteil	Adressteil



Code

Assemblerprogramm (mnemonischer Code, "Merkhilfe"-Code)

Übersetzung durch Assembler

LDA 100;	lade aus Speicherzelle 100 in Reg.
ADA 101;	addiere Speicherzelle 101 zu Reg.
STA 102;	speichere Reg. in Speicherzelle 102

Anweisung in einer höheren Programiersprache Übersetzung durch Compiler

```
c := a + b;
```

**Definition****Programmierung:**

Programmieren ist das Konstruieren und Formulieren von Algorithmen und deren Übertragung in eine Sprache, die ein Computer versteht.

Algorithmus:

Ein Algorithmus ist ein allgemeines, eindeutiges Verfahren zur Lösung einer Klasse gleichwertiger Probleme, gegeben durch einen aus endlich vielen elementaren Anweisungen bestehenden Text.

3 Modellierung



Gliederung

3 Modellierung

3.1 Konzepte, Modelle, Modellbildung

3.2 Simulation

3.3 Software - Entwicklungsprozess

3.4 Fragen mit Musterlösungen: Modellierung

In diesem Kapitel werden die Begriffe der Modellierung erläutert. Darüber hinaus wird auf den beim Einsatz von elektronischer Datenverarbeitung zentralen Begriff der Simulation eingegangen. Methoden der Softwareentwicklung werden kurz angerissen.

3.1 Konzepte, Modelle, Modellbildung



Gliederung

3.1 Konzepte, Modelle, Modellbildung

3.1.1 Graphen

3.1.2 Petri-Netze

3.1.3 Entity-Relationship-Modell (ER-Modell)

3.1.4 Unified Modeling Language (UML)

3.1.5 Aufgaben zum Thema: Konzepte, Modelle, Modellbildung

In der angewandten Informatik geht es im Wesentlichen darum, den Computer zur Unterstützung des menschlichen Handelns einzusetzen. Dies kann natürlich nicht allumfassend erfolgen; es werden deshalb immer nur abgrenzbare Aktivitäten von einem Anwendungssystem abgedeckt. Man spricht davon, dass jeweils nur ein Ausschnitt aus der realen Welt betrachtet wird.



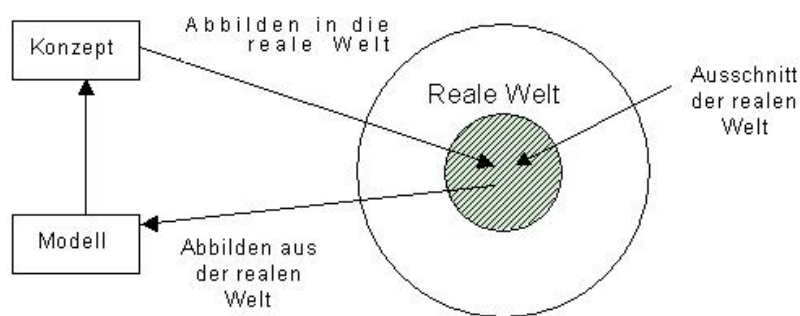
Beispiel

Ein Internet-Warenhaus möchte seine Produkte potentiellen Kunden anbieten und verkaufen. Die Waren sind, einschließlich Preisangabe, darzustellen und zu beschreiben. Der Kunde soll diese bestellen, d.h. in einen Warenkorb legen, können, und er muss Angaben über Lieferanschrift und Form der Bezahlung machen können. Dass ein solches

Warenhaus z.B. auch Mitarbeiter hat, die ein monatliches Gehalt bekommen, ist für den obigen Ausschnitt unerheblich.

Der betrachtete Ausschnitt der realen Welt ist zu modellieren, d.h. so zu beschreiben, dass alles Notwendige vorhanden ist und das nicht Notwendige weggelassen ist. Ein Modell beschreibt die Zusammenhänge des betrachteten Ausschnitts der realen Welt auf einer zumindest halbformalen Ebene und ist die Grundlage des zu erstellenden Lösungskonzeptes für das zu realisierende Anwendungssystem. Das realisierte Konzept wird somit Bestandteil des betrachteten Weltausschnittes. (untenstehender Abbildung)

Ein **Modell** ist immer eine vereinfachte Vorstellung davon, wie ein bestimmter Ausschnitt der realen Welt beschaffen ist. Ein **Konzept** ist eine vereinfachte Vorstellung davon, wie ein bestimmter Ausschnitt der realen Welt aussehen soll bzw. wie er aussieht, wenn diese Vorstellung verwirklicht wird.



Darstellung der Modellbildung

Modell: Etwas aus der realen Welt (Abbild).

Konzept: Etwas für die reale Welt (Vorbild).

Ein Konzept entsteht in der Regel auf Grundlage eines Modells.

Die Reale Welt besteht aus:

Dingen:

Dies sind alle konkreten oder abstrakten Objekte unserer Anschauung und unseres Denkens, wie z. B.

- ein Baum, ein Haus (real existierende),
- ein Drachen, ein Einhorn (in unserer Fantasie existierende), die *Freiheit, der Glauben (abstrakt).

Handlungen:

Unser Tun, um z. B.

- neue Dinge zu erschaffen,
- vorhandene Dinge zu zerstören

Eigenschaften:

Charakterisierung von Dingen durch

- Volumen, Gewicht
- lebendig, leblos

Beziehungen:

Zwischen Dingen

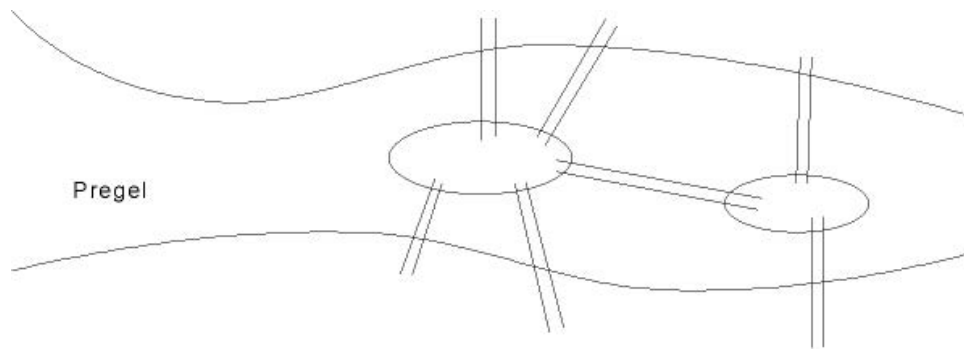
Die Modellierung des Ausschnitts der realen Welt in der Informatik erfolgt mit den folgenden Entsprechungen über:

Dinge	\leftrightarrow	Objekte
Handlungen	\leftrightarrow	Algorithmen (auf Objekte)
Eigenschaften	\leftrightarrow	Attribute (von Objekten)
Beziehungen	\leftrightarrow	Relationen (zwischen Objekten)

3.1.1 Graphen

Die Graphentheorie als Teilgebiet der Mathematik geht zurück auf Leonhard Euler (1707 - 1783). Er formulierte das Königsberger Brückenproblem (1736):

Gibt es einen Rundwanderweg über alle Brücken der Pregel (Fluss durch Königsberg), bei dem jede Brücke genau einmal überquert wird (untenstehende Abbildung) ?



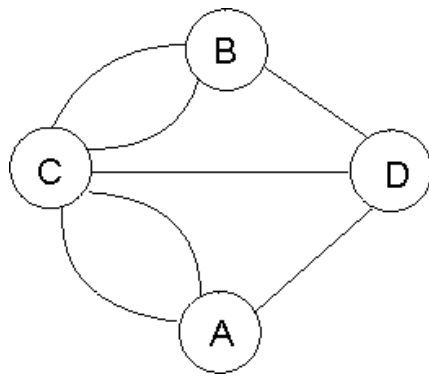
Das Königsberger Brückenproblem

Einen solchen Rundweg gibt es nicht. Diese Antwort lässt sich auf folgende Weise plausibel machen: Angenommen wird, dass es einen solchen Rundgang gibt.

1. Betrachtet wird zuerst ein Ort (Ufer), der nicht Start- und Endpunkt eines solchen Weges ist. Kommt man an diesem Ort über eine Brücke an, so muss man ihn über eine andere Brücke wieder verlassen. Man braucht also mindestens zwei Brücken.
2. Kommt man wiederholt über eine andere Brücke an dem Ort an, so muss der Ort jeweils über eine andere Brücke wieder verlassen werden. Bei einer ungeraden Anzahl von Brücken funktioniert dies also nicht.
3. Betrachtet wird nun noch ein Ort, an dem der Rundweg startet und endet. Über eine Brücke verlässt man den Ort am Beginn des Rundweges, und über eine andere Brücke kehrt man zurück. Man braucht also erneut mindestens zwei Brücken.
4. Kommt man bei diesem Ufer während des Rundganges (ggf. mehrfach) über eine Brücke an, so muss man auch dieses jeweils wieder über eine andere verlassen. Dies funktioniert nicht, wenn die Anzahl der Brücken ungerade ist.
5. Aus all dem Gesagten folgt: Wenn an mindesten einem Ufer eine ungerade Anzahl von Brücken existiert, gibt es einen solchen Rundwanderweg nicht.

Nach einem Blick auf die Karte ist die Antwort nun offensichtlich.

Formal lassen sich die Königsberger Brücken durch einen Graphen beschreiben. Seien A und B die beiden Uferseiten und C und D die beiden Inseln, dann lassen sich die Brücken durch Kanten darstellen (untenstehende Abbildung).



Das Königsberger
Brückenproblem als Graph

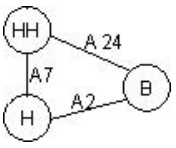
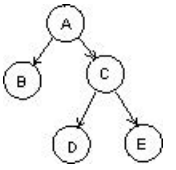
Ein **(gerichteter) Graph** $G=(V,E)$ ist eine endliche nicht leere Menge V zusammen mit einer Menge $E \subseteq \{(v_1, v_2) \mid v_1, v_2 \in V\}$ von Paaren von V . Jedes Element von V heißt Knoten, jedes Element von E heißt Kante. (untenstehende Abbildung)

Ein **Baum** ist ein gerichteter Graph ohne Zyklen mit einem ausgezeichneten Knoten, der **Wurzel**. Alle anderen Knoten haben jeweils genau einen Vorgänger. Knoten ohne Nachfolger heißen **Blätter**. (untenstehende Abbildung)

Gibt es in einem Graphen zu jedem $(v_1, v_2) \in E$ auch $(v_2, v_1) \in E$, so spricht man von einem **ungerichteten Graphen**. (untenstehende Abbildung)

Die grafische Darstellung eines Graphen erfolgt über benannte Kreise als Knoten und über Linien, jeweils zwischen zwei Knoten, als Kanten. Auch die Kanten können benannt sein. Bei gerichteten Graphen haben die Kanten Richtungsangaben durch Pfeile. Bei ungerichteten Graphen entfallen die Richtungsangaben an den Kanten. Zwei gegenläufige gerichtete Kanten zwischen zwei Knoten können u.U. zu einer ungerichteten Kante zusammengefasst werden.

a)	gerichteter Graph		$G = (V, E) \quad V = \{A, B, C, D\}$ $E = \{a, b, c, d\} = \{(A, B), (B, B), (C, B), (D, B)\}$
----	-------------------	--	----------------------------------------------------------------------------------------------------

b)	ungerichteter Graph		$G = (V, E)$ $V = \{HH, H, B\}$ $E = \{A7, A2, A24\}$ $= \{(HH, H), (H, HH), (H, B), (B, H), (HH, B), (B, HH)\}$
c)	Baum		$G = (V, E)$ $V = \{A, B, C, D, E\}$ $E = \{(A, B), (A, C), (C, D), (C, E)\}$



Beispiele zu Graphen

3.1.2 Petri-Netze

Überblick: Petri-Netze

Petri-Netze wurden in den sechziger und siebziger Jahren von Prof. Dr. Carl Adam Petri entwickelt. Sie stellen eine formale Sprache zur Modellierung und Ausführung von parallel laufenden Prozessen aus Technik und Wirtschaft dar. Die Prozesse können dabei gemeinsame Ressourcen verwenden, miteinander kommunizieren und hängen gegenseitig teilweise oder ganz voneinander ab (Nebenläufigkeit). Die Stärke der Petri-Netze liegt in der Möglichkeit der mathematischen Analyse der betrachteten Prozesse. Hierbei werden Methoden der linearen Algebra und der linearen Programmierung angewendet.

Im Laufe der Zeit entstanden verschiedene Dialekte, da erkannt wurde, dass sich ein und dasselbe Problem mit verschiedenen Netzformen lösen lässt.

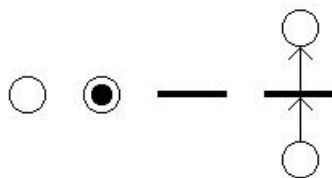
Während der Umsetzung von Petri-Netzen in die Praxis stellten sich zwei Haupthindernisse heraus. Erstens gab es keine Konzepte zur Behandlung von Daten, mit der Folge, dass die Netze groß und schwer überschaubar wurden, da jede Datenmanipulation direkt in der Netzstruktur abgebildet werden musste. Zweitens existierten keine Konzepte für eine hierarchische Gliederung, die es erlaubt hätten,

ein großes Modell aus einzelnen kleinen Modellen über definierte Schnittstellen aufzubauen.

Das erste Hindernis wurde in den späten siebziger Jahren mit den High-Level Petri-Netzen und das zweite in den späten achtziger Jahren mit den hierarchischen Petri-Netzen beseitigt. Der heutzutage bekannteste Dialekt sind die "Coloured Petri-Nets" (auch CP-Nets oder CPN genannt).

Mathematische Grundlagen

Zwei Mengen S und T mit einer Relation F bilden ein Petri-Netz, wenn S und T disjunkt sind, die Vereinigung von S und T nicht leer ist, durch F nur Elemente verschiedener Mengen verknüpft sind und kein Element von S und T isoliert ist. Man spricht bei den Elementen der Menge S von Stellen und bei Elementen der Menge T von Transitionen (Übergängen).

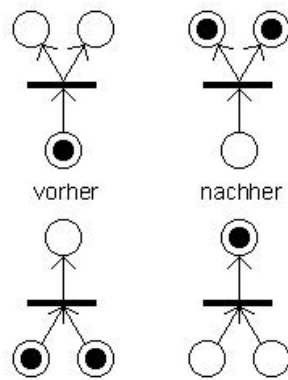


**Stellen und Transitionen
für Petri-Netze**

Die obenstehende Abbildung zeigt von links nach rechts eine Stelle, eine Stelle mit Marke, eine Transition, eine gerichtete Kante, die eine Stelle mit einer Transition verbindet und eine andere gerichtete Kante, die diese Transition wiederum mit einer weiteren Stelle verbindet.

Eine Stelle speichert eine Marke solange, bis sie über eine folgenden Transition abfließen kann. Dabei werden in ihrer ursprünglichen Interpretation Stellen als Bedingungen und nicht sperrende Transitionen als Ereignisse aufgefasst. Man spricht auch von Vor- und Nachbedingungen eines Ereignisses. Sind die Vorbedingungen erfüllt und die Nachbedingungen nicht erfüllt, schaltet bei Eintritt des Ereignisses die Transition, und die Nachbedingungen treten ein. Konkret heißt dies: Eine Transition sperrt nicht und lässt an allen Ausgängen (einem oder mehreren) Marken austreten, wenn an allen Eingängen (einem oder mehreren) jeweils eine Stelle mit einer Marke (Token) besetzt ist, also eine Marke bereit steht.

Die untenstehende Abbildung zeigt zwei einfache Anwendungen von Stellen und Transitionen, die das Prinzip verdeutlichen sollen:



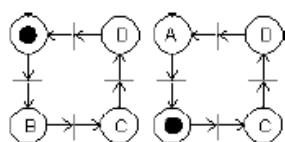
Beispiele für eine schaltende Transition mit drei Stellen

In der oberen Hälfte des Bildes ist dargestellt, wie eine Transition, die nur eine Stelle als Vorbedingung hat, beim Schalten ihre beiden Nachbedingungen erfüllt. Bitte beachten Sie, dass sich hierbei die Anzahl der Token um eins erhöht. Selbstverständlich sind auch noch mehr Nachbedingungen möglich. Demgegenüber zeigt die untere Hälfte des Bildes, wie eine Transition mit 2 Stellen als Vorbedingung schaltet, wenn die Vorbedingung erfüllt ist, d. h., wenn beide Stellen der Vorbedingung mit einem Token belegt sind. Bitte beachten Sie, dass eine Transition auch eine Vorbedingung mit noch mehr Stellen haben kann. Dann müssen ebenso alle Stellen der Vorbedingung mit Token belegt, also erfüllt sein, damit die Transition schaltet.

Beispiele: Petri-Netze

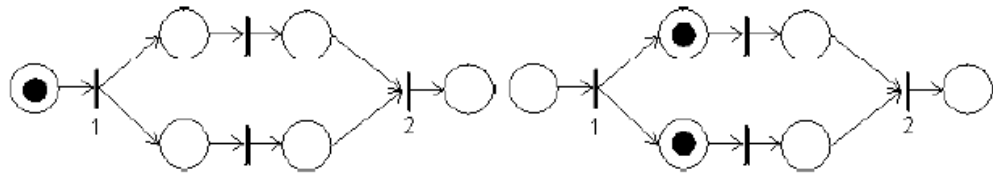
In den folgenden Abbildungen sind unter ein sequentieller, ein nebenläufiger und ein nebenläufiger Prozess mit wechselseitigem Ausschluß als Beispiel für Petri-Netze dargestellt:

a) sequentieller Prozess



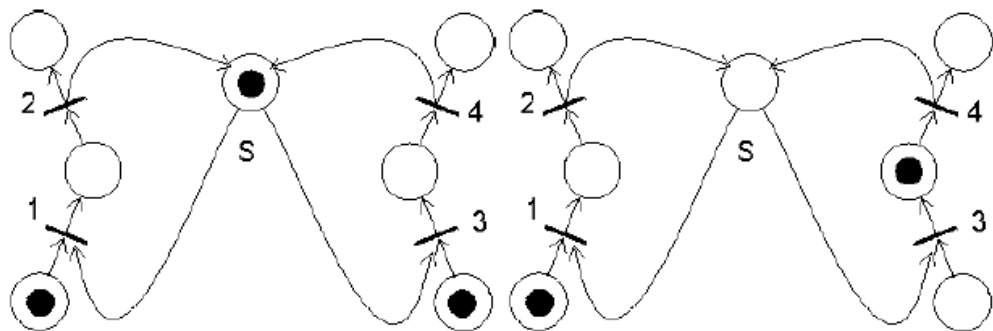
sequentieller Prozess

b) nebenläufiger Prozess



nebenläufiger Prozess

c) nebenläufiger Prozess mit wechselseitigem Ausschluss

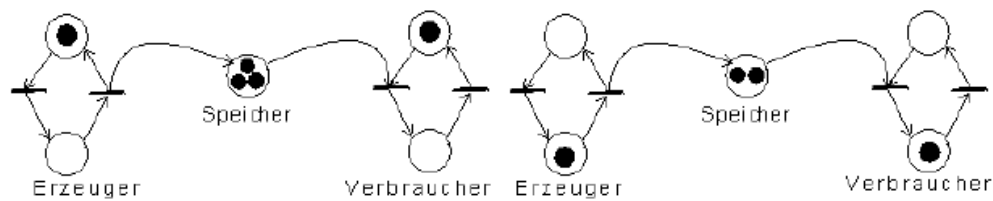


nebenläufiger Prozess mit wechselseitigem Ausschluß

Bei einem sequentiellen Prozess wird die Marke jeweils zu genau einer Stelle weitergegeben. Bei einem nebenläufigen Prozess gibt es u. a. zwei ausgezeichnete Transitionen, eine Verzweigung (Fork) und eine Vereinigung (Join). Eine Verzweigung erzeugt aus einer Marke je eine Marke auf den Eingangsstellen der nebenläufigen Prozesse. Die Vereinigung führt sie wieder zu einer Marke zusammen und ist somit die logische "UND"-Funktion.

Bei einem nebenläufigen Prozess kann es so genannte "kritische Abschnitte" geben, die nur jeweils exklusiv betrieben werden dürfen. Durch eine Verriegelungs-Transition (lock) und eine Entriegelungs-Transition (unlock) wird ein wechselseitiger Ausschluss der kritischen Abschnitte erzwungen.

Mit Petrinetzen lässt sich auch ein Erzeuger-Verbraucher-Prozess modellieren. Hier können die Stellen ausnahmsweise mehrere Marken aufnehmen. Beim Schalten einer Transition wird jeder Vorstelle eine Marke entnommen und jeder Nachstelle je eine Marke hinzugeführt. Voraussetzung für das Schalten einer Transition ist also, dass jede Vorstelle mindestens eine Marke enthält und jede Nachstelle noch eine Marke aufnehmen kann (untenstehende Abbildung).

**Erzeuger-Verbraucher-Prozess**

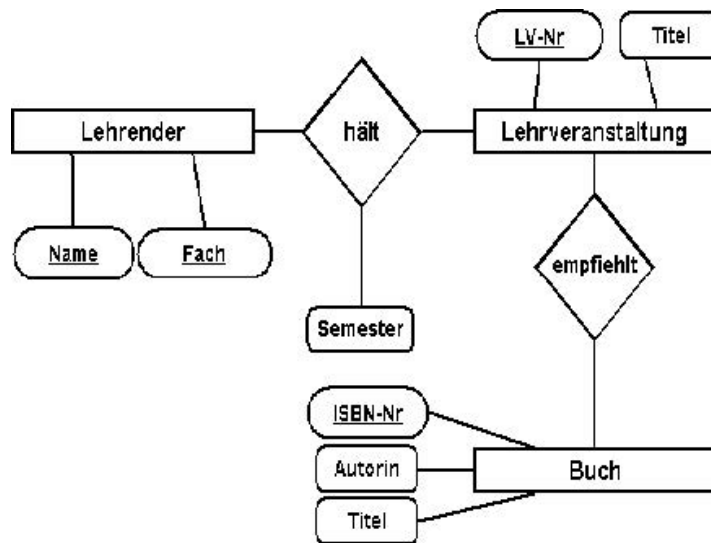
Im Internet finden sich unter der Adresse <http://www.daimi.au.dk/PetriNets/> weitere Informationen. Ein Editor/Simulator für Petri-Netze befindet sich unter der Adresse <http://OLLI.Informatik.Uni-Oldenburg.DE/PetriEdiSim/>.

3.1.3 Entity-Relationship-Modell (ER-Modell)

Das ER-Modell wurde von P. P. Chen im Jahre 1976 vorgestellt. Das Grundkonzept basiert auf:

- Entitäten (Entities) als zu modellierende Informationseinheiten bzw. Objekte,
- Beziehungen (Relationships) zur Modellierung von Beziehungen zwischen Entitäten und
- Attributen als Eigenschaften von Entitäten oder Beziehungen (Schlüsselattribute identifizieren eine Entität eindeutig und werden durch Unterstreichungen gekennzeichnet!)

In untenstehender Abbildung ist ein Beispiel grafisch dargestellt. Dabei sind folgende Darstellungselemente gewählt: Entitäten in Rechtecken, Beziehungen in Rauten und Attribute in Ovalen.



Beispiel für ein ER-Schema

Beziehungen können in einem ER-Modell in verschiedenen Kardinalitäten (Quantitäten) auftreten ($n, m \in \mathbb{N}$).

- 1 : 1 Beziehung



Jede Entitäten-Instanz steht mit höchstens einer anderen Entitäten-Instanz in Beziehung, z.B. Beziehungstyp "verheiratet" zwischen Personen.

- 1 : n Beziehung



Jede Entitäten-Instanz steht mit $n \geq 0$ (keiner oder mehreren) Entitäten-Instanzen in Beziehung, z.B. kann ein Lehrender mehrere Diplomanden betreuen.

- m : n Beziehung



Es gibt keine Restriktion zwischen Paaren von Entität-Typen, z.B. kann jeder Studierende mehrere ($n \geq 0$) Lehrveranstaltungen belegen; eine Lehrveranstaltung kann von mehreren ($m \geq 0$) Studierenden belegt werden.

3.1.4 Unified Modeling Language (UML)

Nach der Etablierung der objektorientierten Programmiersprachen kamen Anfang der 90er Jahre objektorientierte Analyse- und Entwurfsmethoden zur Modellierung der Struktur und der Abläufe von Prozessen auf. Auf der Grundlage dieser Modelle sind Systeme in der Entwicklung programmtechnisch zu realisieren. Die "Unified Modeling Language" (UML) wurde von Booch, Rumbaugh und Jacobson entwickelt und 1994 veröffentlicht.

UML ist eine Modellierungssprache aus überwiegend grafischen Elementen, die in verschiedenen Diagrammen verwendet werden. Es gibt einige diverse Diagrammtypen in UML. In UML 2.0 sind es dreizehn verschiedene. Hier werden die folgenden Diagrammtypen kurz erläutert:

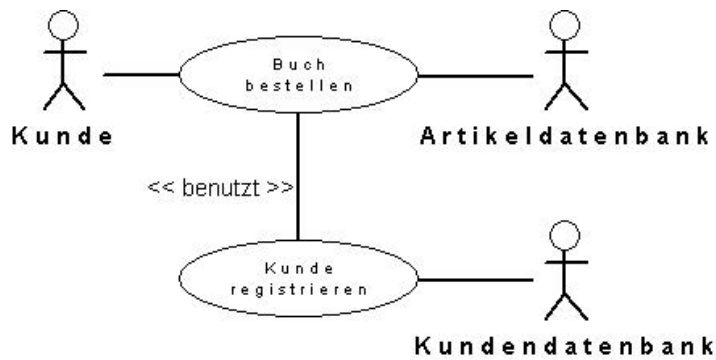
- Anwendungsfalldiagramme
- Klassendiagramme
- Zustandsdiagramme
- Sequenzdiagramme (Interaktionsdiagramme)

Je nach Fortschritt im Softwareentwicklungsprozess (siehe Unterkapitel Software-Entwicklungsprozess) finden die jeweils geeigneten Diagrammtypen Verwendung.

Anwendungsfalldiagramme

Ein Anwendungsfalldiagramm stellt die Beziehung zwischen Akteuren und Anwendungsfällen dar und somit das Systemverhalten aus der Sicht des Anwenders. Obwohl es keinen Designansatz darstellt, dient es dazu, die Zusammenhänge zwischen den einzelnen Elementen des Systems zu verdeutlichen und unterstützt den Entwickler bei der Anforderungsermittlung.

Ein Anwendungsfall beschreibt eine bestimmte Aktivität, durch die das System unterstützt wird. Er wird im Diagramm als eine Ellipse dargestellt, die den Namen des Anwendungsfalles enthält (z.B. "Kunde registrieren"). Akteure stellen die aktiven Teile dar, die in der Lage sind, Anwendungsfälle zu nutzen oder von diesen benachrichtigt werden können.



Beispiel eines Anwendungsfalldiagramms

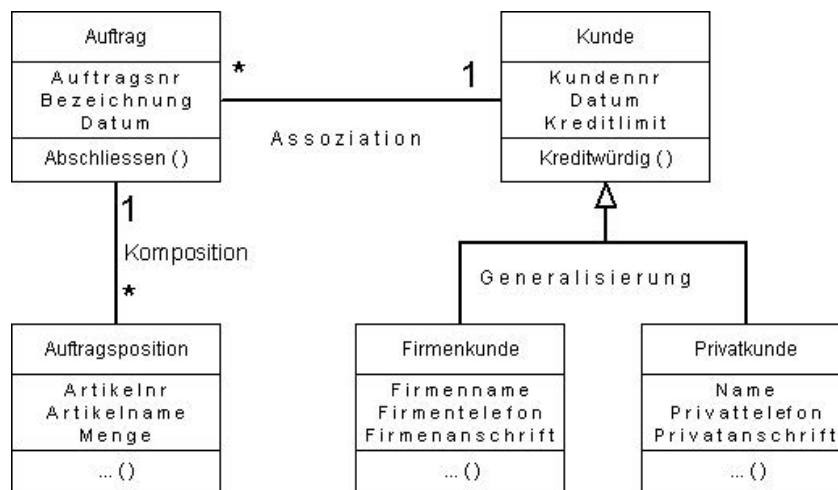
In obenstehender Abbildung gibt es drei Akteure (den Kunden und die beiden Datenbanken) und zwei Anwendungsfälle (Buch bestellen und Kunde registrieren). Gibt der Kunde seine Bestellung auf, wird der Anwendungsfall "Buch bestellen" aktiviert, der zunächst überprüft, ob der Kunde schon registriert ist. Ist dies nicht der Fall, aktiviert er den Anwendungsfall "Kunde registrieren", der Daten in die Kundendatenbank einträgt. Ist der Kunde dann registriert, kann die eigentliche Bestellung getätigt werden. Dazu bedient sich Buch bestellen der Artikeldatenbank. An diesem Beispiel erkennt man, dass ein Anwendungsfalldiagramm keine Rückschlüsse auf die Interna des Systems zulässt. So lässt sich z.B. nicht feststellen, in welcher Weise die Datenbanken manipuliert werden oder wie die Buchbestellung im Detail abläuft. Es vermittelt lediglich einen groben Überblick über das System.

Klassendiagramme

Ein Klassendiagramm stellt die Beziehungen zwischen den einzelnen Klassen (Objekten) des Systems dar. Es besteht aus einer Reihe von Klassen, die durch Assoziation bzw. Generalisierung miteinander verbunden sind. Dabei beschreiben die Assoziationen Beziehungen zwischen den Klassen (respektive Objekten); zum Beispiel: Kunde und Auftrag sind einander zugeordnet, oder einem Auftrag sind Auftragspositionen zugeordnet. Generalisierung ("ist-ein") ist das Gegenstück zur Diversifizierung, also das Gegenstück zur "erbt- von"-Beziehung; zum Beispiel: Firmenkunden und Privatkunden sind jeweils Kunden.

Eine Klasse wird als Rechteck dargestellt, das in drei Bereiche geteilt ist. Im oberen Teil steht der Name der Klasse. Im mittleren Bereich stehen die Attribute (Daten) der Klasse zusammen mit ihren Typen und im unteren Teil stehen schließlich die Methoden (Funktionen), welche die Klasse zur Verfügung stellt (untenstehende Abbildung).

Die Assoziationen und Generalisierungen werden durch Kanten dargestellt. Assoziationen sind immer mit Quantitäten anzugeben. Generalisierungen werden mit einer Richtung versehen.



Klassen und ihre Beziehungen

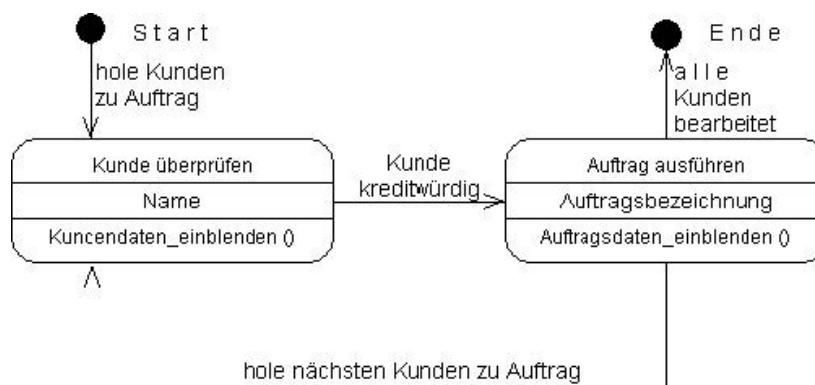
(*) steht für keine oder mehrere

In obigem Klassendiagramm erbt die Klasse Privatkunde die Attribute "Kundennummer", "Datum" und "Kreditlimit" sowie die Methode Kreditwürdig(); Die Klasse Privatkunde fügt zusätzlich die Attribute "Name", "Privattelefon" und "Privatanschrift" sowie weitere Methoden (angedeutet durch "...()") selbst hinzu.

Zustandsdiagramme

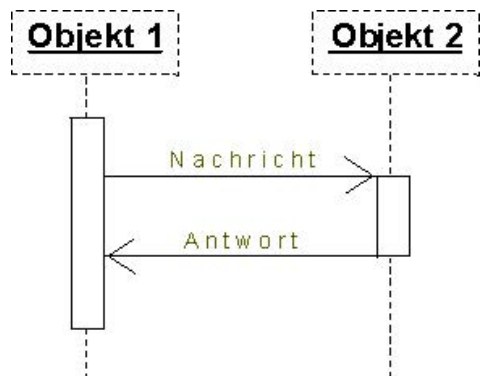
Zustandsdiagramme stellen einen Ablauf als eine Reihe von Zuständen und eine Reihe von Übergängen zwischen diesen Zuständen dar. Im Diagramm werden die Zustände als abgerundete Rechtecke dargestellt und können in bis zu drei Bereiche aufgeteilt sein. Im oberen Teil des Rechtecks steht der Zustandsname, im mittleren Teil stehen eventuell Zustandsvariable und im unteren stehen Aktionen, die ausgeführt werden, wenn ein bestimmtes Ereignis eintritt.

Der Übergang von einem zum nächsten Zustand erfolgt bei Eintritt eines Ereignisses oder Vorliegen einer Bedingung. Grafisch wird das durch eine gerichtete Kante dargestellt (untenstehende Abbildung).



**Beispiel eines Zustandsdiagramms****Sequenzdiagramme**

Sequenzdiagramme modellieren den Nachrichtenaustausch zwischen Objekten. Der zeitliche Ablauf steht dabei im Vordergrund. Im Gegensatz zu den übrigen Diagrammen spielt die Anordnung der Diagrammelemente hier eine Rolle: Die an einer Transaktion beteiligten Objekte werden nebeneinander dargestellt. Von diesen Objekten verlaufen gestrichelte "Lebenslinien" senkrecht nach unten. Sie symbolisieren den zeitlichen Ablauf. Wird die Lebenslinie von einem Balken überlagert bedeutet dies, dass das betreffende Objekt "aktiv" ist und eine Operation ausführt. Normalerweise wird ein Objekt durch einen Methodenaufruf aktiviert und wird nach Senden der Antwort wieder inaktiv. In untenstehender Abbildung ist ein Sequenzdiagramm skizziert.

**Beispiel
Sequenzdiagrammes**

eines

3.1.5 Aufgaben zum Thema: Konzepte, Modelle, Modellbildung

In diesem Unterkapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt. Es empfiehlt sich die Aufgaben selbstständig auf einem Blatt Papier zu lösen. Musterlösungen werden während des Semesters nach und nach freigeschaltet. Aus ähnlichen Aufgaben kann sich die Klausur zusammensetzen.

**Aufgabe****Aufgabe 3.1**

Es sei ein Graph $G = (V, E)$ mit

$$V = \{ H, I, J, K, L, M \}$$

und

$$E = \{ (H, I), (H, K), (I, L), (J, I), (J, M), (K, H), (L, K), (L, J), (M, J), (M, L) \}$$

gegeben.

Stellen Sie G graphisch dar.



Aufgabe

Aufgabe 3.2

Es ist das folgende Problem gegeben:

Ein Bauer möchte mit einem Fuchs, einer Gans und einem Korb voll Korn einen Fluss überqueren. Das Boot, das er dazu zur Verfügung hat, trägt aber nur ihn und entweder den Fuchs, die Gans oder das Korn. Hinzu kommt, dass der unbewachte Fuchs die Gans fressen wird und die unbewachte Gans das Korn fressen wird. Wie kann der Bauer übersetzen, ohne dass die Gans oder das Korn gefressen wird? Stellen Sie das Problem so als Graph dar, dass sich unmittelbar eine Lösung ablesen lässt. Hinweis: Verwenden Sie als Knoten die Zustände, die jeweils vor bzw. nach den Bootsfahrten auftreten können, z. B. (B F G K |) dafür, dass sich alle am linken Ufer des Flusses befinden, und (B G | F K) dafür, dass sich der Bauer mit der Gans am linken Ufer und der Fuchs und das Korn sich am rechten Ufer befinden.



Aufgabe

Aufgabe 3.3

Stellen Sie den in Aufgabe 2 entwickelten Graphen formal dar:



Aufgabe

Aufgabe 3.4

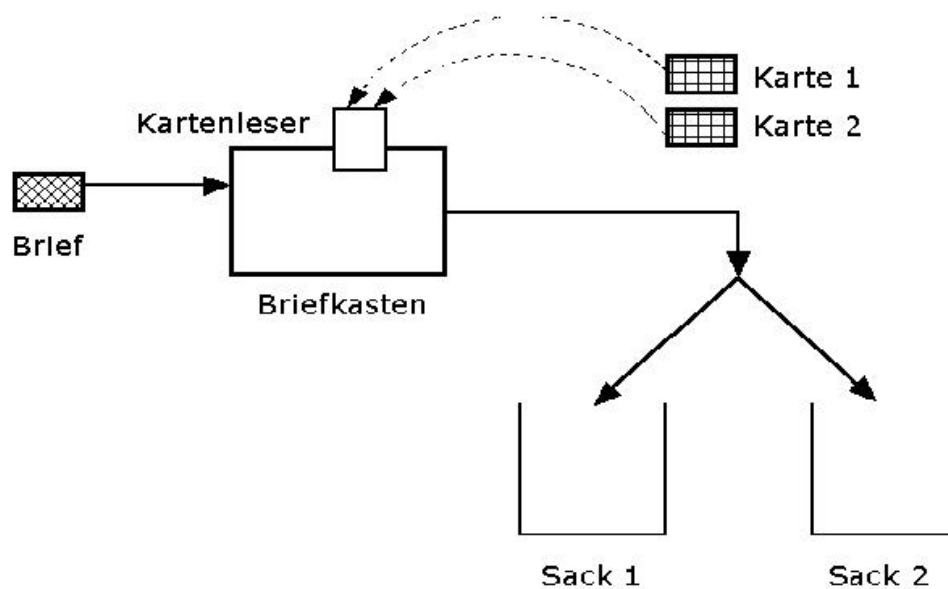
Die nachfolgend aufgeführten Aussagen aus dem Hochschulbereich sind in einem ER-Modell darzustellen:

Eine Hochschule ist gegliedert in mehrere Fachbereiche. Ein Fachbereich bietet mehrere Studiengänge an. Dozenten gehören zu jeweils einem Fachbereich. Dozenten halten Vorlesungen. Studierende schreiben sich für einen Studiengang ein. Studierende hören Vorlesungen.

Geben Sie für die Beziehungen die Quantitäten und für eine Entität beispielhaft einige Attribute an.

**Aufgabe****Aufgabe 3.5**

Gegeben sei eine einfache Briefannahmemaschine mit Vorsortierung. Am Briefkasten befindet sich ein Kartenleser. Es gibt zwei Arten von Karten und zwei Säcke in die die Post sortiert werden soll. Wird Karte 1 eingesteckt, soll der Brief in Sack 1 sortiert werden, wird Karte 2 eingesteckt soll der Brief in Sack 2 sortiert werden (s. Abbildung). Ein Brief kann erst eingesteckt werden, wenn sich eine Karte im Kartenleser befindet. Geben Sie ein Petri-Netz für diese Aufgabenstellung an.

**Briefsortiermaschine**

3.2 Simulation

In diesem Unterkapitel werden zunächst ein paar Betrachtungen zum intuitiven Simulationsbegriff angestellt. Dann wird der Begriff der Simulation, wie er in der angewandten Informatik verwendet wird, ausführlich erklärt und darauf hingewiesen, dass in der Praxis die Sichtweisen von Menschen eine Rolle spielen. Abschließend wird der Simulationsbegriff aus der theoretischen Informatik erwähnt.

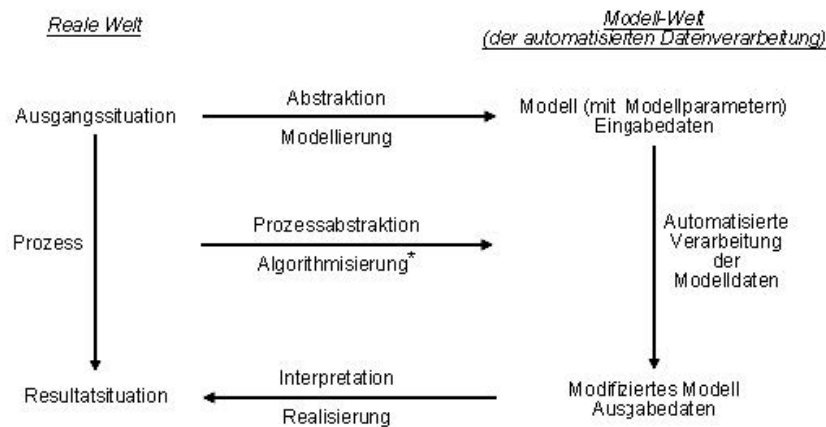
Der intuitive Simulationsbegriff

Beginnen wir bei der Betrachtung des Begriffes "Simulation" hier mit dem intuitiven Verständnis, dass auf einem Computersystem mit geeigneter Software Vorgänge aus der realen Welt nachgebildet werden. Dann ist in unserer heutigen Welt das Vorhandensein von Simulationen bereits eine alltägliche Situation, auch wenn uns dies oft nicht bewusst ist. Wetterberichte mit Wolkenanimationen in den großen Nachrichtensendungen im Fernsehen simulieren das Erscheinungsbild der bewegten Wolken. In Computersystemen ablaufende Crashtests helfen bei der Fahrzeugentwicklung. Das Erkennen von Alarmsituationen in Prozessen, z.B. bei der Fondsverwaltung im Aktienmarkt, erlaubt es, zeitgenau in diese Prozesse steuernd einzugreifen. Die Erprobung von riskanten Operationen am aus Tomographiedaten erstellten 3D-Modell des menschlichen Körpers hilft, Patienten vor Risiken zu bewahren. Und nicht zuletzt haben technisch anspruchsvolle Computerspiele in die menschlichen Lebenswelten Eingang gefunden.

Bei den genannten Beispielen benutzen wir im intuitiven (umgangssprachlichen Sinne) den Begriff der Simulation als Beschreibung dessen, was im Computersystem abläuft. Zum Beispiel: Der Computer "simuliert" den Crashtest, den zweiten Schachspieler, etc. Dies ist in der Betrachtungsweise aus Sicht der Informatik anders:

Der Simulationsbegriff der Angewandten Informatik

In allen eben genannten Beispielen von Angewandter Informatik spielt der Begriff der Simulation eine wichtige Rolle. Dieser Begriff spielt in der Informatik eine zentrale Rolle und ist klar definiert. Anders als bei der intuitiven Verwendung des Begriffes in der Umgangssprache wird in der Informatik der gesamte Vorgang, von der Modellbildung aus der realen Welt über die Verarbeitung der Modelldaten bis hin zur Interpretation der Verarbeitungsergebnisse im Hinblick auf die reale Welt als Simulation bezeichnet. Diese Idee wird in der untenstehenden Abbildung veranschaulicht:



Der Simulationsbegriff der Angewandten Informatik

(*)Algorithmisierung kann hier als Erstellen einer Sammlung von durch die automatisierte Datenverarbeitung ausführbaren Schritten verstanden werden. Der Begriff wird im Kapitel '[[Algorithmen und Datenstrukturen]]' näher erläutert.

Obenstehende Abbildung zeigt das Neben- oder besser: Miteinander von Vorgängen in der realen Welt und in der Welt der automatisierten Datenverarbeitung. Bekannt ist dabei, dass sich in der realen Welt eine Ausgangssituation durch einen gegebenen (physikalischen, wirtschaftlichen, chemischen, mechanischen, etc.) Prozess verändert und in eine möglicherweise nur temporäre Ausgangssituation mündet. Wenn ein solcher Vorgang nun simuliert werden soll, sind dafür mehrere Aspekte wichtig:

- Abstraktion, Modellierung

"Abstraktion" bedeutet das Weglassen von Unwesentlichem. Gemeint ist das Weglassen von Informationen aus der realen Welt, die keinen Einfluss auf den betrachteten Prozess haben. Mit Hilfe der Modellierung wird dann eine Repräsentationsform der Ausgangssituation erzeugt, das Modell, das letztendlich der automatisierten Datenverarbeitung zugeführt wird.

- Modell (mit Modellparametern) / Eingabedaten

Das Modell ist eine Menge von Daten, die eine Situation der realen Welt beschreiben. Allerdings sind diese Daten durch eine automatisierte Datenverarbeitung lesbar (und oft auch veränderbar). Das Besondere an einem Modell sind in der automatisierten Datenverarbeitung oft die Modellparameter. Sie sind variable Daten des Modells, die zwar mit bestimmten Werten fest angesetzt werden, aber bei einer weiteren Datenverarbeitung durchaus anders angesetzt werden können (und auch werden). Ein Programm, das den nächsten Zug beim Schach berechnen kann, kann dies also mit verschiedenen Ausgangsstellungen tun, bei einem Crashtest können z. B. Geschwindigkeit und Aufprallwinkel modifiziert werden, etc. Dies hat besonderen Einfluss auf den (u. a. wirtschaftlichen) Nutzen von Simulationen.

Aus Sicht dieser automatisierten Datenverarbeitung handelt es sich bei einem Modell mit seinen Modellparametern jedoch einfach um die Eingabedaten.

- Prozessabstraktion, Algorithmisierung

Erneut ist das Weglassen von Unwesentlichem, bzw. die Fokussierung auf das Wesentliche, ein wichtiger Vorgang, um aus einem Prozess in der realen Welt eine Verarbeitungsvorschrift für die automatisierte Datenverarbeitung zu machen. Wenn das Wesentliche bekannt ist, wird durch Algorithmisierung, also die Beschreibung von durch die automatisierte Datenverarbeitung ausführbaren Schritten, eine Repräsentation des Prozesses aus der realen Welt in der Welt der automatisierten Datenverarbeitung erzeugt.

- Automatisierte Verarbeitung der Modelldaten

Damit ist hier gemeint, dass ein Computersystem mit geeigneter Software die Eingabedaten liest und Ausgabedaten bereitstellt.

- Modifiziertes Modell / Ausgabedaten

Nach der automatisierten Datenverarbeitung liegen die Ausgabedaten vor. Wenn die Eingabedaten ein Modell einer realen Weltsituation waren, so repräsentieren die Ausgabedaten meist ein modifiziertes Modell dieser Situation.

- Interpretation, Realisierung

Wenn es gelingt, das modifizierte Modell als Repräsentation einer realen Weltsituation zu interpretieren, oder wenn man aufgrund der Modelldaten durch Realisierung eine reale Weltsituation schaffen (bauen, erstellen) kann, so schließt sich das vorliegende Diagramm. Erst dann wird in der Informatik von einer Simulation gesprochen.

- Zusammenfassung:

Der gesamte Zyklus, bei dem aus einer Ausgangssituation in der realen Welt durch Abstraktion bzw. Modellierung ein Modell für die automatisierte Datenverarbeitung gewonnen wird, dieses Modell anschließend automatisiert verarbeitet wird, wobei ein modifiziertes Modell entsteht, dass dann als Resultatsituation in der realen Welt interpretiert wird, oder auf dessen Grundlage eine Resultatsituation in der realen Welt erstellt wird, wird in der angewandten Informatik als Simulation bezeichnet. Dies gilt nur dann, wenn es einen Prozess in der realen Welt gibt (oder gäbe), welcher die Ausgangssituation in die Resultatsituation überführt.

Der Nutzen von Simulationen liegt oft in Folgendem:

- Kosteneinsparungen

Die Durchführung einer oder mehrerer Simulationen in der automatisierten Datenverarbeitung erzeugt geringere Kosten (im Sinne von Geld, Zeit, oder anderen knappen Ressourcen), als der Ablauf der korrespondierenden Prozesse in der realen Welt erfordern würde. Allerdings müssen die teilweise nicht unerheblichen Kosten bei der Modellierung, Algorithmisierung oder Interpretation berücksichtigt werden.

- Vereinfachung bzw. Ermöglichung

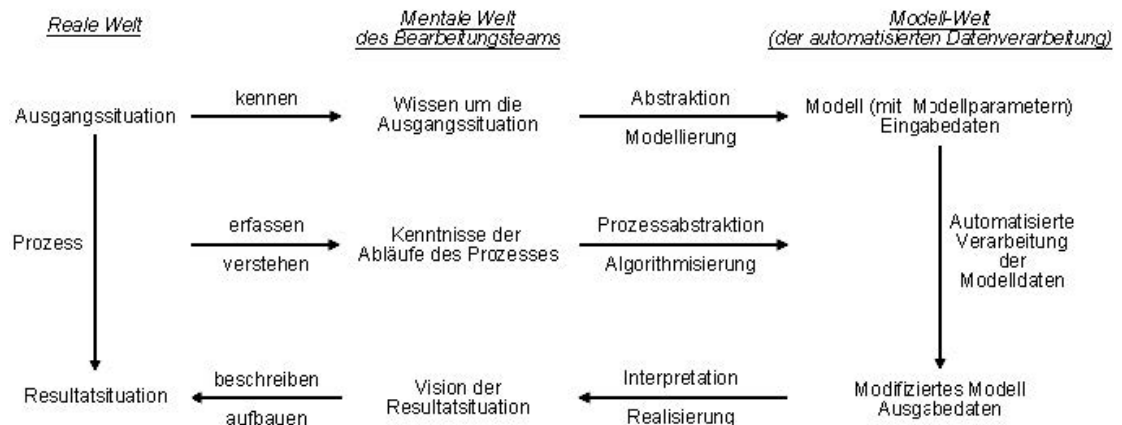
Durch Veränderung der Modellparameter beim Modell der Ausgangssituation können in der Simulation Variationen der Ausgangssituation verarbeitet werden, die in der realen Welt nur schwer oder gar nicht erzeugt werden können. Zum Beispiel: "Wie verändert sich das Erdklima, wenn sich der CO₂-Gehalt der Atmosphäre schnell (oder langsam) verringert?"

- Abwenden von Schaden

Durch Simulation lassen sich Abläufe ausprobieren, bei der in der realen Welt ein Schaden an Material, Gesundheit oder Leben entstehen würde, ohne dass dieser Schaden in der Realität eintritt.

Diese Auflistung erhebt nicht den Anspruch auf Vollständigkeit.

Abschließend sei hier noch erwähnt, dass in der obenstehenden Abbildung ein Idealzustand dargestellt wird. Dabei wird außer Acht gelassen, dass in der Praxis die Vorgänge Abstraktion, Modellierung, Prozessabstraktion, Algorithmisierung, Interpretation und Realisierung von Menschen, z. B. Fachgebietsexperten, Analysten, Softwareingenieuren, Programmierern oder anderen vorgenommen werden, deren Vorgehen und Entscheidungen von Ihren Informationen über bzw. Sichtweisen auf die reale Welt bestimmt werden. Wenn man dies ebenfalls darstellen will, könnte der Sachverhalt folgendermaßen aussehen (ohne Anspruch auf Korrektheit bzw. Vollständigkeit; Anmerkung des Autors):



Der Simulationsbegriff der angewandten Informatik in der Praxis

Aus der obenstehenden Abbildung wird ersichtlich, dass bei der Modellierung der Ausgangssituation und Algorithmisierung eines Prozesses menschliche Arbeit eine Rolle spielt. Außerdem kann unter anderem das mentale Abbild der Realität fehlerhaft oder unvollständig sein, so dass die Simulation nicht der Realität entspricht. Zu einer in der Praxis erfolgreichen Simulation gehört daher immer (so dies möglich ist) ein prüfender Abgleich der Simulationsergebnisse mit der betrachteten Situation in der realen Welt. Unabhängig davon, ob ein solcher Abgleich stattgefunden hat, spricht man beim Vorgang des Erstellens einer Simulation mit Hilfe der automatisierten Datenverarbeitung in Computern von einem Software-Entwicklungsprozess. Dieser schließt meist auch die Modellierung mit ein und wird auf der Seite Software-Entwicklungsprozess weiter erläutert.

Der Simulationsbegriff der theoretischen Informatik

Es sei hier nur kurz erwähnt, dass in der theoretischen Informatik ein Simulationsbegriff verwendet wird, bei dem auf der Seite der automatisierten Datenverarbeitung ein bestimmtes formales Verfahren zugrunde gelegt wird. Darauf wird hier jedoch nicht weiter eingegangen.

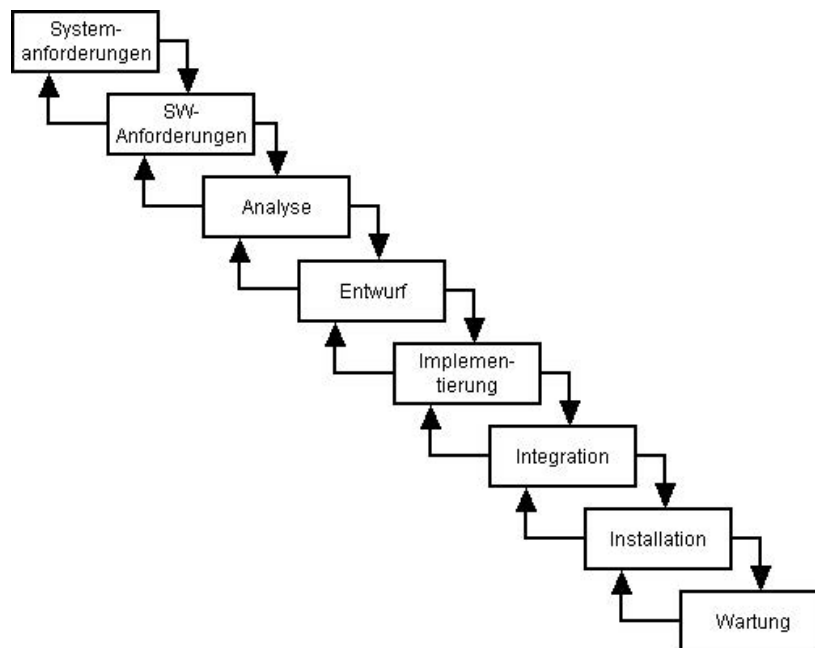
3.3 Software - Entwicklungsprozess

Im Software-Entwicklungsprozess stellt das reine Erstellen des Programmtextes nur einen Schritt unter diversen anderen dar. Es sind einige Schritte vor- und andere nachgelagert. Der gesamte Entwicklungsprozess wird in Vorgehensmodellen beschrieben. Es wird dabei der gesamte Prozess in Teilaufgaben zergliedert und diese

dann in einen logischen und chronologischen Zusammenhang gebracht. Im Folgenden werden einige Vorgehensmodelle beschrieben.

Das Wasserfallmodell

Die Teilaufgaben werden in eine sequentielle Reihenfolge gebracht, welche chronologisch abzuarbeiten sind (untenstehende Abbildung).



Das Wasserfallmodell

Charakteristika:

- Einfach und verständlich
- Top-Down-Vorgehen
- Der Entwicklungsvorgang ist sequentiell
- Jeder Entwicklungsschritt ist in der richtigen Reihenfolge vollständig durchzuführen
- Dokumentgetriebener Ansatz, d.h. am Ende jeder Phase steht ein fertiggestelltes Dokument

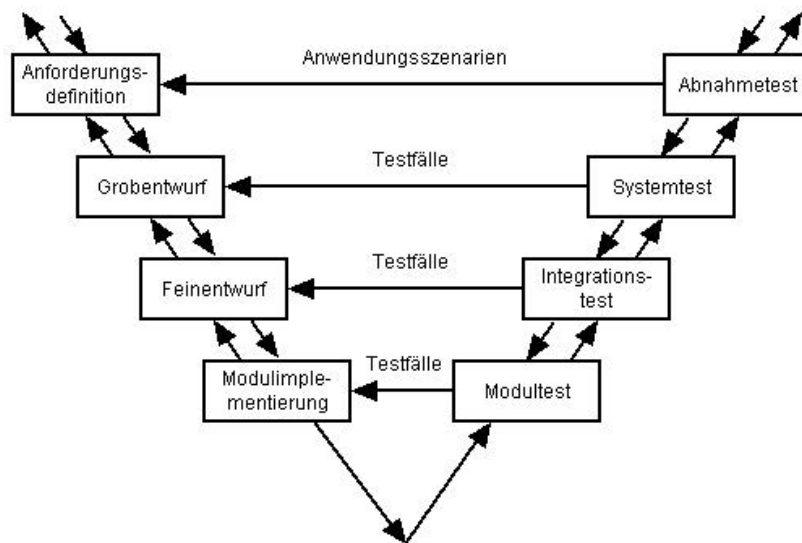
Nachteile:

- Nicht immer sinnvoll, alle Entwicklungsschritte vollständig und sequentiell durchzuführen
- Gefahr, daß die Dokumentation wichtiger wird als das eigentliche System
- Das Ergebnis kann sich immer mehr von der Systemanforderung entfernen, keine globale Rückkopplung

Das V-Modell

Im V-Modell werden die Systemanforderungen zunächst top-down bis zur Implementierung verfeinert und diese Verfeinerungsstufen dann bottom-up bis zum Abnahmetest Testszenarien gegenübergestellt, so dass auf jeder Stufe angemessene Testfälle betrachtet werden können. Auf

diese Weise entstehen Rückkopplungen zu Teilaufgaben, die chronologisch und in der Sequenz weiter entfernt liegen (untenstehende Abbildung).



Das V-Modell

Charakteristika:

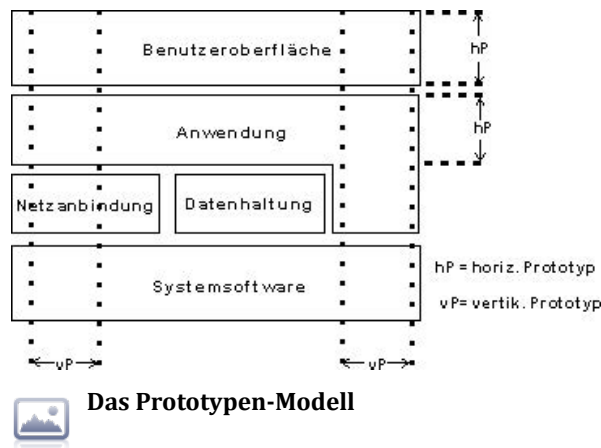
- Integrierte, detaillierte Darstellung zur Software-Erstellung
- Allgemeines Vorgehensmodell mit Möglichkeiten zum "Maßschneidern" auf projektspezifische Anforderung
- Ermöglicht eine standardisierte Abwicklung von Software-Projekten
- Gut geeignet für große Projekte
- Rückkopplung über mehrere Stufen, Qualitätssicherung

Nachteile:

- Durch Trennung in Funktions- und Datensicht ist keine Objektorientierung möglich
- Für kleine und mittlere SW-Entwicklungen zu aufwendig
- Nur mit Rechnerunterstützung handhabbar

Das Prototypen-Modell

Beim Prototypen-Modell (rapid prototyping) wird das Gesamtsystem funktional gegliedert. Jede Funktion wird zunächst nur in Teilen (ggf. auch nur als leerer Rahmen) erstellt, so dass das Gesamtsystem jedoch stets lauffähig ist. Ein solcher Prototyp wird dann zyklisch nach Rückkopplung mit den Auftraggebern bzw. mit den späteren Anwendern vervollständigt. (untenstehende Abbildung)



Das Prototypen-Modell

Charakteristika:

- Reduzierung des Entwicklungsrisikos durch frühzeitigen Einsatz von Prototypen
- In andere Prozessmodelle integrierbar
- Durch geeignete Werkzeuge schnell erstellbar
- Fördert die Kreativität der Entwickler, Lösungsalternativen zu finden
- Starke Rückkopplung mit dem Endbenutzer

Nachteile:

- Höherer Entwicklungsaufwand
- Gefahr, dass ein Prototyp aus Termingründen doch Teil des Endprodukts wird
- Prototypen werden oft als Ersatz für die fehlende Dokumentation angesehen

3.4 Fragen mit Musterlösungen: Modellierung

In diesem Kapitel sind Verständnisfragen mit ihren Musterantworten zu dem vorangehenden Lehrstoff zusammengestellt. Solche Fragen werden auch in der Klausur vorkommen.

Im Folgenden sind die Verweise zu den Fragen aufgelistet. Mit einem Klick auf 'Antwort' am Ende der Fragenseite bekommt man die Antwort für das gestellte Problem angezeigt. Ein Klick auf "Info-Basis ..." führt zu dem Kapitel des für die Aufgabe relevanten Lehrstoffes.

Das Textfeld dient z.Z. lediglich dazu, eigene Antworten zu notieren, um sie besser mit der Musterantwort vergleichen zu können. Eine Speichermöglichkeit des Textes ist damit jedoch nicht gegeben.

**Aufgabe****Frage 3.1**

Erläutern Sie die Begriffe "Konzept" und "Modell"!

Antwort

Ein Konzept ist eine vereinfachte Vorstellung darüber, wie ein bestimmter Ausschnitt der realen Welt aussehen soll bzw. wie er aussieht, wenn diese Vorstellung verwirklicht wird.

Ein Modell ist immer eine vereinfachte Vorstellung darüber, wie ein bestimmter Ausschnitt der realen Welt tatsächlich beschaffen ist.

Vgl. Kapitel Konzepte, Modelle, Modellbildung

**Aufgabe****Frage 3.2**

Was ist ein Graph?

Antwort

Ein (gerichteter) Graph $G = (V, E)$ ist eine endliche nicht leere Menge V , zusammen mit einer Menge $E = \{ (v_1, v_2) \mid v_1, v_2 \text{ Element von } V \}$ von Paaren von V . Jedes Element von V heißt Knoten, jedes Element von E heißt Kante.

Vgl. Kap. Graphen



Aufgabe

Frage 3.3

Was ist ein Baum?

Antwort

Ein Baum ist ein gerichteter Graph ohne Zyklen mit einem ausgezeichneten Knoten, der Wurzel. Alle anderen Knoten haben jeweils genau einen Vorgänger. Knoten ohne Nachfolger heißen Blätter.

Vgl. Kap. Graphen



Aufgabe

Frage 3.4

Wie wird ein Graph grafisch dargestellt?

Antwort

Die grafische Darstellung eines Graphen erfolgt über benannte Kreise als Knoten und über Linien, jeweils zwischen zwei Knoten, als Kanten. Auch die Kanten können benannt sein. Bei gerichteten Graphen haben die Kanten Richtungsangaben durch Pfeile. Bei ungerichteten Graphen entfallen die Richtungsangaben an den Kanten. Zwei gegenläufige gerichtete Kanten zwischen zwei Knoten können u.U. zu einer ungerichteten Kante zusammengefasst werden.

Vgl. Kap. Graphen



Aufgabe

Frage 3.5

Erläutern Sie das Entity-Relationship-Modell!

Antwort

Das Grundkonzept basiert auf:

Entitäten (Entities) ...als zu modellierende Informationseinheiten bzw. Objekte

Beziehungen (Relationships) ...zur Modellierung von Beziehungen zwischen Entitäten

Attributen ...als Eigenschaften von Entitäten oder Beziehungen (Schlüsselattribute sind besonders zu kennzeichnen!)

Vgl. Kap. Petri-Netze



Aufgabe

Frage 3.6

Welche Diagramme stellt die Unified Modeling Language zur Verfügung und was bedeuten sie?

Antwort

Anwendungsfalldiagramm: Es stellt die Beziehung zwischen Akteuren und Anwendungsfällen, d.h. das Systemverhalten aus der Sicht des Anwenders, dar.

Klassendiagramm: Es stellt die Beziehungen zwischen den einzelnen Klassen des Systems dar. Es besteht aus einer Reihe von Klassen, die durch Assoziationen (bzw. Generalisierungen) miteinander verbunden sind.

Zustandsdiagramm: Es stellt einen Ablauf als eine Reihe von Zuständen und eine Reihe von Übergängen zwischen diesen Zuständen dar.

Sequenzdiagramm: Es modelliert den Nachrichtenaustausch zwischen Objekten. Der zeitliche Ablauf steht dabei im Vordergrund.

Vgl. Kap. Entity-Relationship-Modell (ER-Modell)



Aufgabe

Frage 3.7

Was ist ein Algorithmus?

Antwort

Ein Algorithmus ist ein allgemeines (eindeutiges) Verfahren zur Lösung einer Klasse gleichartiger Probleme, gegeben durch einen aus endlich vielen elementaren Anweisungen bestehenden Text.

Vgl. Kap. Simulation

**Aufgabe****Frage 3.8**

Was wird unter Effektivität eines Algorithmus, was unter Effizienz eines Algorithmus verstanden?

Antwort

Effektivität: Der Algorithmus ist ausführbar.

Effizienz: Aufwand eines Algorithmus, bezogen auf die benötigte Rechenzeit bzw. den benötigten Speicherplatz.

Vgl. Kap. Simulation

**Aufgabe****Frage 3.9**

Welche grundlegenden Strukturierungen sind beim Entwurf von Algorithmen möglich?

Antwort

Top-down: Das Problem wird (hierarchisch) in Teilprobleme zerlegt bis die einzelnen Probleme durch elementare Anweisungen lösbar sind. Die Lösung jedes Teilproblems trägt zur Lösung des Gesamtproblems bei.

Bottom-up: Ausgehend von einer konkreten Rechenmaschine wird durch sukzessives Hinzufügen bestimmter Funktionalitäten und Eigenschaften eine abstrakte Maschine entwickelt, bis man bei der Maschine anlangt, die das gewünschte Problem löst.

Vgl. Kap. Simulation



Aufgabe

Frage 3.10

Nennen Sie einige Vorgehensmodelle des Software-Entwicklungsprozesses!

Antwort

- Wasserfallmodell
- V-Modell
- Prototypen-Modell

Vgl. Kap. Software - Entwicklungsprozess

4 Information und Nachricht



Gliederung

4 Information und Nachricht

4.1 Informelle Einführung der Begriffe

4.2 Digitale Nachrichten, Codes

4.3 Nachrichten- und Informationsverarbeitung

4.4 Aufgaben zum Thema: Information und Nachricht

4.5 Fragen mit Musterlösungen: Information und Nachricht

In dem hier verwendeten Kontext ist es zulässig und angemessen, für den Begriff Nachricht den Begriff Daten einzusetzen. Die Daten sind die Objekte, die von einem Rechner mittels eines Algorithmus verarbeitet werden. Die Daten werden von Menschen interpretiert und werden so zu Informationen.

4.1 Informelle Einführung der Begriffe

Die (abstrakte) Information wird durch die (konkrete) Nachricht mitgeteilt. Beziehungen zwischen Information und Nachricht:

- Dieselbe Information kann durch verschiedene Nachrichten mitgeteilt werden, z.B. Nachrichten in verschiedenen Sprachen.
- Es gibt Nachrichten, die keine Information enthalten.
- Es gibt Nachrichten, die nur für einen bestimmten Personenkreis eine Information enthalten.
- Nachrichten können, verschieden interpretiert, verschiedene Informationen ergeben.

Also:	Zuordnung zwischen Information und Nachricht ist nicht eindeutig
Deshalb:	Definition einer Interpretationsvorschrift α zwischen einer Nachricht N und einer Information I , die für Absender und Empfänger gleich ist.
	$N \xrightarrow{\alpha} I$

Eine Interpretationsvorschrift gilt natürlich nicht nur für eine einzelne Nachricht, sondern für eine Menge nach einheitlichen Gesetzen aufgebauter Nachrichten.



Beispiel

Formulieren wir Nachrichten z.B. in deutscher Sprache und verstehen wir deutsch (kennen die Interpretationsvorschrift), so erkennen wir in der Nachricht die Information:

Nachricht		Information
es regnet	(il pleut)	Wassertropfen fallen auf die Erde
es schneit	(il neige)	Eiskristalle fallen auf die Erde

Nachrichtengeräte und -übertragung

- Nachrichtengeräte (Nachrichtenübertragungsgerät) bestehen aus
 - Sender (Effektor)
 - Empfänger (Rezeptor)
 - Verstärker (Relais), gleicher Nachrichtenträger am Eingang und am Ausgang
 - Wandler, verschiedene Nachrichtenträger am Eingang und am Ausgang
- Nachrichtenübertragung erfolgt durch dauerhafte und nicht dauerhafte Nachrichtenträger.

Nicht dauerhafte Nachrichtenträger:

- - Druckwellen, z.B. Schallwellen
 - Elektrische Spannungen und Ströme
 - Elektromagnetische Wellen, z. B. Licht
 - weitere ...

Dauerhafte Nachrichtenträger:

- - Papier, Zeitungen, Bücher, ...
 - Lochkarten, Lochstreifen
 - magnetisierbare Schichten
 - lichtempfindliche Schichten
 - weitere ...

- (Übertragungs-) Kanal: Weg vom Sender zum Empfänger, der durch einen Nachrichtenträger überbrückt wird.

4.2 Digitale Nachrichten, Codes

Die Darstellung und Übertragung von Nachrichten erfolgt auf der Basis veränderlicher physikalischer Größen

Analoge Nachrichten:

Abbildung in einen kontinuierlichen Wertebereich, z.B. gesprochene Sprache

Digitale Nachrichten: Abbildung in einen diskreten Wertebereich z.B. geschriebene Sprache

Zeichenvorrat: Menge aus endlich vielen Zeichen



Beispiel

{0,1,2, ..., 9}

{a,b,c, ... , z}

{0,1}

{0,L}

{gelocht, ungelocht}

{0 V, 5 V}

{wahr, falsch}

Code:	Vorschrift zur Zuordnung eines Zeichenvorrats zu einem anderen Zeichenvorrat
	Zuordnung wird Codierung genannt
	Zuordnung ist meist eindeutig und eindeutig umkehrbar (→ Decodierung)

In untenstehender Tabelle sind diverse Zifferncodes aufgeführt.

Ziffern- symbol	direkt	Gray	Exzeß-3 (Stibitz)	Gray- Stibitz	Aiken	biquinär	1 aus -10	2 aus -5	CCIT-2
0	0000	0000	0011	0010	0000	000001	000000	000100	01101
1	0001	0001	0100	0110	0001	000010	000000	000101	11101
2	0010	0011	0101	0111	0010	000100	000000	000100	11001
3	0011	0010	0110	0101	0011	001000	000000	100010	10000
4	0100	0110	0111	0100	0100	010000	000001	100001	01010
5	0101	0111	1000	1100	1011	100001	000010	100010	00001
6	0110	0101	1001	1101	1100	100010	000100	100100	10101
7	0111	0100	1010	1111	1101	100100	001000	100001	11100
8	1000	1100	1011	1110	1110	101000	010000	100010	01100
9	1001	1101	1100	1010	1111	110000	100000	100100	00011
Stellenwert	8421	15731			2421	543210	9876543210		



Codes für die Ziffern

In untenstehender Tabelle ist der ISO 7-Bit-Code (ASCII-Code mit einem 8. Bit als Prüfbit) aufgeführt.

Dabei werden die Bits aus der Kopfzeile den Bits aus der linken Spalte vorangestellt. Damit waren 128 Zeichen codierbar. Heute wird das 8. Bit nicht mehr als Prüfbit, sondern zur Erweiterung des Zeichenvorrates benutzt. Es können also $2^8 = 256$ Zeichen codiert werden (= erweiterter ASCII-Code).

	000	001	010	011	100	101	110	111
0000	NUL	DLE		0	@	P		p
0001	SOH	DC	!	1	A	Q	a	q
0010	STX	DC	"	2	B	R	b	r
0011	ETX	DC	1.	3	C	S	c	s
0100	EOT	DC	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v

0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EN)	9	I	Y	i	y
1010	LF	SUB	•		J	Z	j	z
1011	VT	ESC	+	•	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL



Zeichenvorrat des ISO-7-Bit-Code

Das Wort "Hallo" in verschiedenen Darstellungsformen:

in Buchstaben:	H	a	l	l	o
in Form des erweiterten ASCII-Codes	01001000	01100001	01101100	01101100	01101111
als Dezimalzahlen	072	097	108	108	111
als Hexadezimalzahlen	48	61	6C	6C	6F

Je 4 Bits des ASCII-Codes bilden dabei eine hexadezimale Ziffer, wobei 0000, 0001, ..., 1001 auf die dezimalen Ziffern 0, 1, ..., 9 abgebildet werden und die restlichen Bitfolgen in Buchstaben:

1010	A
1011	B
1100	C
1101	D
1110	E

1111

F

Codes ganz anderer Art sind:

- Flaggenrecode
- Morse-Code
- Braille-Schrift

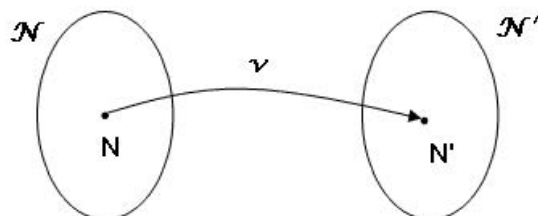
Digitale Nachrichten werden durch endliche Zeichenketten dargestellt. Diese Zeichenketten werden unterteilt in Teil-Zeichenketten bzw. Worte. Worte über einem binären Zeichenvorrat heißen Binärworte. Die Binärworte brauchen keine feste Länge zu haben, die meisten technischen Codes haben diese jedoch, wie z.B. der ISO 7-Bit-Code.

4.3 Nachrichten- und Informationsverarbeitung

Nachrichtenverarbeitung kann als eine Zuordnungs- bzw. Abbildungsvorschrift aufgefasst werden, die den Nachrichten N aus einer Nachrichtenmenge \mathcal{N} neue Nachrichten N' aus einer Nachrichtenmenge \mathcal{N}' zuordnet.

$$\nu : \mathcal{N} \rightarrow \mathcal{N}' \text{ mit } N \xrightarrow{\nu} N'$$

also:



Im folgenden Beispiel wird einer mit Buchstaben kodierte Nachricht, eine mit digitalen Bitmustern kodierte Nachricht zugeordnet.



Beispiel

$$\begin{array}{lcl} \text{es regnet} & \xrightarrow{v_1} & \underbrace{1100101}_e \quad \underbrace{1110011}_s \\ & & \underbrace{0100000}_\sqcup \\ & & \underbrace{1110010}_r \quad \dots \quad \underbrace{1110100}_t \end{array}$$

$$(3, 5) \xrightarrow{v_2} 8$$

$$(a, b) \xrightarrow{v_2} a + b$$

Wobei $a, b \in N$

Die Verarbeitung digitaler Nachrichten kann als Codierung aufgefasst werden.

Eine Menge \mathcal{N} von Nachrichten N ist nutzbar, wenn ihr eine Menge \mathcal{I} von Informationen I mittels einer Abbildungsvorschrift α , einer Interpretation, zugeordnet ist:

$$\alpha : \mathcal{N} \rightarrow \mathcal{I}$$

Betrachte zusätzlich die Zuordnungen

$$\nu : \mathcal{N} \rightarrow \mathcal{N}'$$

und

$$\alpha' : \mathcal{N}' \rightarrow \mathcal{I}'$$

Darstellung in einem Diagramm:

\mathcal{N}	$\xrightarrow{\alpha}$	\mathcal{I}
$\nu \downarrow$		$\sigma \downarrow$
\mathcal{N}'	$\xrightarrow{\alpha'}$	\mathcal{I}'

Frage: Welche Beziehung besteht zwischen \mathcal{I} und \mathcal{I}' ?

σ ist auch eine Abbildungsvorschrift und für $N \in \mathcal{N}$ gilt:

$$\sigma(\alpha(N)) = \alpha'(\nu(N))$$

ν ist dann eine informationstreue Verarbeitungsvorschrift.

Obiges Diagramm heißt dann kommutativ und σ ist eine Abbildungsvorschrift zur Informationsverarbeitung.

Ein kommutatives Diagramm verdeutlicht somit, dass verschiedene Verkettungen von Funktionen (Abbildungen) am Ende das gleiche Ergebnis liefern.

Anders ausgedrückt bedeutet dies, dass es gleichgültig ist, ob man in dem Diagramm von \mathcal{N} über \mathcal{I} nach \mathcal{I}' oder alternativ über \mathcal{N}' durchläuft.

D. h.: Die Interpretation einer Nachricht und die Veränderung dieser Interpretation liefern dasselbe Ergebnis, wie die Umkodierung der Nachricht und die anschließende Interpretation.



Beispiel

(a, b)	$\xrightarrow{\alpha}$	Zahlenpaar, bestehend aus den Zahlen a und b
\downarrow		\downarrow
$+$		Addition
$a + b$	$\xrightarrow{\alpha'}$	Summe der Zahlen a und b

Algorithmen in der Nachrichtenverarbeitung

Die Abbildungsvorschrift einer Nachrichtenverarbeitung

$$\nu : \mathcal{N} \rightarrow \mathcal{N}'$$

muss einen Weg angeben, wie man - ausgehend von einer Nachricht $N \in \mathcal{N}$ - die Nachricht $\nu(N) \in \mathcal{N}'$ konstruieren kann.

Dabei kann unterschieden werden:



Beispiel

1. \mathcal{N} endliche Menge: Auflisten der Paare (Kodierungstabelle).

Beispiel: lateinische Alphabet

a: 00000

b: 00001

c: 00010

...



Beispiel

2. \mathcal{N} unendliche Menge: Konstruktionsvorschrift, Algorithmus.

Beispiel: alle Paare (a,b), dabei sind a, b die natürlichen Zahlen.

Als Konstruktionsvorschrift könnte zum Beispiel die Summe beiden Zahlen, das Produkt usw. verwendet werden.

Ein Algorithmus (abgeleitet von dem persisch arabischen Mathematiker Al-Chowarizmi (9. Jahrhundert n. Chr.), Buch über die Regeln der Wiedereinsetzung und Reduktion) ist eine in einer festgelegten Sprache abgefasste endliche Beschreibung eines Verfahrens unter Verwendung ausführbarer elementarer Verarbeitungsschritte.

4.4 Aufgaben zum Thema: Information und Nachricht

In diesem Unterkapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt. Es empfiehlt sich, die Aufgaben selbstständig auf einem Blatt Papier zu lösen. Musterlösungen werden während des Semesters nach und nach freigeschaltet. Aus ähnlichen Aufgaben kann sich die Klausur zusammensetzen.



Aufgabe

Aufgabe 4.1

Nachrichten $N \in \mathcal{N}$ sollen durch natürliche Zahlen $n \in \mathbb{N}_0$ dargestellt werden, die in jeweils einem Byte durch die Zuordnungsvorschrift

$\nu : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ mit $\nu(n) := n \bmod 2^8$ codiert werden.

- Ist die codierte Nachricht decodierbar? Begründung!
- Für den Operator **mod** ist ein Algorithmus anzugeben, der auf den ganzzahligen Operationen $+$, $-$, $*$ und **div** basiert.



Aufgabe

Aufgabe 4.2

Ein Zeichenvorrat $V = \{\text{blank}, a, b, c, \dots, z, 0, 1, \dots, 9\}$ sei gegeben. Es ist eine Codierung in Binärworte der Länge 5 über dem Zeichenvorrat $\{0, 1\}$ anzugeben.



Aufgabe

Aufgabe 4.3

Es ist die folgende Nachricht gegeben:

1. Sg1-f3, d7-d6;
2. d2-d4, Lc8-g4;
3. c2-c4, Sg8-f6;

Welche Information wird durch diese Nachricht gegeben?

4.5 Fragen mit Musterlösungen: Information und Nachricht

In diesem Kapitel sind Verständnisfragen mit ihren Musterantworten zu dem vorangehenden Lehrstoff zusammengestellt. Solche Fragen werden auch in der Klausur vorkommen.

Im Folgenden sind die Verweise zu den Fragen aufgelistet. Mit einem Klick auf 'Antwort' am Ende der Fragenseite bekommt man die Antwort für das gestellte Problem angezeigt. Ein Klick auf "Info-Basis ..." führt zu dem Kapitel des für die Aufgabe relevanten Lehrstoffes.

Das Textfeld dient z.Z. lediglich dazu, eigene Antworten zu notieren, um sie besser mit der Musterantwort vergleichen zu können. Eine Speichermöglichkeit des Textes ist damit jedoch nicht gegeben.



Aufgabe

Frage 4.1

Welche Beziehung besteht zwischen Information und Nachricht?

Antwort

Die (abstrakte) Information wird durch die (konkrete) Nachricht mitgeteilt.
Beziehungen zwischen Information und Nachricht:

- Dieselbe Information kann durch verschiedene Nachrichten mitgeteilt werden.
- Es gibt Nachrichten, die keine Information enthalten.
- Es gibt Nachrichten, die nur für einen bestimmten Personenkreis eine Information enthalten.
- Nachrichten können, verschieden interpretiert, verschiedene Informationen ergeben.

Vgl. Kap. Informelle Einführung der Begriffe



Aufgabe

Frage 4.2

Was ist eine Interpretationsvorschrift?

Antwort

Eine Interpretationsvorschrift ist eine Abbildung, die Nachrichten aus einer Nachrichtenmenge in eine Information aus einer Informationsmenge abbildet.

Vgl. Kap. Nachrichten- und Informationsverarbeitung



Aufgabe

Frage 4.3

Was ist der Unterschied zwischen einer analogen und einer digitalen Nachricht?

Antwort

Analoge Nachrichten:

Abbildung in einen kontinuierlichen Wertebereich

Digitale Nachrichten:

Abbildung in einen diskreten Wertebereich.

Vgl. Kap. Digitale Nachrichten, Codes



Aufgabe

Frage 4.4

Was ist ein Code?

Antwort

Ein Code ist eine Vorschrift zur Zuordnung eines Zeichenvorrats zu einem anderen Zeichenvorrat. Eine Zuordnung wird Codierung genannt. Eine Zuordnung ist meist eindeutig und eindeutig umkehrbar (Decodierung).

Vgl. Kap. Digitale Nachrichten, Codes

5 Zahlen und Zahlssysteme



Gliederung

5 Zahlen und Zahlssysteme

5.1 Mathematischer Zahlbegriff

5.2 Die Ursprünge von Zahlensystemen

5.3 Stellenwertcodes und Konvertierung ganzer Zahlen

5.4 Darstellung negativer ganzer Zahlen

5.5 Addition und Subtraktion

5.6 Darstellung von Gleitpunktzahlen

5.7 Aufgaben zum Thema: Zahlen und Zahlensysteme

5.8 Fragen mit Musterlösungen: Zahlen und Zahlssysteme

5.1 Mathematischer Zahlbegriff

Zahlenmengen, die in der Mathematik eingeführt werden, sind

- natürliche Zahlen
- ganze Zahlen
- rationale Zahlen
- reelle Zahlen
- komplexe Zahlen.

Der Aufbau der Zahlensysteme geschieht entweder axiomatisch oder konstruktiv. Der Vorteil der axiomatischen Einführung ist ihre Kürze; als Nachteil kann angesehen werden, dass es leicht den Anschein hat, als fielen die Axiome vom Himmel. Dieses Problem tritt beim konstruktiven Aufbau nicht auf; dafür ist dieser, wenn er denn in allen Details durchgeführt wird, etwas langatmig.

Der konstruktive Aufbau geht aus von den natürlichen Zahlen

Die natürlichen Zahlen sind die beim Zählen verwendeten Zahlen (ein Apfel, zwei Äpfel, drei Äpfel usw.). Eine Natürliche Zahl ist ein Element der Menge $\{1,2,3,\dots\}$, (positive natürliche Zahlen) oder $\{0,1,2,3,\dots\}$, (nicht negative natürliche Zahlen).

Für die Menge der natürlichen Zahlen wird das Symbol \mathbb{N} verwendet. Je nach Definition können außerdem Symbole $\mathbb{N}_+ = 1, 2, 3, \dots$, für positive und $\mathbb{N}_0 = 0, 1, 2, 3, \dots$, für nicht negative natürliche Zahlen benutzt werden.

Die natürlichen Zahlen werden mit Hilfe der Peano-Axiomen definiert:

- 0 ist eine natürliche Zahl
- Mit jeder natürlichen Zahl n ist auch $s(n)$ eine natürliche Zahl. ($s(n)$ ist eine Nachfolgerfunktion: $s(n)=n+1$; Nachfolger, englisch: successor)
- Für alle n gilt: $s(n) \neq 0$. Es gibt also keine natürliche Zahl, deren Nachfolger gleich 0 ist.
- Axiom der vollständigen Induktion.
 - Sei $P(n)$ ein unäres Prädikat und $P(0)$; (d. h., dass P eine Aussage ist, die für eine Zahl n gültig sein kann. Diese Aussage gelte hier für die Zahl 0), also: $P(0)$ sei gültig.)
 - Wenn für alle n aus $P(n)$ nun $P(s(n))$ folgt, dann gilt $P(n)$ für alle n .

(Also wenn aus der Gültigkeit der Aussage P für eine Zahl n die Gültigkeit der Aussage für den Nachfolger $s(n)$ der Zahl n gefolgert werden kann, gilt die Aussage P für alle Zahlen n .) (Die Beweismethode der vollständigen Induktion basiert auf diesem Axiom).

Die Unlösbarkeit der Gleichung $n + x = m$ innerhalb der natürlichen Zahlen für gewisse $m, n \in \mathbb{N}$

führt zu den ganzen Zahlen

$$\mathbb{Z} = \dots, -3, -2, -1, 0, 1, 2, 3, \dots$$



Beispiel

die Gleichung $5 + x = 10$ hat eine Lösung im Bereich der natürlichen Zahlen $x = 5$, die Gleichung $6 + x = 4$ hat keine Lösung innerhalb der natürlichen Zahlen $x = -2, x \in \mathbb{N}$.

Entsprechend führen die nicht allgemeine Lösbarkeit von $mx = n \in \mathbb{Z}$ zu den rationalen Zahlen $\frac{n}{m} \in \mathbb{Q}$, mit $m, n \in \mathbb{Z}$, $m \neq 0$ und die nicht allgemeine Lösbarkeit von Gleichungen der Art $x^2 = q \in \mathbb{Q}$ zu den reellen Zahlen \mathbb{R} . Der letzte Schritt ist die Einführung der komplexen Zahlen \mathbb{C} , die u.a. durch die Nichtlösbarkeit im Allgemeinen von quadratischen Gleichungen in \mathbb{R} eingeführt werden können.

Ein wesentliches Ergebnis der Mathematik, besonders für den praktischen Umgang mit reellen Zahlen, ist ihre Darstellbarkeit durch natürliche Zahlen in der folgenden Weise:

Es sei $g \in \mathbb{N}$ mit $g \geq 2$ gegeben. Dann werden in dem folgenden Zusammenhang die natürlichen Zahlen a_v mit $0 \leq a_v < g$ als g -adische Ziffern bezeichnet. Die Summe

$\sum_{v=k}^{\infty} a_v \cdot g^{-v}$ mit den g -adischen Ziffern a_v ($k \in \mathbb{Z}$ beliebig, $v \geq k$) konvergiert dann und

stellt eine Zahl $x \in \mathbb{R}$ dar. Sofern nun nicht gilt $a_v = g - 1$ für fast alle v , nennt man die Summe auch **g-adische Entwicklung** der Zahl x . Es handelt sich hier also um einen Stellenwertcode zur Basis g .

Es gilt nun der folgende Satz [g-adische Entwicklung reeller Zahlen]:

Es seien $g \in \mathbb{N}$ mit $g \geq 2$ und $x \in \mathbb{R}$ gegeben. Dann gibt es genau eine g-adische Entwicklung von x , und es gilt

$$x = \sum_{v=k}^{\infty} a_v g^{-v}$$

Zur besseren Verdeutlichung seien hier einige Basen mit ihren zugehörigen Ziffern aufgeführt:

- Basis: 10; 10-adische Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Basis: 2; 2-adische Ziffern: 0, 1
- Basis: 3; 3-adische Ziffern: 0, 1, 2
- Basis: 5; 5-adische Ziffern: 0, 1, 2, 3, 4
- Basis: 6; 6-adische Ziffern: 0, 1, 2, 3, 4, 5
- Basis: 8; 8-adische Ziffern: 0, 1, 2, 3, 4, 5, 6, 7
- Basis: 16; 16-adische Ziffern: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f

Kennt man also eine Basis eines Zahlensystems (und somit die zugehörigen Ziffern), so lässt sich eine Zahl als (endliche oder unendliche) gewichtete Summe von Basispotenzen aufschreiben. Die einzelnen Koeffizienten (die Gewichte) in der gewichteten Summe entsprechen den Ziffern im Stellenwertcode der Zahl. Aufpassen muss man noch, dass in obiger Darstellung die Nachkommastellen des Stellenwertcodes der Zahl mit positiven Nummern durchnummeriert sind. Dies liegt daran, dass links vom Komma immer nur endlich viele Ziffern stehen (wenn man führende Nullen weglässt), aber rechts vom Komma unendlich viele Ziffern folgen können.



Beispiel

Die Zahl eintausendvierhundertundfünf bezüglich der Basis 10 als:

Stellenwertcode: 1405

10-adische Entwicklung:

$$\text{eintausendvierhundertundfünf} = 1 \cdot 10^{-3} + 4 \cdot 10^{-2} + 0 \cdot 10^{-1} + 5 \cdot 10^{-0}$$

$$= 1 \cdot 10^3 + 4 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0;$$

Die 1 ist hierbei die -3. Nachkommastelle, die 4 die -2. Nachkommastelle, die 0 die -1. Nachkommastelle und die 5 die -0. Nachkommastelle, also: $a_{-3} = 1$, $a_{-2} = 4$, $a_{-1} = 0$ und $a_0 = a_0 = 5$.

Der Index am **a** ist das ν (sprich: ny).



Beispiel

Welchen Wert im Dezimalsystem hat die Zahl mit dem Stellenwertcode 101.11 bezüglich der Basis 2?

Für die gesuchte Zahl gilt die 2-adische Entwicklung:

$$1 \cdot 2^{-2} + 0 \cdot 2^{-1} + 1 \cdot 2^{-0} + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

und somit im 10er-System: $4 + 0 + 1 + 0.5 + 0.25 = 5.75$.

Letzteres ist der Stellenwertcode im 10er-System.



Beispiel

$\frac{5}{6}$
g-adische Entwicklung von $\frac{5}{6}$ mit $g=10$:

Zuerst werde a_ν berechnet (Die Vorgehensweise entspricht dem schriftlichen Teilen mit dem Herunterholen der nächsten Null.):

$$5 = 0 \cdot 6 + 5, a_0=0$$

$$5 \cdot 10 = 8 \cdot 6 + 2, a_1=8$$

$$2 \cdot 10 = 3 \cdot 6 + 2, a_2=3$$

$$2 \cdot 10 = 3 \cdot 6 + 2, a_3=3$$

...

$$\frac{5}{6} = \sum_{\nu=0}^{\infty} a_\nu \cdot 10^{-\nu} = 0 \cdot 10^0 + 8 \cdot 10^{-1} + \sum_{\nu=2}^{\infty} 3 \cdot 10^{-\nu} = 0,8333...$$



Beispiel

g-adische Entwicklung von $0.777\dots$ mit $g=10$

$$0,7777777\dots = \sum_{\nu=1}^{\infty} 7 \cdot 10^{-\nu}$$

5.2 Die Ursprünge von Zahlensystemen

- Zahlengefühl: Endliche Mengen sind ihrer Größe nach vergleichbar
- Zahlendarstellung durch Kerben bzw. Finger, IIIIII
- Zahlenbegriffe für I, II, III, IIII, darüber hinaus "viele"
- Zahlenbegriffe, Zahlendarstellungen durch Ziffernsysteme
(ca. 3000 v. Chr., Sumerer)
- Brahmi-Dezimalziffern ohne Null und ohne Positionsprinzip
(ca. 200 v. Chr., Indien)
- Dezimales Positionssystem mit Null
(ca. 450 n. Chr., Indien)
- Indisches Zahlssystem wird auf islamischem Gebiet eingeführt
(ca. 800 n. Chr., Arabien)
- Arabische Ziffern und Zahlssystem in Europa
(ca. 1000 n. Chr., Spanien)
- Schriftliches Rechnen in Europa
(ab 1200 n. Chr.)
- Normung der Ziffern
(1500 n. Chr., durch Buchdruck)

5.3 Stellenwertcodes und Konvertierung ganzer Zahlen

Im folgenden werden Anwendungen von g-adischen Entwicklungen bzw. Stellenwertcodes gezeigt, die in der elektronischen Datenverarbeitung eine wichtige Rolle spielen. Zur Wiederholung wird jedoch zuerst noch einmal die Darstellung nichtnegativer ganzer Zahlen im Dezimalsystem betrachtet:

$$207 = 2 \cdot 10^2 + 0 \cdot 10^1 + 7 \cdot 10^0$$

Allgemein:

$$a = a_n a_{n-1} \dots a_0 = \sum_{i=0}^n a_i B^i, \quad 0 \leq a_i \leq B$$

Jede Stelle (Position) hat also einen bestimmten Stellenwert, der sich aus seiner Basis berechnet. Nun zu den in der elektronischen Datenverarbeitung wichtigen Darstellungen natürlicher Zahlen

(für **B = 2**, **B = 8**, **B = 16**):

Dualsystem (Leibnitz, 1679)

Das Dualsystem bildet derzeit die Grundlage von Zahlendarstellungen und der Arithmetik in Rechenanlagen. Dies liegt daran, dass sich die beiden Ziffern durch die zwei Schalterzustände "Schalter geschlossen" und "Schalter offen" repräsentieren lassen. (Anmerkung: Im Beispiel bezeichnet der Index am Stellenwertcode einer Zahl die verwendete Basis des Zahlensystems, dargestellt im Dezimalsystem.)

$B = 2$, Ziffern $\{0,1\}$

$$21_{10} = 10101_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

Oktalsystem

Das Oktalsystem spielt(e) in der Informatik eine wichtige Rolle, weil sich z. B. 24 Bit breite Datenworte (verwendet in älteren Großcomputern) leicht als achtstellige Okalzahlen darstellen lassen. Das ist so, weil die Umwandlung vom Dualsystem in das Oktalsystem und umgekehrt sehr einfach ist, da sich mit 3 Bits die Zahlen 0 bis 7 darstellen lassen, was exakt den 7 Ziffern im Oktalsystem entspricht. D. h., dass sich stets drei Bits zu einer Oktalziffer zusammenfassen lassen (, da $2^3 = 8$ gilt). Das folgende Beispiel entspricht der Dezimalzahl 21.

$B = 8$, Ziffern $\{0,1,2,3,4,5,6,7\}$

$$010\ 101_2 = 25_8 = 2 \cdot 8^1 + 5 \cdot 8^0$$

Hexadezimalsystem (Sedezimalsystem)

Analog zum Oktalsystem ist die Umwandlung vom Dualsystem in das Hexadezimalsystem ebenfalls einfach, weil hier je 4 Stellen einer Dualzahl einer Hexadezimalziffer entsprechen (, da hier $2^4 = 16$ gilt). Somit lassen sich 8-stellige, 16-stellige, 32-stellige oder 64-stellige Dualzahlen leicht als 2-stellige, 4-stellige, 8-stellige respektive 16-stellige Hexadezimalzahlen darstellen. Diese Stellenzahlen entsprechen wichtigen Datenwortbreiten in der heutigen Computertechnik (Stand 2009). Erneut entspricht das folgende Beispiel der Dezimalzahl 21.

$B = 16$, Ziffern $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

$$0001\ 0101 = 15 = 1 \cdot 16^1 + 5 \cdot 16^0$$

Konvertierungen

Konvertierungen zwischen Basen, die sich als Zweierpotenzen darstellen lassen, sind einfach (siehe oben). Es gilt sogar ganz allgemein, dass wenn eine Basis B_2 die n -te Potenz einer Basis B_1 ist, dass stets n Ziffern im Stellenwertcode einer Zahl zur Basis B_1 rechtsbündig einer Ziffer im Stellenwertcode der Basis B_2 entsprechen.

Wenn es sich nicht um solche Konvertierungen handelt, sind vor allem die Konvertierungen in das (von und gewohnte) Dezimalsystem wichtig, bzw. aus diesem in ein anderes System. Auf diese Weise kann man auch Konvertierungen zwischen Nicht-Dezimalsystemen, bei denen eine Basis auch nicht durch direkte Potenzierung aus der anderen erzeugt werden kann, berechnen, indem man zunächst eine Konvertierung in das Dezimalsystem und dann eine Konvertierung aus dem Dezimalsystem ausführt.

Konvertierung in das Dezimalsystem

Die Vorgehensweise besteht darin, dass man die andere Basis des Zahlensystems als Dezimalzahl schreibt, die Ziffern des Stellenwertcodes der Zahl zur anderen Basis als Dezimalzahlen (bei Basen kleiner als 10 sind auch Dezimalziffern möglich) schreibt und die 10-adische Entwicklung dieser Zahl ausrechnet:

$$a = a_n a_{n-1} \dots a_0 B = \sum_{i=0}^n a_i B^i$$



Beispiel

$$622_7 = (6 \cdot 7^2 + 2 \cdot 7^1 + 2 \cdot 7^0)_{10} = (294 + 14 + 2)_{10} = 310_{10},$$

$$3b0.2_{13} = (3 \cdot 13^2 + 11 \cdot 13^1 + 0 \cdot 13^0 + 2 \cdot 13^{-1})_{10} = (507 + 143 + 0 + \frac{2}{13})_{10}$$

$$= (507 + 143 + 0 + 0.153846153846153846...)_{10} = (650.153846...)_{10}.$$

Letzteres Beispiel verdeutlicht warum wir uns an dieser Stelle zunächst auf ganze Zahlen beschränken. Es zeigt, dass dieselbe gebrochene Zahl in einem Zahlensystem einen endlichen Stellenwertcode und in einem anderen Zahlensystem durchaus einem unendlichen Stellenwertcode haben kann.

Konvertierung aus dem Dezimalsystem



Code

x : Dezimalzahl

B : Basis

z : Ziffernsequenz zur Basis **B**

Programmcode:

(x,z) := (x,leer)

WHILE x ≠ 0 DO

(x,z) := (x DIV B, Concat(x MOD B , z))

END

Erläuterung: Eine Dezimalzahl, deren Stellenwertcode bezüglich einer anderen Basis bestimmt werden soll, wird ganzzahlig (mit Rest) durch diese neue Basis geteilt. Der Rest dieser Teilung ergibt die "rechtste" Ziffer (also die 0-te Nachkommastelle, entsprechend der 0-ten Vorkommastelle) im Stellenwertcode der Zahl bezüglich der neuen Basis. Der ganzzahlige Quotient wird erneut durch die neue Basis geteilt, man erhält die "zweitrechtste"

Ziffer (also die -1. Nachkommastelle, entsprechend der 1. Vorkommastelle) und so weiter. Das Verfahren endet, wenn der ganzzahlige Quotient Null ist.

Beispiele (Umrechnung der Dezimalzahl 21 in das Dualsystem, das Oktalsystem und das Hexadezimalsystem):

21		2	=	10	Rest	1		
10		2	=	5	Rest	0		

5		2	=	2	Rest	1		
2		2	=	1	Rest	0		
1		2	=	0	Rest	1	→	10101 ₂

21		8	=	2	Rest	5		
2		8	=	0	Rest	2	→	25 ₈

21		16	=	1	Rest	5		
1		16	=	0	Rest	1	→	15 ₁₆

Oder in der im Programmcode verwendeten Bezeichnungsweise (**x := 21, B := 2**) und mit dem Symbol + für die Konkatenation ("Hintereinanderschreibung")):

(x,z)	=	(21, "")		
(x,z)	=	(10, "1"+"")	=	(10, "1")
(x,z)	=	(5, "0"+"1")	=	(5, "01")
(x,z)	=	(2, "1"+"01")	=	(2, "101")
(x,z)	=	(1,"0" + "101")	=	(1, "0101")
(x,z)	=	(0,"1" + "0101")	=	(0,"10101")



In der Online-Version befindet sich an dieser Stelle eine Animation.

Beispiel Zahlenkonvertierung

Verarbeitungsbreite

In Rechenanlagen steht zur Zahlendarstellung nur eine bestimmte Stellenanzahl **t** zur Verfügung. Wenn man wissen will, welche Zahlen mit diesen **t** Stellen darstellbar sind geht man von folgender Grundidee aus:

1. Die kleinste Zahl **z** ist die Zahl, bei der alle ihre Ziffern des Stellenwertkodes 0 sind. Der Wert der Zahl ist:

$$z = 0 \cdot B_0 + 0 \cdot B_1 + \dots + 0 \cdot B^{t-1} \text{ (t Summanden).}$$

2. Die größte Zahl **Z** ist die Zahl, bei der alle ihre Ziffern des Stellenwertkodes **B-1** (also die größtmögliche Ziffer) sind. Der Wert der Zahl ist:

$$Z = (B-1) \cdot B^0 + (B-1) \cdot B^1 + \dots + (B-1) \cdot B^{t-1} \text{ (auch t Summanden).}$$

Somit ergibt sich für den Zahlenbereich:

$$0 \leq x \leq \sum_{i=0}^{t-1} (B-1) \cdot B^i = B^t - 1$$

Die kleinste Zahl ist also die Null, die größte ist die (Basis hoch Stellenzahl) minus 1. Z. B.:

$$B = 2, t = 16 : 0 \leq x \leq 65536 - 1 = 65535$$

Mathematischer Hintergrund: Die Bestimmung der oberen Schranke des Zahlenbereichs beruht auf der Formel:

$$\sum_{i=0}^{t-1} B^i = \frac{B^t - 1}{B - 1}$$

(Das obige Ergebnis für den Zahlenbereich erhält man, wenn man die letzte Gleichung mit **B-1** multipliziert und auf der linken Seite das Distributivgesetz anwendet.)

5.4 Darstellung negativer ganzer Zahlen

Im folgenden wird ein Stellenwertkode zur Basis B mit t Stellen betrachtet:

$$x = \sum_{i=0}^{t-1} a_i B^i$$

wobei $0 \leq a_i \leq B - 1$

Es gibt verschiedene Ansätze zur Darstellung negativer ganzer Zahlen:

1. Vorzeichenbit: vorderste Stelle wird als Vorzeichen interpretiert, d.h.

$a_{t-1} = 0$ positiv

$a_{t-1} \neq 0$ negativ

Für den Zahlenbereich steht somit eine Stelle weniger zur Verfügung und es gilt:

$$\text{Zahlenbereich: } -B^{t-1} + 1 \leq x \leq B^{t-1} - 1$$

$$B = 2, t = 16 \quad -32767 \leq x \leq 32767$$

Bemerkung: Die Null wird doppelt dargestellt.

2. Echtes Komplement / B-Komplement (Zweierkomplement für $B = 2$):

Darstellung von $-x$ durch $B^t - x$, d.h.

$$B^t - x = 1 + \sum_{i=0}^{t-1} (B-1) B^i - \sum_{i=0}^{t-1} a_i B^i = 1 + \sum_{i=0}^{t-1} (B-1-a_i) B^i$$

d.h. durch Bildung des Komplements und Addition von 1.

B=2, t=8		21_{10}	=	0001 0101 ₂
	-	21_{10}	=	1110 1010 ₂
			+	1 ₂

				1110 1011 ₂

Erneute Anwendung führt wieder zur selben positiven Zahl:

				0001 0100 ₂
			+	1 ₂

				0001 0101 ₂

Gleiches Verfahren zur Basis 8:

B=2, t=4		21_{10}	=	0025 ₈
	-	21_{10}	=	7752 ₈
			+	1 ₈

				7753 ₈

Erneute Anwendung:

			0024 ₈	
			+	1 ₈

				0025 ₈

Zahlenbereich: $-B^{t-1} \leq x \leq B^{t-1} - 1$

$$B = 2, t = 16 \quad -32767 \leq x \leq 32767$$

Bemerkung: Die Darstellung der Null ist eindeutig.

Das Zweierkomplement in der heutigen Computertechnik von herausragender Bedeutung, weil sich die Subtraktion einer Zahl auf die Addition des Zweierkomplementes dieser Zahl zurückführen lässt. Ausserdem ist die Umwandlung in das und aus dem Zweierkomplement sehr einfach. Alle in diesem Absatz genannten Operationen lassen sich leicht in Hardware implementieren.

3. Stellenkomplement / B-1-Komplement (Einerkomplement für B = 2): Darstellung von -x durch Bildung des Komplements (ohne Addition von 1) $B^t - 1 - x$:

$$B^t - 1 - x = \sum_{i=0}^{t-1} (B - 1 - a_i) B^i$$

Zahlenbereich: $-B^{t-1} + 1 \leq x \leq B^{t-1} - 1$

Bemerkung: Die Null wird doppelt dargestellt.

Dieses Komplement hat seine Bedeutung verloren, da sich die Addition und Subtraktion nicht so einfach in Hardware implementieren lassen, wie beim Zweierkomplement. Es ist nur noch der Vollständigkeit halber aufgeführt.

Zusammenfassender Vergleich (für B = 2, t = 4):

Zahlenwert mit Vorzeichenbit	Zahlenwert mit 2-Komplement	Zahlenwert mit 1-Komplement	
0000	+0	0	+0
0001	+1	+1	+1
0010	+2	+2	+2

0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	-0	-8	-7
1001	-1	-7	-6
1010	-2	-6	-5
1011	-3	-5	-4
1100	-4	-4	-3
1101	-5	-3	-2
1110	-6	-2	-1
1111	-7	-1	-0

Eine häufig verwendete Darstellung ordnet die Binärmuster auf einem Kreis an (Zahlenkreis).

5.5 Addition und Subtraktion

Für die Verwendbarkeit von Zahlendarstellungen in Rechenanlagen ist auch ein einfaches Rechnen mit diesen Darstellungen notwendig.

Die feste Verarbeitungsbreite führt zu dem Problem, dass die Operationen nicht abgeschlossen sind, d. h. es gibt einen Überlauf aus dem Zahlenbereich, der anzuzeigen ist.

In Computern wird für die Subtraktion und zur Darstellung negativer Zahlen das nachfolgend dargestellte Zweierkomplement verwendet, da Addition und Subtraktion durch nur ein Schaltwerk realisiert werden können. Anhand von sechs Beispielen mit jeweils vier Binärstellen pro Zahl wird dies im Folgenden verdeutlicht:

$$t = 4 : \text{Zahlenbereich } 1000_2 = -8_{10} \leq x \leq 7_{10} = 0111_2$$

1:		4		0100		2:		4		0100	
----	--	---	--	------	--	----	--	---	--	------	--

	+	2	+	0010			-	2	+	1110	
				----						----	
		6		0110	gültig			2		0010	Überlaufunterdrückung
3:	-	4		1100		4:	-	4		1100	
		2	+	0010			-	2		1110	
				----						----	
	-	2		1110	gültig		-	6		1010	Überlaufunterdrückung
5:		4		0100		6:	-	4		1100	
	+	5		0101			-	5		1011	
				----						----	
		9		1001	ungültig		-	9		0111	ungültig

Erläuterungen:

1. Das höchstwertige Bit beider Summanden ist 0.

Das höchstwertige Bit der Summe ist 0.

Ein Übertrag in ein fünftes höchstwertiges Bit (also ein Überlauf) findet nicht statt. Es handelt sich also um eine Addition ohne Überlauf und ohne Vorzeichenwechsel, also ist das Ergebnis gültig.

(Anschaulich gesehen werden zwei genügend kleine positive Zahlen addiert, deren Summe ebenfalls im Wertebereich des Zweierkomplementes mit 4 Stellen darstellbar ist.)

2. Das höchstwertige Bit beider Summanden ist einmal 0 und einmal 1.

Das Ergebnis der Summe aus einer positiven und einer negativen Zahl ist vom Betrag her niemals größer als der vom Betrag her größere der beiden Summanden. (Machen Sie sich dies gegebenenfalls am Zahlenstrahl oder Zahlenkreis klar.)

Ein Übertrag in ein fünftes höchstwertiges Bit (also ein Überlauf) findet zwar statt, aber spielt keine Rolle, weil eine vom Betrage her größere oder gleichgroße Zahl (einer der Summanden) im Zahlenbereich des Zweierkomplementes darstellbar ist, muss es auch eine vom Betrage her gleichgroße oder kleinere Zahl sein.

Es handelt sich also um eine Addition mit einem Überlauf, der nicht beachtet wird, also um eine Überlaufunterdrückung und das Ergebnis ist gültig.

3. Das höchstwertige Bit beider Summanden ist einmal 0 und einmal 1.

Für die Darstellbarkeit des Ergebnisses gilt dieselbe Überlegung wie bei Beispiel 2. Außerdem gibt es noch nicht einmal einen Überlauf, und das Ergebnis ist einfach gültig.

4. Das höchstwertige Bit beider Summanden ist 1.

Das höchstwertige Bit der Summe ist 1.

Ein Überlauf findet statt.

Es handelt sich also um einen Überlauf ohne Vorzeichenwechsel, das heißt, der Überlauf wird nicht beachtet, es gibt eine Überlaufunterdrückung und somit ist das Ergebnis gültig. (Anschaulich gesehen werden hier zwei genügend kleine negative Zahlen addiert, so dass ihre (vom Betrage her größere) Summe immer noch mit 4 Binärstellen im Zweierkomplement darstellbar ist.)

5. Das höchstwertige Bit beider Summanden ist 0.

Das höchstwertige Bit der Summe ist 1.

Es findet ein zwar kein Überlauf statt, aber ein Vorzeichenwechsel, und somit ist das Ergebnis ungültig.

(Anschaulich gesehen werden hier zwei genügend große positive Zahlen addiert, deren Summe dann nicht mehr mit 4 Binärstellen im Zweierkomplement darstellbar ist.)

6. Das höchstwertige Bit beider Summanden ist 1.

Das höchstwertige Bit der Summe ist 0.

Es findet also ein Überlauf mit Vorzeichenwechsel statt und somit ist das Ergebnis ungültig. (Anschaulich gesehen werden zwei genügend kleine (vom Betrage her genügend große) negative Zahlen addiert, so dass Ihre Summe im Zahlenbereich des Zweierkomplementes mit vier Binärstellen nicht mehr darstellbar ist.)

Analoge Überlegungen funktionieren auch bei anderen Stellenzahlen, z. B. 8, 16, 32, 64, usw. .

Genauere Betrachtung des Vorzeichenwechsels bzw. Überlaufs:

Wenn die beiden Summanden (ggf. nach der Zweierkomplementbildung) im höchsten Bit denselben Wert stehen haben (also beide 0 oder beide 1) und an derselben Stelle der Summe der Wert unterschiedlich ist (also 1, wenn beide 0 waren oder 0, wenn beide 1 waren), dann und nur dann liegt ein echter Überlauf des Zahlenbereiches und somit ungültiges Ergebnis vor. Mit anderen Worten: Wenn zwei positive Summanden addiert werden, darf das Ergebnis nicht negativ sein und wenn zwei negative Summanden addiert werden, nicht positiv.

Wir betrachte die beiden hier genannten Fälle nun noch einmal getrennt, um zu einer technisch leicht prüfbar äquivalenten Aussage zu kommen.

Das jeweils höchstwertige Bit der beiden Summanden sei 0, das höchstwertige Bit der Summe jedoch 1. Dieser Fall kann dann und nur dann eintreten, wenn der Übertrag in die höchste Stelle 1 war.

Analog: Das jeweils höchstwertige Bit beider Summanden sei 1, das höchstwertige Bit der Summe jedoch 0. Dieser Fall kann dann und nur dann eintreten, wenn der Übertrag in die höchste Stelle 0 war.

Als Test auf die Gültigkeit des Ergebnisses erhält man also folgenden Zusammenhang: Wenn die beiden Summanden im jeweils höchstwertigen Bit denselben Wert haben, so muss der Übertrag in die höchste Stelle bestimmt werden. (Dies geschieht beim normalen schriftlichen Addieren und bei bestimmten Schaltungen ohnehin.) Ist dieser Übertrag von dem jeweils höchstwertigen Bit der beiden Summanden verschieden, liegt ein ungültiges Ergebnis (Überlauf) vor. In allen anderen Fällen, also insbesondere auch wenn die beiden Summanden sich in ihrem jeweils höchstwertigen Bit unterscheiden haben, liegt ein gültiges Ergebnis vor.

(Anmerkung: Es empfiehlt sich, die hier genannten Zusammenhänge anhand von selbst kreierten Rechenaufgaben zu üben.)

5.6 Darstellung von Gleitpunktzahlen

Auf Konrad Zuse (1910 - 1995) zurückgehend, werden reelle Zahlen in Rechnern durch Gleitpunktzahlen angenähert (1937):

$x = m \cdot b^e$	wobei:	m = Mantisse, b = Basis, e = Exponent
-------------------	--------	---------------------------------------------------

m und e werden mit $B = b = 2$ dargestellt, negative Zahlen durch ein Vorzeichenbit. Das Problem an dieser Darstellungsweise ist, dass sie nicht eindeutig ist. Ein Beispiel:

$$4 = 2.8284271 \dots 2^{\frac{1}{2}} = 4 \cdot 2^0 = 2 \cdot 2^1 = 1 \cdot 2^2 = 0,5 \cdot 2^3$$

Obwohl Mantisse und Exponent in obiger Zeile in jedem Gleichungsabschnitt variieren, wird doch stets die Zahl 4 dargestellt. Um aus diesem Mehrdeutigkeitsdilemma herauszukommen kann z. B. für die Anwendung in Computern vereinbart werden, dass gilt:

e ist ganzzahlig, $|m| < 1$

Die Mantisse ist also stets ein Dualbruch der Form: $0.a_1a_2a_3\dots$, wobei man die 0 vor dem Dezimalpunkt auch weglassen kann.

$m = a_1a_2 \dots a_n$	=	$\sum_{i=1}^n a_i B^{-i}$	
$m = .101$	=	$1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$	
	=	$\frac{1}{2} + \frac{1}{8}$	
	=	$0,5 + 0,125$	$= 0,625$

Leider sind die Gleitpunktdarstellungen immer noch nicht eindeutig, wie folgendes Beispiel zeigt

(Anmerkung: Es gilt stets, dass die Null vor dem Dezimalpunkt und der Dezimalpunkt weggelassen sind):



Beispiel

$$\begin{aligned}
 x_1 &= 0101 \cdot b^{011} = \left(\frac{1}{4} + \frac{1}{16}\right) \cdot 2^3 \\
 &= 0,3125 \cdot 2^3 = 2,5 \\
 x_2 &= 0101 \cdot b^{010} = \left(\frac{1}{2} + \frac{1}{8}\right) \cdot 2^2 \\
 &= 0,625 \cdot 2^2 = 2,5
 \end{aligned}$$

Deshalb werden die Darstellungen für Zahlen $\neq 0$ normalisiert, d. h.:

Normalisierung: $b^{-1} \leq |m| < 1$

- Für die Basis 2 ist die Mantisse also gleich oder größer als $\frac{1}{2}$ und kleiner als 1. Im Dualsystem erreicht man dies durch:
 - Stellen in der Mantisse nach links verschieben, so dass das höchstwertige Bit gleich 1 ist
 - Stellenanzahl der Verschiebung im Exponenten subtrahieren.

Im obigen Beispiel ist also nur die Zahl x_2 eine gültige Darstellung.

5.7 Aufgaben zum Thema: Zahlen und Zahlensysteme

In diesem Unterkapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt. Es empfiehlt sich, die Aufgaben selbstständig auf einem Blatt Papier zu lösen. Musterlösungen werden während des Semesters nach und nach freigeschaltet. Aus ähnlichen Aufgaben kann sich die Klausur zusammensetzen.



Aufgabe

Aufgabe 5.1

Wandeln Sie die folgenden Darstellungen natürlicher Zahlen in Darstellungen bzgl. der neuen gegebenen Basis um

a. $10_{10} = (\dots)_{16}$

b. $100_8 = (\dots)_7$

c. $20_5 = (\dots)_2$

d. $123_4 = (\dots)_3$

e. $1024_{10} = (\dots)_{16}$



Aufgabe

Aufgabe 5.2

Wie lauten die hexadezimalen Anfangsadressen für byteweise organisierte Speicherbereiche, die auf den Grenzen 1K, 4K, 1M, 16M, 1G beginnen? Die Adressbreite sei 32 Bit.



Aufgabe

Aufgabe 5.3

Wandeln Sie die Darstellungen ganzer Zahlen in den folgenden Aufgaben um in Darstellungen des Zweierkomplements und des Einerkomplements und lösen Sie die Aufgaben in dieser Darstellung (Stellenzahl $t = 5$):

- a. $10 + 14$
- b. $6 - 12$
- c. $-11 + 4$
- d. $-13 - 15$

5.8 Fragen mit Musterlösungen: Zahlen und Zahlssysteme

In diesem Kapitel sind Verständnisfragen mit ihren Musterantworten zu dem vorangehenden Lehrstoff zusammengestellt. Solche Fragen werden auch in der Klausur vorkommen.

Im Folgenden sind die Verweise zu den Fragen aufgelistet. Mit einem Klick auf 'Antwort' am Ende der Fragenseite bekommt man die Antwort für das gestellte Problem angezeigt. Ein Klick auf "Info-Basis ..." führt zu dem Kapitel des für die Aufgabe relevanten Lehrstoffes.

Das Textfeld dient z.Z. lediglich dazu, eigene Antworten zu notieren, um sie besser mit der Musterantwort vergleichen zu können. Eine Speichermöglichkeit des Textes ist damit jedoch nicht gegeben.



Aufgabe

Frage 5.1

Welche Zahlenmengen gibt es, welche sind abzählbar und welche nicht?

Antwort

- Menge der natürlichen Zahlen (abzählbar)
- Menge der ganzen Zahlen (abzählbar)
- Menge der rationalen Zahlen (abzählbar)
- Menge der reellen Zahlen (nicht abzählbar)
- Menge der komplexen Zahlen (nicht abzählbar)

Vgl. Kap. Mathematischer Zahlbegriff



Aufgabe

Frage 5.2

Was wird unter der g-adischen Entwicklung reeller Zahlen verstanden?

Antwort

Es seien $g \in \text{Element von } \mathbb{N}$ mit $g \geq 2$ und $x \in \mathbb{R}$ gegeben. Dann gibt es genau eine g-adische Entwicklung von x , und es gilt

$$x = \sum_{k \leq v \text{ inf}} a_v \cdot g^{-v}.$$

wobei $a_v \in \mathbb{N}$ mit $0 \leq a_v < g$ g-adische Ziffern.

Anmerkung: Für $g=10$ ergeben sich die Dezimalzahlen mit gegebenenfalls unendlich vielen Stellen hinter dem Komma.

Vgl. Kap. Mathematischer Zahlbegriff



Aufgabe

Frage 5.3

Was wird mit den Cantorschen Diagonalverfahren gezeigt?

Antwort

Über das erste Cantorsche Diagonalverfahren wird die Abzählbarkeit der Menge der rationalen Zahlen gezeigt, über das zweite die Überabzählbarkeit der Menge der reellen Zahlen.

Vgl. Kap. Mathematischer Zahlbegriff



Aufgabe

Frage 5.4

Wie lassen sich negative ganze Zahlen binär darstellen?

Antwort

Die Darstellung negativer ganzer Zahlen kann erfolgen über:

- Vorzeichenbit
- B-Komplement, echtes Komplement
- B-1 Komplement, Stellenkomplement

Vgl. Kap. Darstellung negativer ganzer Zahlen

**Aufgabe****Frage 5.5**

Was ist eine Gleitpunktzahl und was versteht man unter Normalisierung?

Antwort

Über Gleitpunktzahlen lassen sich reelle Zahlen annähern: $x = m * b^e$, wobei m die Mantisse, b die Basis und e der Exponent ist.

Eine normalisierte Gleitpunktzahl ist eine Darstellung, bei der das höchstwertigste Bit von 0 verschieden ist.

Vgl. Kap. Darstellung von Gleitpunktzahlen

6 Algorithmen und Datenstrukturen



Gliederung

6 Algorithmen und Datenstrukturen

6.1 Algorithmen

6.2 Datenstrukturen

6.3 Sortieren

6.4 Aufgaben zu Algorithmen und Datenstrukturen

Dabei wird im Unterkapitel Algorithmen auf einen der grundlegenden Begriffe der Informatik eingegangen, den Algorithmus, die Beschreibung eines schrittweisen Verfahrens zur Lösung gleichartiger Probleme. Anschließend wird zunächst der Algorithmusbegriff näher betrachtet. Dann wird gezeigt, welcher Art die Theoretischen Fragestellungen im Bezug auf Algorithmen sein können. Danach werden noch verschiedene Aspekte genannt, die bei der Umsetzung eines Verfahrens aus der realen Welt in einen Algorithmus, also bei der Algorithmisierung, zu beachten sind. Dies sind zum Beispiel Programmierparadigmen, Programmiersprachen und Kontrollstrukturen.

An anderer Stelle des Studienmoduls, im Unterkapitel 3.2 Simulation, wird erläutert, dass bei der Modellierung der realen Welt in Systemen der automatisierten Datenverarbeitung Daten entstehen bzw. verwendet werden. Hier, im Unterkapitel 6.2 Datenstrukturen werden Informationen dazu bereitgestellt, wie einfache und komplexere Daten in Systemen der automatisierten Datenverarbeitung dargestellt werden können. Es wird dabei im Abschnitt Abstrakte Datentypen sowohl auf einfache Daten eingegangen, als auch auf zusammengesetzte Daten, die keine Zeiger (Pointer) verwenden. (Was Zeiger/Pointer sind ist in dem genannten Abschnitt erklärt; Anmerkung des Autors.) Im Abschnitt Dynamische Datenstrukturen werden dagegen mit Listen und Bäumen zusammengesetzte Daten beschrieben, deren effektiver Nutzwert aus der Verwendung von Zeigern entsteht. Dabei wird auch erläutert, wie diese Datenstrukturen aus einem abstrakten Datentyp entstehen.

Abschließend wird im Bereich Sortieren gezeigt, wie die Betrachtung der Algorithmen und die Betrachtung der Datenstrukturen miteinander verschmelzen, da das Beispiel auf die Datenstruktur eines Feldes abgestimmt ist. Es ist auf andere Datenstrukturen anpassbar. Dafür muss dann aber der Algorithmus leicht verändert werden.

6.1 Algorithmen



Gliederung

6.1 Algorithmen

6.1.1 Theoretische Fragestellungen zu Algorithmen

6.1.2 Algorithmisierung

Motivation

Wenn jemand vor einem Problem einer bestimmten Art steht, und diese Person einen Zettel besitzt, auf dem

- eine (detaillierte) Handlungsanweisung steht, wie mit dieser Art von Problemen umzugehen ist,
- und die Person mechanisch, und ohne um den Sinn zu wissen, die einzelnen Schritte der Handlungsanweisung umsetzt,
- und so das gegebene Problem einer Lösung zuführt,

so kann man bei der Handlungsanweisung auf dem Zettel von einem Algorithmus (im intuitiven Sinne) sprechen.



Beispiel

Cremepudding

Problem: Es ist ein Cremepudding zuzubereiten.

Lösung: Kochrezept

Zutaten: $\frac{3}{4}$ l kalte Milch, 1 Päckchen Puddingpulver, zwei gut gehäufte Esslöffel Zucker, 1 Ei.

Zubereitung: Man nehme 6 Esslöffel von der Milch und verquirle damit das Puddingpulver, den Zucker und das Eigelb des Eis. Man bringe die Milch zum Kochen, gebe dann das verquirelte Puddingpulver hinzu und lasse alles unter Rühren kurz aufkochen. Das zu steifem Schnee geschlagene Eiweiß wird sofort nach dem Kochen unter die Speise gehoben und diese in die Gläser gefüllt.

Algorithmus CP (Cremepudding)

3

Aus 4 l kalter Milch, einem Päckchen Puddingpulver, 2 gut gehäuften Esslöffeln Zucker und einem Ei soll ein Cremepudding für 4 Personen zubereitet werden.

CP1	<i>[Verquirl – Milch]</i>	Es sind 6 Esslöffel von der Milch abzunehmen.
CP2	<i>[Ei trennen]</i>	Ei in Eigelb und Eiweiß trennen.
CP3	<i>[Verquirlen]</i>	Die 6 Esslöffel Milch (CP1), das Puddingpulver, den Zucker und das Eigelb (CP2) verquirlen.
CP4	<i>[Eischnee]</i>	Das Eiweiß (CP2) zu steifem Schnee schlagen.
CP5	<i>[Milch aufkochen]</i>	Die restliche Milch auf den Herd stellen und dort stehen lassen, bis sie kocht.
CP6	<i>[Puddingpulver zur Milch]</i>	Das verquirlte Puddingpulver (CP3) in Milch (CP5) geben.
CP7	<i>[Mischung aufkochen]</i>	Die mit dem Puddingpulver vermischte Milch (CP6) solange umrühren, bis sie gerade aufkocht.
CP8	<i>[Eiweiß hinzufügen]</i>	Das geschlagene Eiweiß (CP4) sofort unter die Speise (CP7) heben.
CP9	<i>[Abfuellen]</i>	Den fertigen Cremepudding (CP8) in Gläser geben. Der Algorithmus endet.

Algorithmus, intuitiv

Ausgehend von oben stehender Motivation wird nun der intuitive Algorithmusbegriff noch etwas präzisiert:

**Definition****Intuitiver Algorithmus**

Ein Algorithmus ist eine Vorgehens- oder Berechnungsvorschrift mit folgenden Eigenschaften:

- Sie ist ein endlicher Text.
- Sie enthält ausführbare Einzelschritte.
- Durch die Ausführung dieser Einzelschritte wird eine geeignete Problemsituation einer bestimmten Art schrittweise einer Lösung zugeführt, respektive geeignete Eingabedaten über Zwischenergebnisse zu Ausgabedaten verarbeitet.
- Für eine solche geeignete Problemsituation und für geeignete Eingabedaten hält

der Algorithmus nach endlich vielen Schritten an. Man sagt auch: Er terminiert.

Algorithmen können dabei unter anderem folgende Eigenschaften haben oder auch nicht haben. Sie sind (möglicherweise):

Vollständig,

das heißt, sie liefern ein Ergebnis für alle denkbaren Variationen einer Problemsituation bzw. für alle denkbaren Variationen gleichartiger Eingabedaten.

Determiniert,

das heißt, dass dieselbe Problemsituation auch bei mehrfacher Verarbeitung jeweils derselben Lösung zugeführt wird. Oder auch: Für die gleichen (nicht nur gleichartigen) Eingabedaten berechnet der Algorithmus die gleichen Ausgabedaten.

Deterministisch,

das heißt, der Algorithmus ist determiniert und führt darüber hinaus bei wiederholter Ausführung für dieselbe Problemsituation / mit den gleichen Eingabedaten dieselben Einzelschritte mit denselben Zwischenergebnissen aus.

In untenstehenden Beispiel ist ein determinierter Algorithmus dargestellt, der unter bestimmten Bedingungen jedoch nicht deterministisch ist. Hierbei wird zuerst der Algorithmus dargestellt und anschließend erläutert, warum er unter Umständen nicht deterministisch ist:

**Beispiel****Paralleler Cremepudding**

Problem: Es ist ein Cremepudding zuzubereiten.

Lösung: Kochrezept

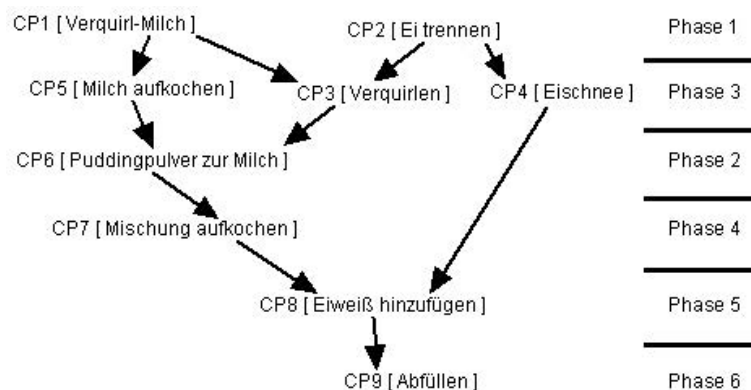
Zutaten: 3/4 l kalte Milch, 1 Päckchen Puddingpulver, zwei gut gehäufte Esslöffel Zucker, 1 Ei.

Zubereitung: Man nehme 6 Esslöffel von der Milch und verquirle damit das Puddingpulver, den Zucker und das Eigelb des Eis. Man bringe die Milch zum Kochen, gebe dann das verquirelte Puddingpulver hinzu und lasse alles unter Rühren kurz aufkochen. Das zu steifem Schnee geschlagene Eiweiß wird sofort nach dem Kochen unter die Speise gehoben und in die Gläser gefüllt.

Algorithmus PCP (Paralleler Cremepudding)

Aus 3/4 l kalter Milch, einem Päckchen Puddingpulver, 2 gut gehäuften Esslöffeln Zucker und einem Ei soll ein Cremepudding für 4 Personen zubereitet werden.

PCP [CP in Phasen]: Man führt die im Algorithmus CP angegebenen Schritte nach dem Zeitdiagramm in untenstehender Abbildung aus.

**PCP [CP in Phasen]**

Der Algorithmus ist deshalb determiniert, weil immer der gleiche Cremepudding hergestellt wird. Wenn wir annehmen, dass jeder Schritt, sobald er ansteht, sofort ausgeführt wird und alle Schritte genau eine Phase lang andauern (wir haben also für jeden der Schritte einen Koch und alle Köche benötigen jeweils für jeden der Arbeitsschritte genau dieselbe Zeit), so erzeugt der Algorithmus auch immer dieselben Zwischenergebnisse:

Phase 1: CP1 und CP2

Die Verquir-Milch wird abgemessen und das Ei getrennt.

Phase 2: CP5, CP3 und CP4

Die restliche Milch wird aufgekocht, die Verquirl-Milch, der Zucker, das Puddingpulver und das Eigelb werden verquirlt, der Eischnee wird geschlagen.

Phase 3: CP6

Das verquirlte Puddingpulver wird in die Milch gegeben.

Phase 4: CP7

Die Mischung wird aufgekocht.

Phase 5: CP8

Das Eiweiss wird unter die Masse gehoben.

Phase 6: CP9

Der Pudding wird in Gläser gefüllt.

Mit der in obiger Liste dargestellten Art der Ausführung der einzelnen Arbeitsschritte ist der Algorithmus also nicht nur determiniert, sondern sogar deterministisch, weil stets (in jeder Phase) dieselben Zwischenergebnisse entstehen.

Betrachten wir nun denselben Algorithmus erneut, nun allerdings mit der Änderung, dass zum Kochen des Puddings genau zwei Köche zur Verfügung stehen:

Phase 1: CP1 und CP2

Die Verquirl-Milch wird von Koch 1 abgemessen und das Ei durch Koch 2 getrennt.

Phase 2: CP5, CP3 und CP4

Die restliche Milch wird aufgekocht, die Verquirl-Milch, der Zucker, das Puddingpulver und das Eigelb werden verquirlt, der Eischnee wird geschlagen.

:

In Phase 2 entsteht nun ein Konflikt. Es sollen drei Schritte ausgeführt werden, aber es stehen nur zwei Köche zur Verfügung. Es gibt also drei Möglichkeiten, wie die beiden Köche in Phase 2 vorgehen. Falls sie sich gegen CP5 oder CP3 entscheiden, wird sogar noch mindestens Phase 3 beeinflusst. Es entstehen also unterschiedliche Zwischenergebnisse. Das bedeutet: Der Algorithmus PCP mit einem Team von nur zwei Köchen ist zwar determiniert, weil immer ein gleichartiger Cremepudding hergestellt wird, aber nicht deterministisch, weil unterschiedliche Zwischenergebnisse entstehen.

**Anmerkung**

Der Nichtdeterminismus bietet hier Raum für Optimierungen: Werden in Phase 2 die Schritte CP5 und CP3 zuerst ausgeführt, kann anschließend in Phase 3 gleich CP6 ausgeführt werden. Wird mindestens einer der beiden Schritte CP5 oder CP3 nicht ausgeführt, so kann in Phase 3 der Schritt CP6 nicht ausgeführt werden. Anders ausgedrückt: Nichtdeterminismus bei

paralleler Ausführung von Einzelschritten des Algorithmus bietet die Möglichkeit zur Optimierung (oder zur suboptimalen Abarbeitung).

Es sei noch einmal betont: Um zu beurteilen, ob ein Algorithmus deterministisch ist, sind Kenntnisse über den Ausführungsmechanismus (den Interpretationsmechanismus, der den Algorithmus ausführt) erforderlich.

Analog dazu muss bei der Beurteilung, ob ein Algorithmus vollständig ist oder nicht, außer dem Algorithmus selbst zusätzlich beachtet werden, für welche Ausgangssituationen (Eingabedaten) dieser zulässig ist.

Um dies zu verdeutlichen, wird im untenstehenden Beispiel der Algorithmus des Euklid betrachtet, mit dem der größte gemeinsame Teiler zweier natürlicher Zahlen berechnet wird.

**Beispiel****Algorithmus des Euklid**

Problem: Der größte gemeinsame Teiler zweier natürlicher Zahlen ist zu berechnen.

Lösung: Algorithmus des Euklid

Algorithmus E (Euklid)

I. [Restbildung] Es wird m durch n geteilt; der Rest sei r .

(Es gilt $0 \leq r < n$)

II. [Ist Rest Null?] Wenn $r = 0$, endet der Algorithmus; n ist das Ergebnis.

III. [Vertausche] Setze $m := n$ und $n := r$ und gehe zu Schritt I zurück.

Bei jedem Durchlauf i mit i ist Element von $\{1, 2, 3, \dots\}$ gilt dabei: $m_i \div n_i = q_i + r_i$, also: m_i ganzzahlig geteilt durch $n_i = q_i$ Rest r_i .

Anwendung des Algorithmus

Algorithmus E mit $m := 119$ und $n := 544$

1.	$119 = 0 \cdot 544 + 119$ $r := 119$	I
2.	$r \neq 0$	II
3.	$m := 544 \quad n := 119$	III
4.	$544 = 4 \cdot 119 + 68$ $r := 68$	I
5.	$r \neq 0$	II
6.	$m := 119 \quad n := 68$	III
7.	$119 = 1 \cdot 68 + 51$ $r := 51$	I
8.	$r \neq 0$	II
9.	$m := 68 \quad n := 51$	III
10.	$68 = 1 \cdot 51 + 17$ $r := 17$	I
11.	$r \neq 0$	II
12.	$m := 51 \quad n := 17$	III
13.	$51 = 3 \cdot 17 + 0$ $r := 0$	I
14.	$r = 0; n = 17$	II

Rechner zum Euklid-Algorithmus (12KB)



In der Online-Version befindet sich an dieser Stelle eine Animation.

Rechner zum Euklid-Algorithmus

Bei der Anwendung des Algorithmus wird folgendes deutlich: Teilt man zunächst die kleinere durch die größere Zahl, so bleibt die kleinere Zahl als Rest und nach dem Vertauschen: $m := n$ und $n := r$ wird beim nächsten Durchlauf die größere durch die kleinere Zahl geteilt. Danach läuft der Algorithmus weiter ab, bis der Rest beim Teilen Null ist, also $r = 0$ gilt.

Betrachtet wird nun die Menge der Natürlichen Zahlen: Diese wird in verschiedenen Kulturen unterschiedlich aufgefasst. Manchmal wird die 0 dazugezählt und manchmal nicht. Wir haben also als Kandidaten für die Natürlichen Zahlen die Mengen $\{0, 1, 2, 3, \dots\}$ und $\{1, 2, 3, \dots\}$. Erstere Menge sei auf dieser Seite im folgenden mit \mathbb{N}_0 bezeichnet, letztere mit \mathbb{N} .

Man erhält somit in Bezug auf die Vollständigkeit unterschiedliche Aussagen über den Algorithmus des Euklid, je nachdem, welche der beiden Mengen (\mathbb{N}_0), der natürlichen Zahlen mit Null oder \mathbb{N} , der natürlichen Zahlen ohne Null) man als Menge der natürlichen Zahlen auffasst:

Über der Menge der \mathbb{N} ist der Algorithmus des Euklid vollständig, da zwei zufällig ausgewählte Zahlen stets durcheinander geteilt werden können. Wie oben gezeigt, spielt es dabei keine Rolle, welche der beiden Zahlen die größere ist.

Über der Menge der \mathbb{N}_0 ist der Algorithmus des Euklid nicht vollständig, da die Division immer dann nicht durchführbar ist, wenn die zweite zufällig ausgewählte Zahl die Null ist. Es gibt also (sogar unendlich viele) Kombinationen der Eingabedaten, auf die der Algorithmus nicht anwendbar ist.

Es ist also wichtig, bei der Beurteilung der Vollständigkeit eines Algorithmus die Grundmenge der Eingabedaten sorgfältig in Betracht zu ziehen.

6.1.1 Theoretische Fragestellungen zu Algorithmen

In der Informatik steht der Begriff des Algorithmus nicht einfach so im Raum, sondern es werden die Eigenschaften einzelner Algorithmen, die einem bestimmten Zweck dienen, untersucht. Aber auch Untersuchungen von Algorithmen an sich oder von Mengen ähnlicher Algorithmen werden angestellt. Auf dieser Seite werden zunächst mögliche theoretische Fragestellungen, ebenso wie Beispiele dazu, aufgeführt. Dann wird für eine der Fragestellungen gezeigt, wie diese für einen gegebenen Algorithmus beantwortet wird.

Die Theorie der Algorithmen beschäftigt sich mit den folgenden Fragestellungen:

- Welche Probleme sind mit Algorithmen lösbar?

Berechenbarkeit eines Problems (Effektivität des Verfahrens, ist das Verfahren überhaupt ausführbar?)

- Wie leistungsfähig ist ein Algorithmus?

Komplexität, Effizienz (Speicherplatz, Rechenzeit des Algorithmus)

- Sind zwei Algorithmen gleichwertig?

Äquivalenz (gleiche Eingaben ergeben gleiche Ausgaben)

Es folgen Beispiele zu den eben genannten Fragestellungen.



Beispiel

Das Halteproblem als Beispiel für ein nicht berechenbares Problem

Problem: Gibt es einen Algorithmus T , der für einen beliebigen anderen Algorithmus A oder sich selbst mit beliebigen Eingaben I entscheidet, ob der Algorithmus A terminiert?

Lösung: Es gibt keinen solchen Algorithmus. Die Beweisidee wird hier nur angedeutet: Es wird angenommen, dass es einen Test T_1 gäbe, der "ja" zurückliefere, wenn irgendein beliebiges anderes Programm P mit seiner Eingabe E , die beide von T_1 analysiert werden, hält. Daraufhin wird ein zweiter Test T_2 definiert, der weiterläuft, wenn ein Programm, angewendet auf sich selbst, hält, und der "ja" zurückliefert, wenn ein Programm angewendet auf sich selbst nicht hält. Dann wird T_2 auf sich selbst angewendet. Dies führt in beiden Fällen zum Widerspruch: Hält T_2 nicht, so müsste es, angewendet auf sich selbst, "ja" ausgeben. Hält T_2 jedoch und gibt "ja" aus, so müsste es, angewendet auf sich selbst, weiterlaufen. Somit kann es also den Test T_1 und damit T nicht geben. (Anmerkung: Dieses Beispiel soll hier nur den Sachverhalt verdeutlichen, dass die theoretische Informatik Probleme kennt, für die keine Lösung programmiert werden kann.)



Beispiel

Komplexität von Algorithmen zur Berechnung der Summe aufeinanderfolgender Zahlen

Problem: Gegeben seien zwei Algorithmen S_1 , und S_2 , die folgendes tun:

S_1 : Im Speicher s steht zu Beginn eine 0. Eine Reihe von aufeinanderfolgenden Zahlen wird sukzessive eingelesen und jeweils im Speicherplatz z abgelegt. Der Inhalt des Speicherplatzes z wird zu dem Wert im Speicherplatz s addiert und das Ergebnis im Speicherplatz s abgelegt. Wenn alle Zahlen eingelesen sind, wird der Wert des Speicherplatzes s als Ergebnis ausgegeben

S_2 : In einem Speicherplatz n steht zu Beginn eine 0. Eine Reihe von aufeinanderfolgenden Zahlen wird sukzessive eingelesen. Die jeweils neu eingelesene Zahl wird im Speicherplatz n abgelegt. Wenn alle Zahlen eingelesen sind, wird der Wert folgenden Ausdrucks berechnet: $(n \cdot (n + 1))/2$, wobei für n der Wert aus dem Speicherplatz n eingesetzt wird. Das Ergebnis dieser Berechnung wird im Speicherplatz n abgelegt und als Ergebnis ausgegeben. Es ist sichergestellt, dass die Eingabedaten eine Reihe direkt aufeinander folgender natürlicher Zahlen sind, die mit 1 beginnt. Wieviele Rechenschritte und wieviel Speicherplatz brauchen die beiden Algorithmen S_1 bzw. S_2 ?

Lösung: Speicherplatz: Der Algorithmus S_1 braucht die zwei Speicherplätze s und n , der Algorithmus S_2 nur einen Speicherplatz, n . Rechenzeit: Der Algorithmus S_1 braucht soviele Additionen, wie Zahlen eingelesen werden, also n Additionen, wenn n der Wert der letzten Zahl ist. Ausserdem wird das Ergebnis der Additionen jeweils, also auch n -mal, gespeichert. Der Algorithmus S_2 speichert n -mal die eingelesene Zahl, führt dann eine Addition, eine Multiplikation und eine Division aus. Da Multiplikation und Division meist mehr Zeit kosten als eine Addition, kann der Algorithmus S_2 bei kleinen Zahlenketten also möglicherweise mehr Rechenzeit verbrauchen als S_1 . Allerdings wird sich bei großen Zahlenketten durchsetzen, dass S_1 , zusätzlich zum Einlesen der jeweiligen Zahl, noch n Additionen ausführt. Für große Zahlenfolgen ist der Algorithmus S_2 also besser.

Ein weiteres Beispiel mit ausführlicher Bestimmung der (Zeit-) Komplexität findet sich im Abschnitt Komplexität weiter unten auf dieser Seite.



Beispiel

Gleichwertigkeit von Algorithmen zur Berechnung der Summe aufeinanderfolgender Zahlen

Problem: Wieder seien zwei Algorithmen S_1 , und S_2 gegeben. Es sind dieselben wie im vorigen Beispiel. Es ist erneut sichergestellt, dass es sich bei den Eingabedaten um eine Reihe direkt aufeinander folgender natürlicher Zahlen handelt, die mit 1 beginnt. Sind die beiden Algorithmen gleichwertig?

Lösung: Die beiden Algorithmen sind für die erlaubten Eingabedaten gleichwertig. Der Mathematiker Carl-Friedrich Gauß hat gezeigt, dass der Ausdruck $(n \cdot (n + 1))/2$ den gleichen Wert hat wie die Summe der natürlichen Zahlen von 1 bis n .

Zu beachten ist noch, dass in dem vorherigen und dem nachfolgendem Beispiele zwar dasselbe Paar zweier Algorithmen zur Bestimmung einer Summe aufeinander folgender Zahlen verwendet wurde; jedoch wurden die beiden Algorithmenpaare in den beiden Beispielen aus unterschiedlichen Blickwinkeln (unter unterschiedlichen Fragestellungen) betrachtet. Die Fragestellungen waren Komplexität (vorheriges Beispiel) und Gleichwertigkeit (im nachfolgendes Beispiel).

In beiden Beispielen wurden die Fragestellungen beinahe intuitiv beantwortet. Die folgende Betrachtung zur Komplexität von Algorithmen soll zeigen, dass theoretische Fragestellungen zu Algorithmen meist jedoch formaler untersucht werden.

Komplexität

Bei der Bestimmung der Rechenzeit eines Algorithmus wird als Maß die Anzahl der benötigten Rechenschritte in Abhängigkeit von der Eingabe herangezogen. Rechenschritte mit sehr geringem Zeitbedarf werden dabei häufig vernachlässigt (z.B. Addition gegenüber Multiplikation). Sehr großen Einfluss auf die Rechenzeit eines Algorithmus hat die Anzahl von Schleifendurchläufen. Es ist üblich, die Komplexität eines Algorithmus bzgl. seiner Schleifendurchläufe in Abhängigkeit von der Eingabe auszudrücken.

Drei Fälle werden bei Komplexitätsbetrachtungen unterschieden:

- Rechenzeit im besten Fall (best case, BEC)
- Rechenzeit im schlechtesten Fall (worst case, WOC)
- Rechenzeit im Mittel (average case, AVC)

Die unterschiedlichen Fälle werden über alle zulässigen Eingaben des Algorithmus bestimmt. Bester und schlechtester Fall sind oft einfach zu bestimmen. Die Berechnung der mittleren Rechenzeit ist häufig wesentlich aufwendiger. Dabei ist die Verteilung der Eingaben zu berücksichtigen.



Beispiel

Betrachtung der Anzahl der Schleifendurchläufe beim Algorithmus des Euklid zur Bestimmung des größten gemeinsamen Teilers

Betrachte als Beispiel die Anzahl der Schleifendurchläufe des Euklidischen Algorithmus (untenstehende Abbildung):

K Schritte zur Bestimmung von $\text{ggT}(m,n)$ ($m \geq n$). Die Nummer des jeweiligen Schleifendurchlaufs sei mit i bezeichnet. Wie bereits bei der Vorstellung des Algorithmus angemerkt wurde, gilt bei jedem Schleifendurchlauf: $m_i \div n_i = q_i + r_i$, wobei \div die ganzzahlige Teilung bezeichnet. Dementsprechend gilt bei jedem Schleifendurchlauf auch:

$$m_i = q_i \cdot n_i + r_i.$$

Betrachtet wird also nun ein exemplarischer Durchlauf. Bei jedem Schleifendurchlauf gilt in Schritt 2 dementsprechend:

$$m_1 = q_1 \cdot n_1 + r_1 \text{ mit } m_1 = m \text{ und } n_1 = n$$

$$m_2 = q_2 \cdot n_2 + r_2 \text{ mit } m_2 = n_1 \text{ und } n_2 = r_1.$$

\vdots

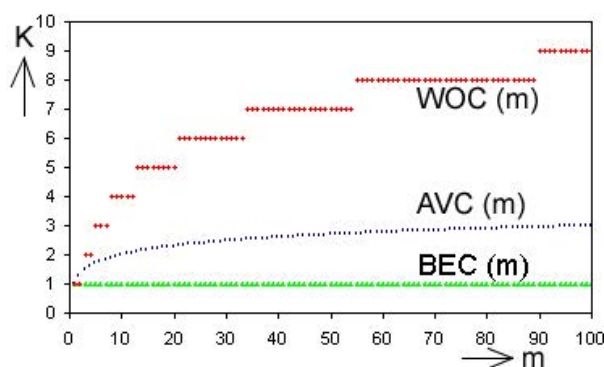
$$m_i = q_i \cdot n_i + r_i$$

$$m_{i+1} = q_{i+1} \cdot n_{i+1} + r_{i+1} \text{ mit } m_{i+1} = n_i \text{ und } n_{i+1} = r_i$$

\vdots

$$m_k = q_k \cdot n_k + r_k \text{ mit } r_k = 0 \text{ und damit } n_k = \text{ggT}(m,n)$$

Durch formale Untersuchungen kann gezeigt werden, dass die Anzahl der Schleifendurchläufe K nur von der (größeren) Zahl m abhängt und sich für den best case, average case und worst case wie folgt entwickelt:



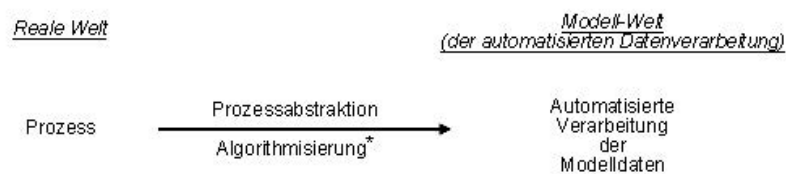
Rechenzeitbestimmung

(Wenn m das kleinere der beiden Argumente des Algorithmus des Euklid ist, so ist für k jeweils 1 dazu zu addieren, wegen der

einen zusätzlichen Vertauschung beim ersten Schleifendurchlauf.)

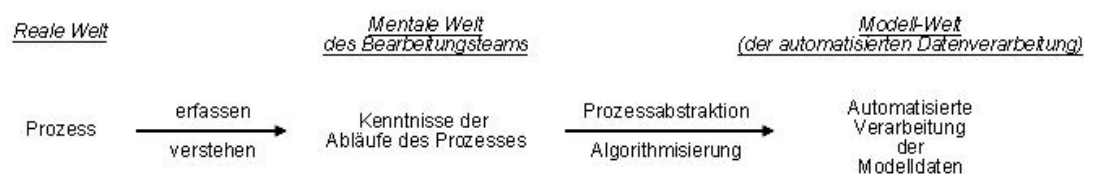
6.1.2 Algorithmisierung

Wie bereits im Unterkapitel Simulation dargestellt wurde, ist unter Algorithmisierung die Umsetzung eines Prozesses aus der realen Welt in die Beschreibung eines Ablaufes in der Modell- Welt (der automatisierten Datenverarbeitung) zu verstehen, welche von dem automatisierten Verfahren ausgeführt (interpretiert) werden kann. Die untenstehende Abbildung verdeutlicht noch einmal diesen Zusammenhang:



Algorithmisierung

Wichtig ist hierbei, dass es sich bei obiger Darstellung um eine idealisierte Sichtweise handelt, während in der Praxis oft folgende Situation (untenstehende Abbildung) vorkommt:



Algorithmisierungspraxis

Hierbei zeigt sich auch, dass im Bearbeitungsteam (Entwicklungsteam) zwei unterschiedlichen Fähigkeiten besondere Bedeutung zukommt. Erstens, der Analyse, also dem Erfassen und Verstehen des Prozesses in der realen Welt. Zweitens, der Algorithmisierung, also dem Formulieren der schrittweisen Vorgehensweise für den automatisierten Verarbeitungsprozess. Anders ausgedrückt: Die Qualität sowohl des Erkennens als auch des Darstellens tragen entscheidend zur Qualität eines Algorithmus bei.

Strukturierung und Entwurf von Algorithmen

Der Weg vom Problem zum Algorithmus kann auf zwei grundlegende Arten erfolgen:

- Top - down:

Das Problem wird (hierarchisch) in Teilprobleme zerlegt bis die einzelnen Probleme durch elementare Anweisungen lösbar sind. Die Lösung jedes Teilproblems trägt zur Lösung des Gesamtproblems bei. (Die Grundidee stammt von E. W. Dijkstra, 1969, und N. Wirth, 1971)

- Bottom - up:

Ausgehend von einer konkreten Rechenmaschine wird durch sukzessives Hinzufügen bestimmter Funktionalitäten und Eigenschaften eine abstrakte Maschine entwickelt, bis man bei der Maschine anlangt, die das gewünschte Problem löst. Beispiel: Maschinenbefehle, Assembler, höhere Programmiersprache, Anwendungsprogramm

Für beide Ansätze gibt es diverse Zerlegungsarten:

- Funktionsorientierte Zerlegung

Ausgehend von den funktionalen Anforderungen der Aufgabe erfolgt eine aufgabenorientierte Zerlegung des Gesamtprogramms in Prozeduren.

- Datenorientierte Zerlegung

Die Struktur eines Programmsystems soll die Struktur der zu verarbeitenden Daten widerspiegeln.

- Modulatorientierte Zerlegung

Programmsystem: Sammlung von Modulen
Module: Vereinigung von Daten und den darauf ausführbaren Operationen

- Objektorientierte Zerlegung

Programmsystem: Sammlung von miteinander kommunizierenden Objekten. Jedes Objekt verfügt über von außen nicht sichtbare Datenstrukturen und den darauf ausführbaren Operationen.

Anmerkung: In der Praxis können durchaus Mischformen aus verschiedenen Zerlegungsarten vorkommen, zum Beispiel, dass das Zerlegen des Programmsystems modulatorientiert erfolgt, während die einzelnen Module jedoch objektorientiert und die Methoden der einzelnen Objekte wiederum funktionsorientiert zerlegt werden.

Programmierparadigmen versus Programmiersprachen

Beim Aufschreiben von Algorithmen (beim Programmieren) gibt es prinzipiell unterschiedliche Arten, dieses zu tun. Diese unterschiedlichen Arten nennt man Programmierparadigmen. Diese sind als Reinformen einer Notationsweise zu verstehen. Solche Programmierparadigmen sind:

Imperative Programmierung

Diese (befehlsorientierte) Schreibweise basiert auf dem Grundgedanken, dass Speicherinhalte / Werte durch Anweisungen (statements) modifiziert werden. Die Speicherinhalte bzw. Werte sind durch die Speicheradressen bzw. Variablennamen im Speicher auffindbar und können so von anderen Anweisungen weiter verarbeitet werden.

Beispiel: $s := a + 5$; dem Speicher, der durch den Variablennamen s repräsentiert wird, wird als neuer Inhalt die Summe aus dem Speicherinhalt, der durch den Variablennamen a gefunden wird, und der Zahl 5 zugewiesen.

Funktionale Programmierung

Diese Schreibweise ist stark an die Funktionennotation der Mathematik angelehnt. Durch Funktionen (mit 0 bis n Parametern) werden Werte berechnet, die ausgegeben oder gespeichert werden oder ihrerseits als Parameter für andere Funktionen dienen.

Beispiel: `drucke(ggt(30,55))`, wobei `drucke` eine Funktion mit einem Argument und `ggt` eine Funktion mit zwei Argumenten ist. Die Funktion `drucke` schickt ihr Argument an den Drucker, und die Funktion `ggt` berechnet den größten gemeinsamen Teiler ihrer Argumente.

Objektorientierte Programmierung

Diese Schreibweise ist angelehnt an unsere Alltagsicht auf die Welt. So sind z. B. Fahrräder (Klasse) gleichzeitig auch Zweiräder (Klasse). Bei beiden Klassen handelt es sich um abstrakte Klassen; das heißt, dass von ihnen keine Instanzen (Objekte) erzeugt werden können. Auch die Unterklasse von Fahrrad, Rennrad, ist dementsprechend noch eine abstrakte Klasse. Wohingegen das vor mir stehende Fahrrad (Objekt) ein `RennradModellXY` (Unterklasse von Rennrad) ist. Diese Klasse ist nun keine abstrakte Klasse, da die spezifischen Rennradmodelle gebaut werden können (existieren, instanziiert sind). Objekte (oder Klassen) können eine bestimmte Funktionalität bereitstellen. Beispielsweise kann man bei einem Fahrrad treten und lenken. Wichtig ist noch, dass übergeordnete Klassen ihre Eigenschaften (Attribute entspricht Daten; Methoden entspricht Funktionalität) an abgeleitete (untergeordnete) Klassen vererben.

Anschaulich: Klassen können als Konstruktionsstätte der Objekte aufgefasst werden.

(Anmerkung: Die Bezeichnung "abstrakt" hat, wenn sie im Zusammenhang von Klassen benutzt wird, eine andere Bedeutung, als im Kontext des "Abstrakten Datentyps", der in den folgenden Unterkapiteln erläutert wird.) Beispiel:

**Definition**

```
Klasse Bruch{ Zahl zaehler; Zahl nenner; ist_äquivalent_zu;  
(Bruch bru){...; ...};
```

Verwendung:

```
Bruch B1 = neu Bruch(2,5);
```

```
Bruch B2 = neu Bruch (4, 10);
```

```
B1.ist_äquivalent_zu(B2);
```

In obigem Beispiel wird die Klasse Bruch definiert, welche die Attribute Zähler und Nenner bereitstellt. Außerdem besitzt sie die Methode einen Bruch, der als Parameter übergeben wird, mit dem Objekt selbst zu vergleichen. Der Bruch B1 (Objekt) wird als 2/5 erzeugt, der Bruch B2 analog als 4/10. Der Bruch B1 testet mit Hilfe der (eigenen) Methode B1.ist_äquivalent_zu, ob der als Parameter übergebene Bruch B2 zu B1 äquivalent ist.

Deklarative Programmierung

Bei der Deklarativen Programmierung werden Fakten und logische Aussagen im Speicher abgelegt. Außerdem verfügt die Programmierplattform über ein eingebautes Verfahren zur logischen Beweisführung, die so genannte Inferenzmaschine. Bei einer Anfrage an das System mittels einer logischen Aussage versucht das System, diese Aussage zu beweisen. Während der Beweisführung findet das statt, was man auf konventionellen Systemen als Programmlauf bezeichnet.

Bekannte Programmiersprache: ProLog.

Es gibt noch weitere Programmierparadigmen, von denen hier nur noch genetische Algorithmen und neuronale Netze genannt seien. Bei Genetischen Algorithmen mutieren geeignet codierte Algorithmen, kreuzen sich und erzeugen neue Algorithmen. "Bessere" Algorithmen vermehren sich leichter. Bei Neuronalen Netzen sind einzelne Prozessoreinheiten, (eines neuronalen Netzes) hochgradig mit anderen Prozessoreinheiten des neuronalen Netzes vernetzt. Einzelne Prozessoreinheiten

"feuern" (d.h. aktivieren Ihre Ausgänge), wenn ein bestimmter innerer Schwellenwert in Abhängigkeit der an den Eingängen anliegenden Werte überschritten ist. Das Neuronale Netz ist fast immer so konzipiert, dass es sich nach einiger Zeit in einen stabilen (auch nicht periodisch schwingenden) Zustand einschwingt. Wenn dies so ist, dann lernt ein neuronales Netz in einem Lernmodus anhand von angelegten Eingabe- und Ausgabepaaren, welche davon jeweils zusammengehören. Liegen später in einem Betriebsmodus bestimmte Eingaben am Neuronalen Netz an, so "assoziiert" es die ähnlichste gelernte Ausgabe. Achtung: Es kann nicht nicht assoziieren! Das heißt, auch wenn an der Eingabe bedeutungslose Daten anliegen, wird eine Ausgabe erzeugt. Die "Programmierung" eines Neuronalen Netzes besteht also im Finden einer geeigneten Netztopologie, einer geeigneten Codierung für die Ein- und Ausgaben sowie dem Bereitstellen guter Lernbeispiele.

Kontrollstrukturen

Vor allem in den ersten drei der oben genannten Programmierparadigmen gibt es bestimmte Kontrollstrukturen. Kontrollstrukturen legen fest, in welcher Reihenfolge die ausführbaren Einzelschritte einer Programmiersprache ausgeführt werden. (Anmerkung des Autors: In diesem Kapitel wird davon ausgegangen, dass Anweisungen, die in einer Notation aufgeschrieben sind, direkt ausgeführt werden. Man spricht dabei jeweils von einer abstrakten Sprachmaschine, die genau die Anweisungen der jeweiligen Notation versteht.) Beispiele für Kontrollstrukturen sind:

Einzelne Anweisungen nacheinander ausführen (serielle Anweisungen)

Dies ist intuitiv verständlich: Erst wird ein Wert berechnet, dann an den Drucker geschickt.

Einzelne Anweisungen gleichzeitig ausführen (parallele Anweisungen)

Das Aufkochen der Milch (CP5), das Verquirlen der Verquirl-Milch mit dem Ei (CP3), und das Schlagen des Eischnees (CP4) können in Beispiel 5.2 von drei Köchen gleichzeitig durchgeführt werden.

Einzelne Anweisungen nur unter bestimmten Bedingungen ausführen (bedingte Anweisungen)

Wenn der Aschenbecher des Autos voll ist, kaufe ein neues Auto, andernfalls (sonst) mache eine Spritztour mit Person XY.

Sich wiederholende Anweisungen (Iterationen)

Diese kommen in verschiedenen Variationen vor, z. B.:

Solange eine Bedingung (nicht) erfüllt ist, wiederhole: Ende von wiederhole;
wiederhole: Ende von wiederhole, solange eine Bedingung (nicht) erfüllt ist; Für eine bestimmte Menge/Anzahl, wiederhole: Ende von wiederhole;

Solche Kontrollstrukturen lassen sich z. B. in Form von Flussdiagrammen einheitlich darstellen. In den verschiedenen Programmiersprachen werden sie jedoch leicht unterschiedlich aufgeschrieben, und es sind auch nicht immer alle der oben genannten vorhanden. Um an dieser Stelle einer Diskussion der verschiedenen Programmiersprachen aus dem Wege zu gehen, wird hier eine

Notation in so genanntem Pseudocode vorgestellt. Der Pseudocode ist dabei teilweise an die natürliche Sprache angelehnt und sollte intuitiv verständlich sein.

6.2 Datenstrukturen



Gliederung

6.2 Datenstrukturen

6.2.1 Abstrakte Datentypen

6.2.2 Dynamische Datenstrukturen

Motivation

Bei der Erläuterung des Simulationsbegriffes im Unterkapitel Simulation wurde beschrieben, dass eine Ausgangssituation in der realen Welt durch Abstraktion in Eingabedaten, die der automatisierten Datenverarbeitung zugeführt werden können, überführt wird. Damit stellt sich die Frage: Wie sehen solche Daten überhaupt aus?

Nun, dies hängt hauptsächlich von der gewählten Beschreibungsform (im beruflichen Alltag eines Programmierers: von der Programmiersprache) ab. Dieses hat den Nachteil, dass alle Beteiligten, die über eine Datenstruktur kommunizieren wollen, diese Programmiersprache kennen sollten. Um von dieser Sprachgebundenheit wegzukommen, können Daten in der Informatik auch mit Hilfe von (abstrakten) Datentypen, die programmiersprachenunabhängig sind, aufgeschrieben werden. Die Notation ist dabei an die Mathematik angelehnt.

So gibt es zum Beispiel die Basisdatentypen, die in vielen Programmiersprachen aufgeschrieben werden können. Aus diesen können strukturierte Datentypen erzeugt oder, z. B. durch Aneinanderreihung von Daten eines (Basis-) Datentyps, zusammengesetzte Daten erzeugt werden. Fügt man zu den eben genannten Datentypen noch Abbildungen (im Sinne der Mathematik) auf diesen Datentypen hinzu, so ist der Schritt zu den Abstrakten Datentypen vollzogen.

Die genaue Erklärung der abstrakten Datentypen erfolgt in diesem Unterkapitel vor den Beispielen zu den Basisdatentypen. Ein einfaches Beispiel zu anderen abstrakten Datentypen, die in vielen Programmiersprachen nicht enthalten sind, sind Brüche.

Benutzt man nun einen "kleinen" abstrakten Datentyp, um aus ihm eine "höherwertige" Datenstruktur zusammenzusetzen, so kommt man z. B. zu den dynamischen Datenstrukturen. Von diesen werden im Rahmen dieses Moduls Listen und Bäume ausführlich vorgestellt.

Weitere Betrachtungen zum Sortieren (in einem Array) runden dieses Kapitel ab.

6.2.1 Abstrakte Datentypen

Motivation

Gehen wir noch einmal zurück zu den positiven ganzen Zahlen, also den Integer-Zahlen und deren Repräsentation durch eine feste Anzahl von Bytes im Speicher. Im Folgenden wird das vorhandene Wissen noch einmal informal dargestellt:

- Eine feste Anzahl Bytes repräsentiert eine positive ganze Zahl (einschließlich der 0).
- Wird die Byte-Darstellung einer Zahl zu einer anderen auf festgelegte Weise addiert, so erhält man im Normalfall wieder die Byte-Darstellung einer positiven ganzen Zahl.
- Unter bestimmten Bedingungen tritt bei der Addition der Zahlen ein Überlauf auf.
- Weitere Operationen besitzt dieser Datentyp nicht.

Wenn jetzt noch die Termini "eine feste Anzahl Bytes", "auf festgelegte Weise addiert", "im Normalfall" und "unter bestimmten Bedingungen" genau spezifiziert werden, ist komplett festgelegt, wie sich dieser Datentyp verhält, bzw. was man mit Ihm rechnen kann.

Aus dieser Betrachtung kommt man zu der Grundidee, dass die exakte Beschreibung der Repräsentation von Daten, ZUSAMMEN mit der Beschreibung ALLER auf Ihnen ausführbaren Abbildungen, als eine Einheit aufgefasst wird. Diese Einheit nennt man einen Abstrakten Datentyp.

(Anmerkung: Die Abbildungen können eines oder mehrere Elemente des Datentyps zu einem neuen Funktionswert abbilden. Dabei impliziert das Wort "auf"" bei "auf Ihnen ausführbaren Operationen", dass der Funktionswert der Abbildung ebenfalls ein Element des Datentyps ist.)

Das Ziel eines abstrakten Datentyps ist:

Eine Beschreibung für eine Menge von Daten zusammen mit all ihren Operationen bereitzustellen. Dabei ist diese Beschreibung unabhängig von einer späteren Implementierung in einer beliebigen, existierenden und konkreten Programmiersprache.

(Anmerkung: Ein abstrakter Datentyp der Informatik entspricht also ungefähr einer Algebra in der Mathematik.)

Betrachtet werden nun logische (boolesche) Werte: In der (Aussagen-)Logik gibt es die booleschen Werte wahr (w) und falsch (f). Außerdem sind oft drei logische Operationen auf ihnen definiert: Nicht, und, oder (not, and, or respektive \neg , \vee , \wedge).



Beispiel

Logische Werte als abstrakter Datentyp

Problem: Wie ist ein abstrakter Datentyp für logische Werte zu definieren?

Lösung: Als abstrakter Datentyp wird dies so geschrieben:

$B = \{\text{wahr}, \text{falsch}\}$ B ist die Grundmenge der beiden angegebenen

Wahrheitswerte. Es folgen die Operationen auf der Grundmenge B oder auf $B \times B$:

$\neg: B \rightarrow B: b \mapsto \neg b$, wobei; $\neg \text{wahr} = \text{falsch}$, $\neg \text{falsch} = \text{wahr}$,

$\wedge: B \times B \rightarrow B: (b_1, b_2) \mapsto b_1 \wedge b_2$ gemäß untenstehender Wahrheitstafel,

\wedge :

		b_1	
b_2		wahr	falsch
	wahr	wahr	falsch
	falsch	falsch	falsch



Wahrheitstafel logisches und

$\vee: B \times B \rightarrow B: (b_1, b_2) \mapsto b_1 \vee b_2$ gemäß untenstehender Wahrheitstafel.

\vee :

b_1			
b_2		wahr	falsch
	wahr	wahr	wahr
	falsch	wahr	falsch



Wahrheitstafel logisches oder

Das 4-Tupel (B, \neg, \wedge, \vee) ist dann ein abstrakter Datentyp.

Hier ist der abstrakte Datentyp relativ formal aufgeschrieben. In der SW-Entwicklungspraxis werden meist nur die Grundmenge und die Operationen angegeben (und nicht das Tupel des resultierenden Datentyps). Außerdem gilt ein abstrakter Datentyp in der Praxis der SW-Entwicklung auch dann als abstrakter Datentyp, wenn in Einzelfällen eine geeignete Fehlermeldung definiert ist. (Zum Beispiel: Fließkommazahlen und die vier Grundrechenarten: Die Division durch null löst eine definierte Fehlermeldung aus. Die Fließkommazahlen mit den Operationen und der Fehlermeldung werden in der Praxis dennoch als abstrakter Datentyp bezeichnet, obwohl strenggenommen die Fehlermeldung als Operationsergebnis aus der Grundmenge der Fließkommazahlen herausführt.) Im weiteren Verlauf dieses Kapitels werden weitere abstrakte Datentypen vorgestellt.

6.2.1.1 Basisdatentypen

Wichtige Basisdatentypen, die in vielen Programmiersprachen vorkommen, sind:

Skalare Werte

Ganze Zahlen (im Rechner binär dargestellt):

Manchmal sind nur positive Werte zugelassen.

Typische Operationen sind: +, - (falls negative Werte dargestellt werden können), Multiplikation, ganzzahlige Division, etc. Einige dieser Operationen lösen gegebenenfalls Überläufe aus. Beispiele in bekannten Programmiersprachen: int in C, C++, Java, integer in PASCAL, u. a.

Fließkommazahlen in Mantisse-Exponent-Darstellung:

Typische Operationen sind:

+, -, Multiplikation, Division (ohne Null), aber auch oft (mit Hilfe von mathematischen Bibliotheken) wissenschaftliche Funktionen. Auch hier können Überläufe oder andere sinnvolle Fehlermeldungen ausgelöst werden. Beispiele in bekannten Programmiersprachen sind: real in PASCAL, float in C, C++, Java

Buchstaben und andere Schriftzeichen:

Typische Operationen sind: Codierungen oder Konkatenation. Codierungen bilden Zeichen oft auf Zeichen ab. Sie können aber auch aus diesem Datentyp hinaus in den Datentyp Zeichenkette führen. Konkatenation (Aneinanderhängen) führt immer in den Datentyp Zeichenkette. Demnach bilden Buchstaben und die Konkatenation keinen abstrakten Datentyp. (Zeichenketten und Konkatenation wiederum bilden allerdings schon einen abstrakten Datentyp).

Beispiele in bekannten Programmiersprachen sind: char in C, Java u. a.

Wahrheitswerte: True, False:

Typische Operationen sind: logisches und, logisches oder, logisches nicht

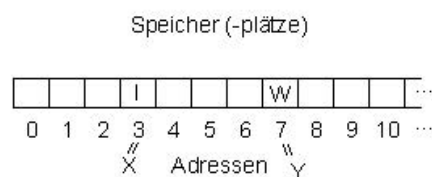
Beispiele in bekannten Programmiersprachen sind: bool in C, C++, Java, boolean in PASCAL

Byte (ein Wort aus normalerweise 8 Bit):

Typische Operationen sind: bitweises und, bitweises oder, bitweises nicht.

Adressen:

Was eine Adresse ist, ist in dem folgenden untenstehender Abbildung erläutert:



"I" und "W" repräsentieren die Speicherinhalte / Werte.
 "X" und "Y" sind die Bezeichner (Synonyme) für die Adressen 3 und 7.



Speicherung eines Wertes und seine Adresse

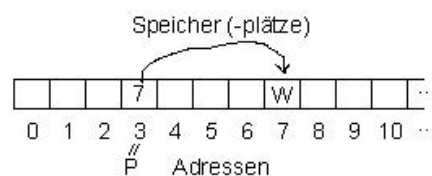
Hierbei wird angenommen, dass alle Speicherplätze linear nacheinander im Speicher angeordnet sind. Wenn ein Wert (hier: I, W) in einen Speicherplatz geschrieben wird, so ist zu diesem Speicherplatz die Adresse (hier: 3, 7) bekannt.

Mit Hilfe dieser Adresse kann der Inhalt des Speicherplatzes gelesen oder gesetzt werden.

Typische Operationen sind: Inkrementierung und Dekrementierung um einen bestimmten Distanzwert (Offset), der ein lineares Vielfaches von der Länge eines Wertes, der im Speicherplatz gespeichert ist, beträgt. Man berechnet so die Adresse eines anderen Speicherplatzes. (Achtung: Fehler bei der Berechnung von Adressen führen oft zu unerwartetem Verhalten eines laufenden Programms!) Beispiele in bekannten Programmiersprachen sind: Der Adressoperator in C, C++.

Pointer:

Das Konzept des Pointers sieht folgendermaßen aus:



"W" repräsentiert einen Speicherinhalt,
 "7" repräsentiert den Speicherinhalt der Variablen P
 (mit der Adresse 3) und ist selbst eine Adresse!



Speicherung eines Pointers auf einen Wert

In diesem Fall wird in einem Speicherplatz (hier zum Bezeichner P mit der Adresse 3) kein Wert abgespeichert, sondern die Adresse eines anderen Speicherplatzes (hier 7), in dem dann ein Wert gespeichert sein kann (hier "W"). Visuell kann man das auch durch einen Pfeil darstellen. Ein synonymes Wort für Pointer ist Zeiger. Zu beachten ist, dass dieses Konzept aneinanderreihbar ist. Das bedeutet, Pointer von Pointern die auf einen Pointer zeigen, der auf einen Wert zeigt sind möglich, aber wegen der hohen Anfälligkeit für Denkfehler der Programmierenden nur selten in der Praxis im Einsatz. Typische Operationen sind: Das Setzen und Ändern der im Pointer gespeicherten Adresse. Achtung: Hierbei gibt es einen Sonderfall! Wenn als Adresse ein bestimmter ausgezeichnete Wert eingetragen wird, der per Definition keine Adresse ist, so spricht man von einem Null- Pointer (manchmal auch NIL, oder void-Pointer). Es handelt sich hierbei um einen Pointer, der auf "gar nichts" zeigt. Auch die Werte von Pointern können inkrementiert oder dekrementiert (z. B. mit konstantem Offset), oder zueinander zuaddiert (seltener subtrahiert) werden. (In diesem Fall ist der Wert des zweiten Pointers oft der benötigte Offset.)

Typische Beispiele in bekannten Programmiersprachen sind: *P (Dereferenzierung eines Pointers) in C, C++; in Java werden Pointer implizit immer dann zur Speicherung genutzt, wenn nicht Werte aus Basisdatentypen, sondern Objekte gespeichert werden sollen. Anmerkung des Autors: Stellen Sie sicher, dass Sie das Konzept des Pointers verstanden haben, bevor sie die Inhalte zu den Themen Listen und Bäume lesen.

Strukturierte Werte

Strukturen

Hierbei werden andere Datentypen oder Strukturen in einer übergeordneten Struktur zusammengefasst. Auf die einzelnen Komponenten einer Struktur wird durch einen geeigneten Selektor zugegriffen. Strukturen repräsentieren ausschließlich Daten, keine Methoden. Achtung: Auch Pointer können Komponenten einer Struktur sein. Ein Beispiel für eine Struktur ist: Struktur Bruch = {Zahl Zähler, Zahl Nenner}. Angenommen man hat einen Bruch b, so kann man selektieren: b.Zähler respektive b.Nenner.

Typische Operationen sind: Das Lesen und Setzen der Komponenten einer Struktur. Typische Beispiele in bekannten Programmiersprachen sind: struct in C, C++

Klassen

Klassen besitzen analog zu Strukturen Komponenten, die Daten repräsentieren (sog. Attribute). Darüber hinaus können sie Komponenten haben, die Funktionalität beinhalten, die Methoden (in Form von Prozeduren, verschiedenen Arten von Funktionen). Auf die vielfältigen Möglichkeiten dieses objektorientierten Ansatzes wird an dieser Stelle nicht weiter eingegangen. Es sei nur soviel gesagt, dass sich Klassen sehr gut zur Implementierung von Abstrakten Datentypen (siehe auch das gleichnamige Unterkapitel) eignen.

Typische Operationen sind: Das Schreiben oder Lesen der Attribute sowie das Aufrufen der Methoden. Typische Beispiele in bekannten Programmiersprachen sind: class ... in C++, Java

Arrays

Die folgende Abbildung verdeutlicht, dass es sich bei Arrays (Feldern) um Aneinanderreihungen von Daten gleichen Typs handelt.

"B": Ein Buchstabe

B	E	R	N	D
15	16	17	18	19
// F				

Die Buchstaben "BERND" sind die Speicherinhalte des Arrays von Buchstaben mit dem Bezeichner "F". Es hat die Anfangsadresse 15 und die Endadresse 19.



Array F aus Buchstaben

Dabei sind die einzelnen Speicherplätze eines Arrays indiziert (durchnummeriert), meist beginnend mit 0. Die einzelnen Plätze eines Arrays können sowohl einfache als auch strukturierte Datentypen oder auch Pointer enthalten. Es gibt sowohl Arrays mit fester Länge als auch mit variabler Länge. Typische Operationen sind: Das Schreiben oder Lesen der Plätze eines Arrays. Wenn dies möglich ist, die Veränderung der Länge eines Arrays. Schreib- und Lesezugriffe mit Arrayindizes, die nicht innerhalb des Indexbereiches eines Arrays liegen, können zu unterschiedlichsten Fehlern beim Programmablauf führen. Im Idealfall wird nur eine Fehlermeldung, die auf den falschen Index hinweist, generiert. Schlimmer ist es meist, wenn die Werte trotz fehlerhaftem Index weiterhin gelesen oder gar geschrieben werden können.

Aufzählungstypen

Was ist in diesem Kontext eine Aufzählung? Eine Aufzählung ist eine nummerierte Reihenfolge, wie z. B.: 1, 2, 3, oder a, b, c, oder erstens, zweitens, drittens, viertens, oder andere Reihenfolgen. Wichtig ist, dass die Elemente eines Aufzählungstyps eine Reihenfolge haben und dass die Anzahl der Elemente endlich ist. (Anmerkung des Autors: In der Programmierpraxis werden eigentlich fast immer Aufzählungstypen mit wenigen Elementen genutzt, da sich für Aufzählungstypen mit vielen Elementen die positiven ganzen Zahlen nutzen lassen.) In der Definition von Aufzählungstypen werden daher diese Typen entweder durch ihre Reihenfolge definiert oder auf (kleine) positive ganze Zahlen abgebildet. Typische Operationen auf Aufzählungstypen sind: Lesen des Wertes; Aufzählungstypen werden oft nur definiert, um Ihre Elemente als Konstante für Vergleiche zu nutzen.

Anmerkungen

Obige Angaben erheben nicht den Anspruch auf Vollständigkeit.

Zu beachten ist noch, dass Vergleiche auf einem Datentyp oft nicht in oder auf den Datentyp abbilden, sondern in und auf Wahrheitswerte.

Bezeichner (Identifizier)

Bezeichner (Identifier) nehmen eine Sonderstellung ein. Sie sind meist KEIN Datentyp (Ausnahmen sind z. B. LisP-Dialekte, in denen sie als eine Form von so genannten Atomen sehr wohl ein Datentyp sind), sondern dienen bei der Algorithmusbeschreibungsnotation (Programmiersprache) als Platzhalter für zu definierende konstante oder variable Daten respektive konstante oder variable Zeiger auf Daten. Somit können Bezeichner als menschenlesbares Synonym für die Adresse eines Speicherplatzes aufgefasst werden. Darf der Speicherplatz nach dem ersten Schreiben nicht mehr geändert werden, so handelt es sich um eine Konstante, andernfalls um eine Variable. Dabei ist es belanglos, ob der Speicherplatz einen Datenwert oder einen Zeiger auf einen Datenwert enthält. In dem in obiger Abbildung dargestellten Beispiel eines Arrays, ist "F" der Bezeichner.

6.2.1.2 Andere abstrakte Datentypen



Beispiel

Brüche als abstrakter Datentyp

Problem: Die rationalen Zahlen (Brüche) sind als abstrakter Datentyp zu spezifizieren.

Lösung: Idee: Es gibt eine Klasse Bruch mit den Attributen Zähler und Nenner, die ganze Zahlen sind. Der Vereinfachung halber wird hier angenommen, dass ein Nenner niemals 0 ist. Auf dieser Klasse gibt es die Operationen $+$, $-$, \cdot , \div .

Klasse $\text{Bruch}\{\text{Zähler}, \text{Nenner}\}$ mit Zähler N_0 , Nenner N .

$+$: $\text{Bruch} \times \text{Bruch} \rightarrow \text{Bruch}$,

$$b_1 \times b_2 \mapsto \text{Bruch}(b_1.\text{Zähler} \cdot b_2.\text{Nenner} + b_1.\text{Nenner} \cdot b_2.\text{Zähler}, b_1.\text{Nenner} \cdot b_2.\text{Nenner})$$

$-$: $\text{Bruch} \times \text{Bruch} \rightarrow \text{Bruch}$,

$$b_1 \times b_2 \mapsto \text{Bruch}(b_1.\text{Zähler} \cdot b_2.\text{Nenner} - b_1.\text{Nenner} \cdot b_2.\text{Zähler}, b_1.\text{Nenner} \cdot b_2.\text{Nenner})$$

\cdot : $\text{Bruch} \times \text{Bruch} \rightarrow \text{Bruch}$,

$$b_1 \times b_2 \mapsto \text{Bruch}(b_1.\text{Zähler} \cdot b_2.\text{Zähler}, b_1.\text{Nenner} \cdot b_2.\text{Nenner})$$

\div : $\text{Bruch} \times \text{Bruch} \rightarrow \text{Bruch}$,

$$b_1 \times b_2 \mapsto \text{Bruch}(b_1.\text{Zähler} \cdot b_2.\text{Nenner}, b_1.\text{Nenner} \cdot b_2.\text{Zähler}) (\text{Bruch}, +, -, \cdot, \div)$$

ist nun der gesuchte abstrakte Datentyp.

Es bleibt den geneigten Lesenden überlassen, sich weitere abstrakte Datentypen zu überlegen.

6.2.2 Dynamische Datenstrukturen

Motivation

Auf den ersten Blick scheint hier etwas grundlegend Neues betrachtet zu werden. Dem ist jedoch nicht so. Betrachtet wird wieder ein abstrakter Datentyp. Seine Methoden sind jedoch so angelegt, das mit dem Datentyp etwas "höherwertiges" Aufgebaut werden kann. Dies soll zunächst an einem Alltagsbeispiel verdeutlicht werden:

Betrachten wir einen (vereinfachten) Eisenbahnwaggon mit seinen zwei Kupplungen und den Methoden, den Schließzustand der vorderen respektive der hinteren Kupplung zu wechseln (aus offen wird zu und umgekehrt). Bei einem einzelnen Eisenbahnwaggon kann man nun, solange man will, die vier möglichen Kupplungszustände durchwechseln.

Spannend wird das ganze, wenn zwei oder mehrere Waggon in geeigneter räumlicher Nähe (z. B. auf einem Gleisstück) so nahe beieinander stehen, dass sich deren Kupplungen berühren. Betrachtet man einen einzelnen Waggon, werden immer noch nur Kupplungszustände geändert; dennoch kann auf einer anderen Betrachtungsebene zum Verbinden und Lösen von Zugteilen kommen. (Eisenbahnfans mögen uns bitte gewisse technische Inkorrektheiten hier nachsehen.)

Das interessante (dynamische) an diesem Beispiel ist die Fähigkeit der Waggon, zu unterschiedlich langen Zugteilen verbunden werden zu können. Nachgetragen sei hier noch, dass man Waggon durchaus als Speicher für Personen oder Fracht auffassen kann.

Was kann man also in den Kontext von Datenstrukturen mitnehmen, wenn man von Waggon, Kupplungszuständen und Zugteilen abstrahiert?

Gebraucht wird ein Container entsprechen dem Platz im Waggon und mindestens eine Möglichkeit einen Container mit einem anderen zu verbinden.

Eine auf der abstrakten Ebene ähnliche Struktur wurde bereits im Kapitel 3 Modellierung im Unterkapitel Graphen betrachtet. Wir haben hier Knoten, entsprechend den Containern (Waggon), und (gerichtete) Kanten, entsprechend den Verbindungen (Kupplungen). Mit dieser Ausstattung lassen sich nun (recht komplizierte) Graphen erstellen.

In der Denke eines Datentyps ausgedrückt, brauchen wir also eine Struktur, die Speicher für Werte (oder Zeiger auf Speicher für Werte) bereitstellt sowie mindestens einen Zeiger auf ein anderes Element derselben Struktur und natürlich auch Methoden zum Lesen und Schreiben dieser Zeiger.

Bleibt noch die Frage, welcher Art die Strukturen sein sollen, die mit einem solchen Datentyp aufgebaut werden können, und inwiefern diese in der Informatik bzw. SW-Entwicklung von Nutzen sein können?

Mit dem hier gesagten im Hinterkopf kann man sich nun den Listen oder Bäumen zuwenden.

6.2.2.1 Listen



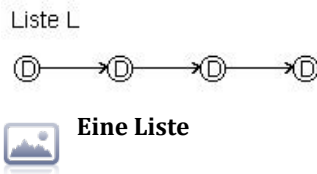
Hinweis

Sie sollten verstehen, dass auf alle Objekte, die man zum Aufbau oder zur Handhabung einer Liste braucht, also Listenkopf, Iterator und Listenknoten nur über Variablen oder Pointer zugegriffen werden kann. Deshalb muss bei allen Operationen auf oder mit diesen Objekten darauf geachtet werden, dass Objekte die später noch verwendet werden sollen stets über einen Variablennamen oder über einen Zeiger zugreifbar bleiben. Dies gilt insbesondere auch bei der Abarbeitung einer Operation, die aus mehreren Schritten besteht. Sie sollten bei den einfachen Operationen auf Listen, Einfügen, Löschen oder Verändern von Daten an festgelegten Stellen der Liste (Anfang, Ende oder bei einem Iterator) die Grundidee, den Pseudocode und vor allem auch die zeichnerische Darstellung verstehen.

Motivation

Dieses Unterkapitel soll die Grundidee der Liste, wie sie im Kontext der Informatik verstanden wird, verdeutlichen. Wir alle kennen Listen aus dem alltäglichen Leben in Form von z.B. Kassenbons, Einladungslisten (für Feiern), Stücklisten in der Produktion oder bei der Warenausgabe, etc.. Das besondere an solchen Listen ist, dass sie nahezu beliebig lang werden können. Traditionell handelt es sich dabei oft um untereinander geschriebene oder gedruckte Einträge auf Papier. Will man dieses Konzept der Liste in die Informatik bzw. in eine Datenstruktur übertragen, so stellt sich allerdings die Frage, wie man damit umgeht, dass man oft statt des Papiers nur Speicher(-elemente) hat, und ein Lesen des Speichers aus einer Draufsicht, so wie ein Mensch von oben (aus der 3. Dimension) auf ein Blatt Papier schaut, kann der Computer so ohne weiteres nicht durchführen. Man erhält also ein leicht abgewandeltes Konzept. Eine Liste wird durch

Aneinanderreihung von Listeneinträgen aufgefasst. (In der folgenden Abbildung steht "D" für Daten.):



Einfach verkettete Liste

Zunächst wird die Grundidee noch informal betrachtet:

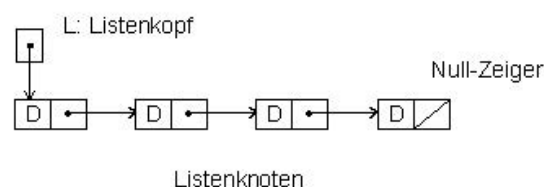
Eine Liste ist eine Aneinanderreihung von einzelnen Listenelementen. Dabei zeigt ein Listenkopf auf den ersten Listenknoten (wenn die Liste mindestens einen Knoten enthält), und jeder Listenknoten auf den Nächsten (wenn es einen nächsten Listenknoten gibt), oder der Zeiger ist leer. Anzumerken bleibt noch, dass der Listenknoten auch auf Daten Zugriff hat, da eine Liste, die keinerlei (Nutz-)Daten enthalten kann, keinen wirklichen Sinn macht. Die obige Abbildung verdeutlicht diesen Zusammenhang:

Realisierung

Bei der Realisierung einer Liste braucht man nur wenige Datentypen, nämlich:

einen Pointer auf einen Listenknoten (den Listenkopf), einen Listenknoten, der aus zwei Anteilen besteht: einem Objekt eines (Nutz-)Datentyps und einem Zeiger auf einen Listenknoten, und einen Null-Pointer.

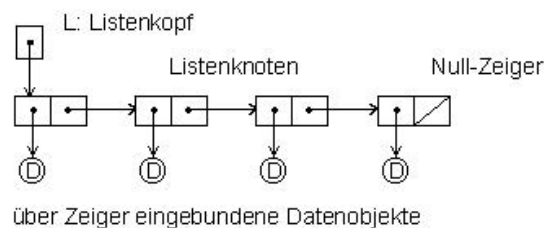
Die untenstehende Abbildung zeigt, wie aus diesen "Zutaten" die dynamische Datenstruktur Liste entsteht:



Naive einfach verkettete Liste

Eigentlich hat man in obiger Abbildung auch ohne den Listenkopf die Struktur der Liste im Speicher erzeugt. Allerdings könnte dann nicht darauf zugegriffen werden. Der Listenkopf stellt also die Zugriffsmöglichkeit auf die Liste bereit.

Anmerkung: Die zu speichernden Daten befinden sich bei der in obiger Abbildung gezeigten Liste direkt im Datentyp Listenknoten. Dies hat den Nachteil, dass für jeden zu speichernden Datentyp ein eigener Listenknotendatentyp, somit ein eigener Pointer auf den Listenknotendatentyp und somit eine eigene Datenstruktur Liste entsteht. Diesen Nachteil kann man bei objektorientierter Programmierung mit einem kleinen Trick leicht vermeiden:

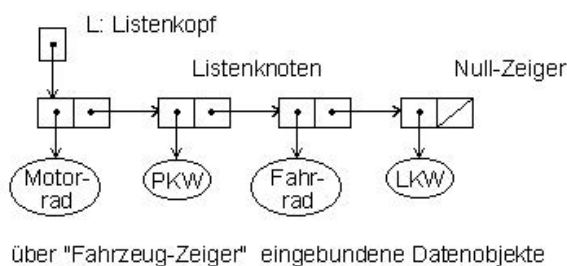


Einfach verkettete Liste

Bei diesem Ansatz ist der zweigeteilte Listenknoten anders aufgebaut. Er enthält

- einen Pointer auf einen Eltern-(Nutz-)Datentyp, von dem alle anderen Nutzdatentypen erben und
- einen Zeiger auf den Listenknoten.

Der Vorteil ist, dass der Zeiger auf den Eltern-(Nutz-)Datentyp auch auf alle von diesem Typ ererbenden Datentypen zeigen kann. So ist es möglich, mit einer Sorte Zeiger die Daten verschiedener Nutzdatentypen in der Liste aufzubewahren. Dies geht sogar gemischt in einer Liste. Die untenstehende Abbildung zeigt eine solche Liste am Beispiel des Eltern-Datentyps Fahrzeug und der ererbenden Datentypen PKW, LKW, Motorrad und Fahrrad:



Einfach verkettete Liste mit unterschiedlichen Datentypen

Anmerkung des Autors: Auf parametrisierbare Datentypen wird hier nicht weiter eingegangen. Bei den weiteren Betrachtungen zu Listen und Bäumen wird davon ausgegangen, dass die Daten immer über Zeiger eines Eltern-Nutz-Datentyps (ENDT) in jeweiligen Knotentypen eingehängt werden.

Listenknoten als abstrakter Datentyp:

Obige Abbildung stellt einen Listenknoten als zweiteilige Struktur dar. Der erste Teil ist ein Zeiger auf irgendeinen Eltern-Nutz-Datentyp, der zweite Teil ist ein Zeiger auf einen weiteren Listenknoten. Somit haben wir einen Datentyp Listenknoten. Die folgende informale Überlegung zeigt, wie der Datentyp Listenknoten als abstrakter Datentyp definiert werden kann.

- Die Menge aller Listenknoten sei nun die Grundmenge LN.
- Eine Operation (SE) auf den Elementen dieser Grundmenge sei das Setzen des ENDT- Zeigers,
- eine weitere das Setzen des LN-Zeigers (SL).

Mit diesen beiden Operationen kann man Listen aufbauen, aber die Inhalte der Listen sollen ja auch weiterverarbeitet werden können. Man braucht also noch:

- Eine Operation (LE) auf den Elementen der Grundmenge LN, das Lesen des ENDT-Zeigers und
- eine Operation (LL) auf den Elementen der Grundmenge LN, das Lesen des LN-Zeigers.

Wenn man davon ausgeht, dass die Operationen SE, SL, LE und LL als Ergebnis jeweils den betroffenen Listenknoten zurückgeben, so ist (LN, SE, SL, LE, LL) ein abstrakter Datentyp. Formal ist die Definition leider etwas aufwendiger, aber die hier dargestellte Grundidee stimmt. Somit ist am Beispiel der einfach verketteten Liste erläutert, wie ein relativ einfacher abstrakter Datentyp in der sinnvollen Verwendung eine höherwertigere Struktur, eine dynamische Datenstruktur bildet.

Intermezzo:

Bisher haben wir ein Art der Liste, die einfach verkettete Liste, betrachtet bei der man vom Listenknoten durch Lesen der Zeigerwerte immer nur in eine Richtung durch die Liste gelangen kann. Es ist leicht einsehbar, dass auch Modelle von Listen, bei denen man von einem Listenknoten in zwei Richtungen zu einem benachbarten Listenknoten gelangen kann, Sinn machen. Um dem gerecht zu werden werden auf dieser Seite beide Listentypen vorgestellt. Zunächst geht es damit weiter, dass für die einfach verkettete Liste einige Operationen beispielhaft beschrieben werden, dann

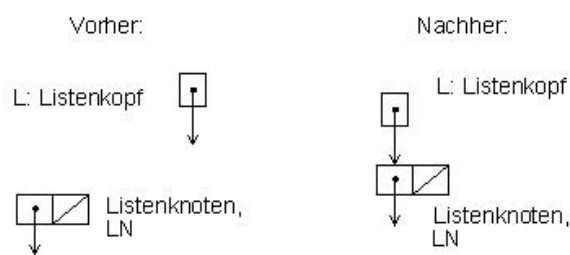
werden die für zweifach (doppelt) verkettete Listen erforderlichen Listenknoten und ebenfalls einige Operationen beschrieben.

Operationen auf einer einfach verketteten Liste:

Beispiele zu Operationen auf einer einfach verketteten Liste seien: (i) das Einfügen eines Listenknotens an beliebiger Stelle der Liste oder (ii) das Löschen eines beliebigen Listenknotens aus einer Liste (es gibt natürlich noch weitere.). Diese Operationen auf der Datenstruktur der einfach verketteten Liste werden dabei zurückgeführt auf Operationen im abstrakten Datentyp Listenknoten. Zum Beispiel: Das Setzen des Zeigers auf den Folgeknoten auf einen Wert, der auch Null sein kann.

Im Folgenden wird ohne Beschränkung der Allgemeinheit angenommen, dass es keine undefinierten Zeigerwerte gibt; ein Zeiger zeigt also entweder auf ein Objekt eines geeigneten Datentyps oder auf Null.

Einfügen des ersten Listenknotens



Einfügen eines ersten Listenknotens

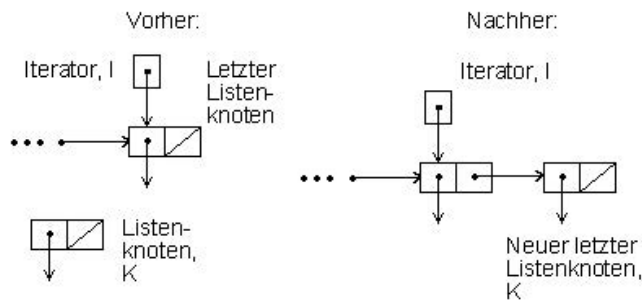
Grundidee:

Voraussetzungen: Der Listenzeiger zeigt auf Null. Ein Listenknoten ist vorhanden. Die Operation besteht aus:

1. Setze den Wert des Listenzeigers auf den Listenknoten. Weitere Operationsschritte sind nicht erforderlich. Pseudocode:

L points_to LN

Einfügen eines neuen letzten Listenknotens



Einfügen eines neuen letzten Listenknotens

Grundidee:

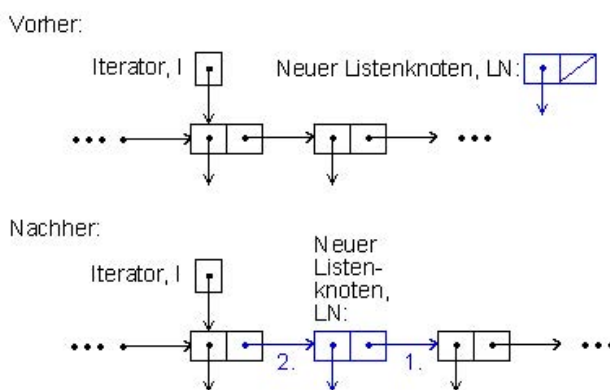
Voraussetzungen: Eine Liste enthält mindestens einen Listenknoten. Der Zeiger des letzten Listenknotens zeigt auf Null. Ein Iterator, I, zeige auf den letzten Listenknoten. Ein weiterer Listenknoten, K, ist vorhanden.

Die Operation besteht aus:

1. Setze den Nachfolgezeiger des Listenknotens, auf den der Iterator zeigt, auf den neu einzufügenden Listenknoten.

Weitere Operationsschritte sind nicht erforderlich.

Pseudocode (LN sei der Listenknoten, der an die Liste angehängt werden soll.):
 I.Listnode.Next points_to K; Einfügen eines Listenknotens zwischen zwei Listenknoten



Einfügen zwischen zwei Listenknoten

Grundidee:

Voraussetzungen: Eine Liste enthält mindestens zwei Listenknoten. Der Zeiger des letzten Listenknotens zeigt auf Null. Ein weiterer Listenknoten ist vorhanden. Ein Iterator I zeigt auf den Listenknoten, hinter dem ein neuer Listenknoten LN eingefügt werden soll. Der Listenknoten hinter dem eingefügt werden soll, hat einen Nachfolger.

Die Operation besteht aus:

1. Setze den Nachfolgerzeiger von LN auf den Nachfolgerzeiger des Listenknotens, auf den I zeigt.
2. Setze den Nachfolgerzeiger des Listenknotens, auf den der Iterator zeigt, auf den neu einzufügenden Listenknoten, LN.

Weitere Operationsschritte sind nicht erforderlich. Pseudocode (LN sei der Listenknoten, der an die Liste angehängt werden soll.): LN.Next points_to I.Listnode.Next; I.Listnode.Next points_to LN;

Einfügen eines Listenknotens an beliebiger Stelle der Liste

Grundidee: Die Grundidee besteht darin, diesen Einfügevorgang auf die drei bereits genannten Einfügevorgänge zurückzuführen.

Voraussetzungen: Eine Liste L sei gegeben. Es ist nicht bekannt, ob L (einen oder mehrere) Listenknoten enthält. Wenn sie (mindestens einen) Listenknoten enthält, so zeigt der Zeiger des letzten Listenknotens auf Null. Ein weiterer Listenknoten, LN, ist vorhanden. Es existiere eine Identifikation für Listenknoten.

Die Operation besteht aus:

1. Wenn L leer ist, dann führe Einfügen des ersten Listenknotens aus.
2. Andernfalls: Setze einen Zeiger (Iterator: I) auf den ersten Listenknoten.
3. Solange der Listenknoten, auf den I zeigt nicht identifiziert ist: Wenn der Nachfolgerzeiger des Listenknotens auf den I zeigt leer ist, dann melde, der zu identifizierende Knoten wurde nicht gefunden, Abbruch des Einfügens. Wenn nicht abgebrochen wurde, setze den Iterator auf den Nachfolgeknoten des aktuellen Listenknotens.
4. (Nun zeigt der Iterator I auf einen identifizierten Knoten)

Wenn der Nachfolgezeiger des Listenknotens, auf den der Iterator zeigt, leer ist, dann führe Einfügen eines neuen letzten Listenknotens aus. andernfalls führe Einfügen eines Listenknotens zwischen zwei Listenknoten aus.

Weitere Operationsschritte sind nicht erforderlich.

Pseudocode (LN sei der Listenknoten, der an die Liste angehängt werden soll, L sei eine Liste.):

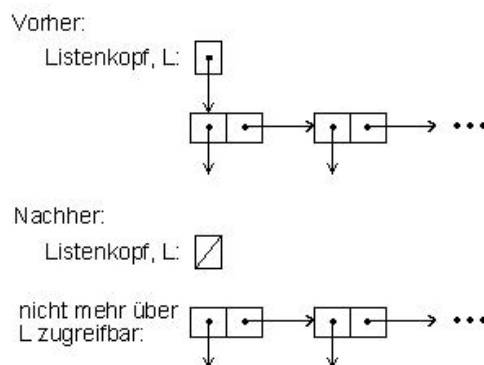
```
if L is_empty then L.Einfügen_des_ersten_listenknotens(LN) else // L ist nicht
leer Konstruiere Iterator I; I points_to L.First; While not I.Listnode.is_identified
do if I.Listnode.Next.is_empty then Meldung: Der zu identifizierende Listenknoten
```

konnte nicht gefunden werden. Abbruch des Einfügens; fi; I points_to I.Listnode.Next; od; // I.Listnode ist der zu identifizierende Knoten! (dies ist ein Kommentar) if I.Listnode.Next.is_empty; then I.Listnode.Next points_to LN; else L.Einfügen_zwischen_zwei_Listenknoten(I,LN); fi; fi;

Hierbei sei "od" die Markierung des Endes vom "do" und "fi" die Markierung des Endes vom "if".

Neben dem Einfügen von Listenknoten gibt es natürlich auch noch das Löschen von Listenknoten:

Löschen der gesamten Liste



Löschen einer kompletten Liste

Grundidee:

Voraussetzungen: Der Listenzeiger zeigt auf einen Listenknoten. Die Operation besteht aus:

1. Setze den Wert des Listenzeigers auf den Wert Null.

Weitere Operationsschritte sind nicht erforderlich. Dies ist allerdings nur in dieser allgemeinen Betrachtung so. Bei existierenden Programmiersprachen muss unterschieden werden, ob der Auswertungsautomatismus die nicht mehr gebrauchten Knoten selber löscht, oder ob sie einfach als "Speichermüll" übrigbleiben. Bei letzterem Fall müssten alle Listenknoten der Liste L vor dem setzen des Null-Zeigers in den Listenkopf explizit im Programmcode gelöscht werden (zur Vermeidung sogenannter memory leaks).

Pseudocode:

L points_to Null

**Aufgabe****Aufgaben****Aufgabe 6.0**

1. Einfügen/Löschen eines einzigen Listenknotens (bei ansonsten leerer doppelt verketteter Liste).
2. Einfügen/Löschen eines Listenknotens am Anfang einer doppelt verketteten Liste.
3. Einfügen/Löschen eines Listenknotens am Ende einer doppelt verketteten Liste.
4. Einfügen nach einem Listenknoten (zwischen zwei Listenknoten), auf den der Iterator zeigt.
5. Einfügen/Löschen an beliebiger Stelle einer doppelt verketteten Liste.

Zusammenfassung und Ausblick

Abschließend sei noch einmal erwähnt, dass hier aus dem abstrakten Datentyp Listenknoten die dynamische Datenstruktur Liste aufgebaut wird.

Bei modernen Programmiersprachen werden Listen oft über (Standard-)Bibliotheken als sprachergänzendes 'Utility' bereitgestellt. Es gibt dort auch sortierte Listen, Listen mit Zugriff auf das n-te Element, sowie parametrisierbare Listentypen. Auf diese Dinge wird hier jedoch nicht weiter eingegangen.

6.2.2.2 Bäume**Motivation**

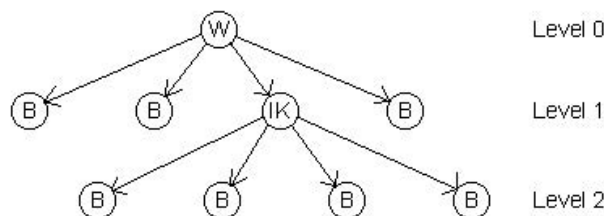
Zunächst werden folgende geografischen Bezeichnungen betrachtet: Bundesrepublik Deutschland, Bremen, Hamburg, Niedersachsen, Schleswig-Holstein, Hannover, Braunschweig, Oldenburg, Jever. Soll jetzt durch Pfeile markiert werden, welcher geografische Ort in welchem geografisch enthalten ist oder welcher geografische Ort geografisch welchen enthält. so erhält man folgende Abbildung):





Geografische Ortsangaben, die einander teilweise enthalten

Bei Abstraktion auf die Struktur ergibt sich somit:



Ein Baum als Abstraktion einer Struktur

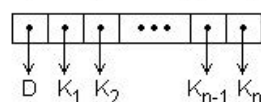
Informal gesprochen wird eine solche Darstellung, wie im Kapitel Graphen, bereits angedeutet, in der Informatik als Baum bezeichnet. Ein Baum besitzt eine Wurzel (W), innere Knoten (IK) und Blätter (B). Diese Knoten eines Baumes sind verbunden durch Kanten (obige Abbildung: gerichtete Kanten). Die Anzahl der Kanten, die man durchlaufen muss, um von einem Knoten zur Wurzel zu gelangen, ist der Level des Knoten. Der maximale im Baum vorkommende Level ist die Höhe (Tiefe) des Baumes.

Ähnlich wie bei den Listen, gibt es auch bei Bäumen einen elementare abstrakten Datentypen, aus deren Objekten die Bäume aufgebaut ist. Diese Datentypen heissen Baumknoten (Auf sie wird im folgenden Unterabschnitt weiter eingegangen). Da Bäume, genauso wie Listen unterschiedlich groß und auch in der GröÙsse veränderbar sind, bilden auch sie eine dynamische Datenstruktur.

Baum und Baumknoten:

Ebenso wie Listen aus Listenknoten aufgebaut sind, basieren Bäume auf dem abstrakten Datentyp des Baumknotens. Der Unterschied zu einem Listenknoten ist in der folgenden Abbildung dargestellt (Wieder wird hier davon ausgegangen, dass eventuelle Nutzdaten über einen geeigneten Zeiger im Knoten angehängt werden können.):

D: Datenzeiger K_i : Nachfolgerzeiger (Kinder)



Ein Baumknoten

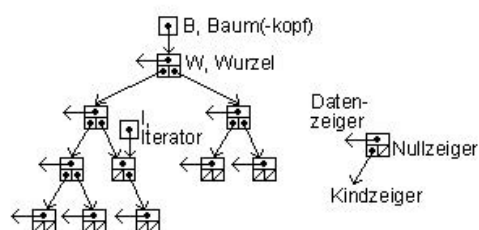
Ein Baumknoten besitzt einen Datenzeiger (D) sowie n Nachfolgerzeiger (Kindzeiger, child pointer). Bei einem Baumknoten ist Folgendes zu beachten: Die Anzahl der Nachfolger ist grundsätzlich variabel; allerdings werden unter anderem auch Baumknoten mit einer festen Anzahl n von Nachfolgern (Kindern) verwendet. Oft werden zwei Nachfolgerzeiger verwendet. Man spricht dann von einem Binärbaum. Selbstverständlich können die Zeiger in einem Baumknoten ebenso wie bei Listenknoten Null-Zeiger sein. Will man diesen Baumknotentyp als abstrakten Datentyp auffassen, werden noch die Operationen auf den Baumknoten gebraucht:

- Setzen des Datenzeigers auf ein Datenobjekt oder den Null-Zeiger.
- Setzen eines der n Nachfolgerzeiger auf einen Nachfolgeknoten oder den Null-Zeiger.
- Das Lesen des Inhaltes eines der Zeiger, oder eine geeignete Fehlermeldung, sofern diese Zeiger auf nicht zulässige Daten zeigen sollten.

Somit ist der abstrakte Datentyp Baumknoten definiert.

Betrachtet wird nun zunächst eine informale Definition der Datenstruktur Baum: 1. Ein Baumknoten, dessen Nachfolgerzeiger leer sind, ist ein Baumes (ein Baum der nur aus seiner Wurzel besteht). 2. Ein Baumknoten, bei dem mindestens ein Nachfolgezeiger auf einen Baum zeigt, ist ein Baum (,der selbst wieder Teil eines anderen Baumes sein kann).

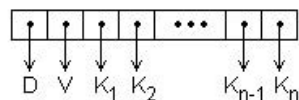
Somit wird die Definition eines Baumes hier auf den Baumknoten zurückgeführt. (Es gibt eine formale Definition, die ohne den Baumknoten auskommt; auf diese wird in diesem Studienmodul jedoch nicht näher eingegangen.) Also wird nun im nächsten Bild dargestellt, wie mit der oben vorgestellten Baumknotenstruktur ein Baum erzeugt wird. Die Darstellung zeigt einen Binärbaum, bei dem jeder Baumknoten genau zwei Nachfolger hat. Bäume ohne Vorgängerzeiger entsprechen in gewisser Weise der einfach verketteten Liste, weil man sich mit einem Iterator ausgehend von der Wurzel immer nur zu den Blättern, jedoch nie in die Gegenrichtung bewegen kann.



Ein einfacher Binärbaum

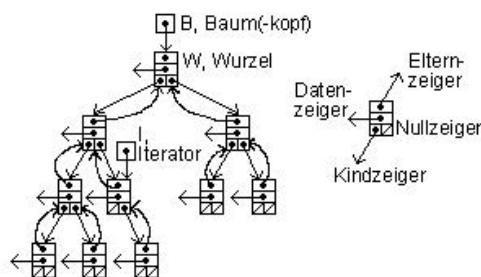
Um diese Einschränkung, dass ein Iterator sich immer nur von dem Baumknoten, auf den er zeigt zu den Blättern hinbewegen kann, zu überwinden, braucht man einen weiteren Zeiger im Baumknoten: den Zeiger auf den Vorgänger- oder Elternknoten, oder in der Praxis oft als "Parent- Pointer" bezeichnet. Ein solcher Baumknoten sieht so aus;

D: Datenzeiger V: Vorgängerzeiger K_i : Nachfolgerzeiger



Ein Baumknoten mit Vorgängerzeiger

Damit lassen sich Bäume mit folgender Struktur aufbauen:



Ein Binärbaum mit Eltern-Zeigern

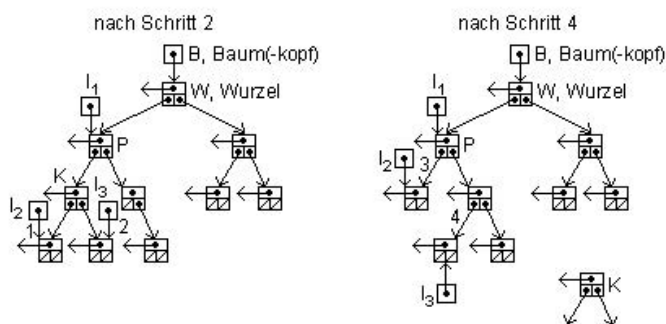
In dieser Art von Bäumen kann sich ein Baum-Iterator von der Wurzel weg und zur Wurzel hin bewegen. (Anmerkung des Autors: Soll ein Baum-Iterator sich zudem auf einer Ebene unter den Kindern eines Baumknoten bewegen können, so müssen diese als Liste (am besten zweifach) verkettet sein.

Analog zu den einfach respektive zweifach verketteten Listen müssen nun sämtliche Einfüge- Lösch- oder Änderungsmethoden für Bäume ohne respektive mit Parent-Zeiger definiert werden, um die jeweilige Baumart als abstrakten Datentyp zu definieren. Dieses Unterfangen würde den Umfang dieses Unterabschnitts erheblich anwachsen lassen, ohne das jedes Mal viel substantiell neues vorgestellt würde, deshalb beschränkt sich dieser Unterabschnitt darauf, einige wenige der möglichen Methoden beispielhaft vorzustellen. Es bleibt den geneigten Lesenden überlassen, sich die fehlenden zu erarbeiten.

Beispielhafte Methoden für Bäume

Um einen einfachen Einstieg in das Thema zu finden beschränken sich beide Beispiele auf Binärbäume. Zunächst wird ein Binärbaum betrachtet, bei dem die Baumknoten keinen Zeiger auf Ihren Eltern-Knoten haben. Hierbei wird das Löschen eines Knotens, der nicht Blatt ist erläutert. Danach wird ein Binärbaum betrachtet, bei dem die Baumknoten sehr wohl einen Zeiger auf Ihren Eltern-Knoten haben. Hier wird das Einfügen eines neuen Baumknotens vor einem Blatt erläutert.

Löschen eines inneren Baumknotens (bei Baumknoten ohne Vorgänger-Zeiger)



Löschen eines inneren Baumknotens in einem einfachen Binärbaum

Grundidee:

Voraussetzungen: Ein Baum enthält mindestens einen inneren Knoten, K, der nicht Wurzel des gesamten Baumes ist. Der Knoten K habe zwei Nachfolgeknoten. Ein Iterator, I_1 , zeigt auf den Baumknoten P, dessen Kind K ist. Die Zeiger im Baum, die nicht auf einen Nachfolgeknoten zeigen, zeigen auf Null.

Die Operation besteht aus:

1. Setze einen Zeiger (Iterator: I_2) auf den 1. Nachfolger des Baumknotens K und
2. einen Zeiger (Iterator I_3) auf den 2. Nachfolger des Baumknotens K.
3. Setze den Nachfolgezeiger des Baumknotens P, der auf K zeigt, auf den ersten Nachfolger des Knotens K.
4. Finde einen Knoten K_2 , der mindestens einen Null-Zeiger als Nachfolgezeiger besitzt. Setze diesen Nachfolgezeiger auf den Knoten, auf den I_3 zeigt.

Weitere Operationsschritte sind nicht erforderlich.

Pseudocode (K sei der Baumknoten, der aus dem Baum gelöscht werden soll.):

Konstruiere Iterator I_2 ;

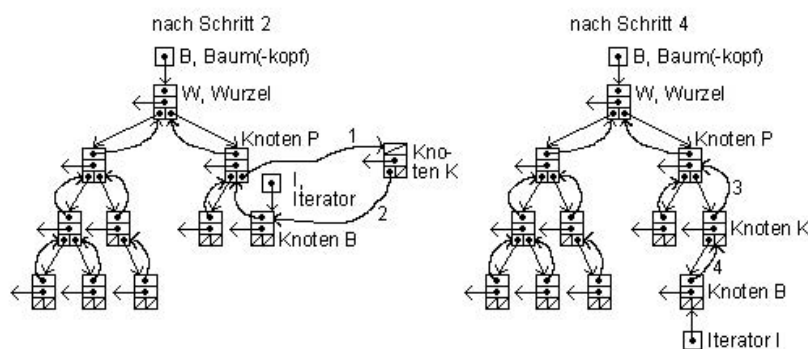
```

I2 points_to K.First;
Konstruiere Iterator I3;
I3 points_to K.Second;
If (P.First = K) then P.First points_to I2.Treenode;
If (P.Second = K) then P.Second points_to I2.Treenode;

Find K2 with (K2.First = Null oder K2.Second = Null);
If (K2.First = Null) then K2.First points_to I3.Treenode;
If (K2.Second = Null) then K2.Second points_to I3.Treenode;

```

Einfügen vor einem Blatt (bei Baumknoten mit Vorgänger-Zeiger)



Einfügen eines inneren Binärbaumknotens vor einem Blatt

Grundidee:

Voraussetzungen: Ein Baum enthält mindestens einen inneren Knoten, P. Der Knoten P habe mindestens einen Nachfolger B. Der Nachfolger B ist ein Blatt. Ein Iterator, I zeigt auf den Baumknoten B, dessen Parent P ist. Die Zeiger im Baum, die nicht auf einen Nachfolgeknoten zeigen, zeigen auf Null. Es existiert ein weiterer Baumknoten K, der zwischen den Knoten P und B eingefügt werden soll.

Die Operation besteht aus:

1. Setze den Nachfolgezeiger des Vorgängers des Knotens B (des Knotens P), der auf B zeigt, auf den Knoten K.
2. Setze den ersten Nachfolgezeiger des Baumknotens K, auf den Knoten B.
3. Setze den Vorgängerzeiger des Knotens K auf den Vorgänger des Baumknotens B (den Baumknoten P).
4. Setze den Vorgängerzeiger des Knotens B auf den neuen Baumknoten K. Weitere Operationsschritte sind nicht erforderlich.

Pseudocode (K sei der Baumknoten, der zwischen P und B eingefügt werden soll. Ein Iterator I zeigt auf den Baumknoten B.:

If (I.Treenode.Previous.First = B) then I.Treenode.Previous.First points_to K;

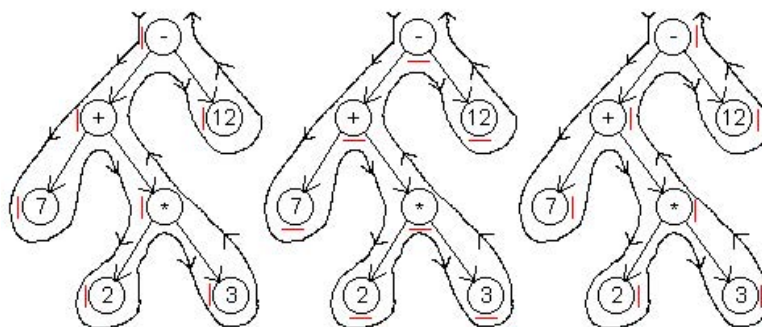
If (I.Treenode.Previous.Second = B) then I.Treenode.Previous.Second points_to K;

If (K.First points_to I.Treenode;

K.Previous points_to I.Treenode.Previous;

I.Treenode.Previous points_to K;

Zu beachten ist in beiden Fällen, dass das Finden eines Knoten die Traverse durch einen (ein Durchlaufen eines) Baum(es) erfordert. Eine solche Traverse kann auf verschiedene Arten erfolgen. In der folgenden Abbildung sind drei mögliche Arten der Traverse von links nach rechts dargestellt, wobei die Traverse die gepfeilte Route entlang des Baumes ist:



Traverse durch einen Baum mit Eltern-Zeigern

Eine Anwendung der Traverse gibt es in der Informatik z. B. bei der Analyse von Bäumen, die Rechenausdrücke darstellen. Je nach Art der Traverse erhält man unterschiedliche Notation für denselben im Baum gespeicherten Ausdruck:

Preorder(-Traverse)

Ein Knoten wird gefunden, wenn er bei der Traverse links passiert wird. Für mathematische Terme ergibt sich so die Prefix-Notation: - + 7 . 2 3 12

Inorder(-Traverse)

Ein Knoten wird gefunden, wenn er bei der Traverse unten passiert wird. Für mathematische Terme ergibt sich so die Infix-Notation: $7 + 2 \cdot 3 - 12$ (Die Bedeutung des Terms der obigen Bäume ist übrigens: $(7 + (2 \cdot 3)) - 12$, da "Punkt- vor Strichrechnung" gilt und "+" bzw. "-" linksassoziativ sind.)

Postorder(-Traverse)

Ein Knoten wird gefunden, wenn er bei der Traverse rechts passiert wird. Für mathematische Terme ergibt sich so die Postfix-Notation: $7 \ 2 \ 3 \ . \ + \ 12 \ -$ (Die Postfix-Notation wird gelegentlich auch umgekehrte polnische Notation (UPN) genannt.)

Diese Traversen können sehr leicht rekursiv implementiert werden. Da Rekursion bisher noch nicht Gegenstand dieses Studienmoduls ist, ist die Ausführbarkeit der Find-Operationen hier als gegeben zu betrachten.

Zusammenfassung

Ein Baum ist eine dynamische Datenstruktur, die durch Zusammenfügen von Objekten des abstrakten Datentyps "Baumknoten" entsteht. Dabei können die Baumknoten einen Vorgänger- Zeiger haben, oder auch nicht. Die Anzahl der Nachfolgerzeiger eines Baumknotens ist entweder variabel oder fest. Wenn Baumknoten jeweils zwei Nachfolgerzeiger haben, so fügt man diese Baumknoten zu Binärbäumen zusammen. Diese spielen in der Informatik eine wichtige Rolle.

Abschlussbemerkung: Es gibt verschiedene Arten von Bäumen, auf die hier nicht näher eingegangen wird.

6.3 Sortieren

Motivation

Wir alle kennen aus dem alltäglichen Leben, dass etwas sortiert ist. Dies geht über die CDs beim großen Musikhändler, über Telefonbücher bis zu Lagerbeständen, die z. B. nach Artikelnummern oder nach Größe sortiert sind. Der Sinn von Sortierungen (das Sortieren an sich kostet ja einen gewissen Aufwand) besteht darin, nach einem Sortiervorgang Dinge oder Daten leichter einordnen oder wiederfinden zu können. Der Rest dieses Unterkapitels nennt ein paar in der Informatik bei der Datensortierung verwendete Sortierverfahren und stellt beispielhaft eines davon eingehend vor.

Sortierverfahren

Sortierv Verfahren in der Angewandten Informatik sortieren in der automatisierten Datenverarbeitung Daten. Diese können zum Beispiel in Arrays gespeichert sein, in Datenstrukturen wie Listen oder Bäumen oder in anderen komplexeren Datenmodellen. Einige einfache Sortierv Verfahren sind:

Bubblesort

Ein Speicherbereich wird von der ersten bis zur vorletzten Position durchlaufen, bei entsprechendem Vergleichsergebnis wird das Datum eines Speicherplatzes mit dem des nächsthöheren Speicherplatzes vertauscht. Dies wird wiederholt von der ersten bis zu vorvorletzten Position, dann von der ersten bis zur viertletzten usw., bis alle richtig sortiert sind. Bubblesort ist ein einfaches, aber langsames Sortierv Verfahren.

Insertionsort

Insertionsort ist auch ein langsames Verfahren, aber deutlich besser als Bubblesort. Im Unterabschnitt Insertertionsort in diesem Unterkapitel (weiter hinten) wird Insertionsort ausführlich erläutert.

Quicksort

Ein Verfahren, bei dem der zu sortierende Bereich in Abhängigkeit von dem Vergleichsergebnis mit einem so genannten Pivotelement in zwei Bereiche aufgeteilt wird. In einem Bereich sind die Daten, die kleiner sind als das Pivotelement, im anderen die, die größer sind. Anschließend werden die beiden Bereiche nach demselben Prinzip webersortiert. Dies Verfahren ist meist recht schnell, kann aber im schlechtesten Fall ähnlich schlecht sein wie die beiden bereits genannten Verfahren.

Heapsort

Heapsort sortiert in einem Array. In einer ersten Phase wird in dem Array eine baumartige Vorsortierung vorgenommen (Heap). Dabei sind die Elternknoten größer (bei umgekehrter Sortierung kleiner) als ihre Kinder (Heap-Eigenschaft). In der zweiten Phase wird stets das erste, das größte (oder auch das kleinste) Element entnommen und hinten (ggf. vor dem vorigen größten/kleinsten) eingefügt. Dann wird der Heap repariert, so dass er wieder die Heap-Eigenschaft erfüllt. Dann kann wieder das nächste größte / kleinste Element hinten einsortiert werden usw. Als Ergebnis erhält man aufsteigend / absteigend sortierte Daten. Dieses Verfahren ist auch im schlechtesten Fall recht schnell.

Es gibt noch weitaus mehr Sortierv Verfahren, auf die in diesem Modul jedoch nicht näher eingegangen wird.

Insertionsort

Sortieren durch Einfügen (Insertsort, Insertionsort oder Insort) ist den meisten von uns durch das Kartenspielen bekannt: Wenn man ein unsortiertes Blatt auf der Hand hält und dieses aber sortiert halten möchte, so zieht man Karten, die bereits gut sortiert sind heraus und steckt sie an passender Stelle in der Hand wieder ein. Allerdings kann der Mensch hierbei von seinem Verständnis der Karten profitieren und die Züge, je nach vorhandener "Hand" (Karten, die er besitzt) so planen, dass er mit recht wenigen Entnehmen-und-Einfügen-Schritten auskommt. Eine automatisierte Datenverarbeitung dagegen braucht ein Verfahren, dass, zwar nach dem gleichen Prinzip vorgehend, für beliebige (unsortierte) Hände sicher (schematisch) funktioniert, also einen geeigneten Algorithmus. Zur Erläuterung sei hier zunächst die Grundidee des Sortiervfahrens durch Einfügen wiedergegeben; danach folgt eine Darstellung in Pseudocode, und abschließend wird das

dynamische Verhalten des Verfahrens bildhaft dargestellt:

Verfahren

Eine sortierfähige Datenstruktur sei vorhanden. Diese kann ein Feld (Array) sein, oder auch eine doppelt verkettete Liste. Hier wird beispielhaft ein Feld verwendet, dessen Felder (von 1 bis n) nummeriert sind; z. B. (ohne Beschränkung der Allgemeinheit) von links nach rechts:

Ausgehend von der zweiten Position (von links) und dann bis zum Ende des Feldes wird an der aktuellen Position jeweils ein Wert, der nicht passend einsortiert ist, entnommen. Daraufhin werden die Inhalte aller Felder links davon, die rechts von dem entnommenen Wert sein müssen, um ein Feld nach rechts aufgerückt. Dies geschieht in einer inneren Schleife. In das nun frei werdende Feld wird der vorher entnommene Wert, nun garantiert der Sortierung entsprechend, eingefügt.

Da beim zweiten Feld begonnen wird, müssen am Anfang schlimmstenfalls die Inhalte der beiden ersten Positionen vertauscht werden. Bei allen weiteren aktuellen Positionen kann man davon ausgehen, dass die Positionen links davon bereits sortiert sind. Deshalb kann durch Aufrücken garantiert eine geeignete Stelle gefunden werden, an der der entnommene Wert der Sortierung entsprechend wieder eingefügt werden kann.

Pseudocode

Problem: Es ist eine Zahlenfolge (z.B. ganze Zahlen) aufsteigend zu sortieren.

Lösung: Die Werte in den Plätzen $P[1]$, $P[2]$, ..., $P[m]$ werden durch Einfügen sortiert. Am Ende stehen die Werte in den Plätzen in aufsteigender Reihenfolge. Eine Notation in einer leicht anderen Variante von Pseudocode ist wie folgt:

1. für(alle Felder n , vom zweiten bis zum letzten, m , erhöhe n nach jedem Schleifendurchlauf um 1 und tue:) {
2. die Position j sei $n - 1$;
3. aktuellerWert sei der Wert des n . Feldes;
4. Solange ((j eins oder größer ist) und (der Wert an der Stelle j des Feldes P größer als der aktuelle Wert ist)) tue: {
5. Übertrage den Wert an der Stelle j des Feldes P in die Stelle $j+1$ des Feldes P ; (aufrücken)
6. verringere den Wert von j um eins;
7. }; (Ende der Solange-Schleife)
8. $P[j+1] = \text{aktuellerWert}$;
9. }; (Ende der Für-Schleife)

Dynamisches Verhalten

Um auf das dynamische Verhalten einzugehen, folgt hier eine Notation des Algorithmus Insertionsort in einer an C oder Java angelehnten Notation. Dabei werden die Werte in den Plätzen $P[1]$, $P[2]$, ..., $P[m]$ durch Einfügen sortiert. Am Ende stehen die Werte in den Plätzen in aufsteigender Reihenfolge:



Code

```
1. for( $n = 2; n \leq m; n = n + 1$ ) { 2.  $j = n - 1$ ;  
3. aktueller Wert =  $P[n]$ ;  
4. while( $j \geq 1 \ \&\& \ P[j] > \text{aktuellerWert}$ ) { 5.  $P[j+1] = P[j]$ ;  
6.  $j = j - 1$ ;  
7. };  
8.  $P[j+1] = \text{aktuellerWert}$ ;  
9. };
```

		47 10 50 48 35 3
10 47 10 50 48 35 3	10 47 47 50 48 35 3	10 47 50 48 35 3
50 10 47 50 48 35 3	50 10 47 50 48 35 3	10 47 50 48 35 3
48 10 47 50 48 35 3	48 10 47 50 50 35 3	10 47 48 50 35 3
35 10 47 48 50 35 3	35 10 47 47 48 50 3	10 35 47 48 50 3
3 10 35 47 48 50 3	3 10 10 35 47 48 50	3 10 35 47 48 50



Dynamisches Verhalten des Algorithmus 'Sortieren durch Einfügen'

In der vorigen Tabelle ist das dynamische Verhalten des Verfahren Insertionsort wie folgt dargestellt:

- In der dritten (rechten) Spalte sieht man das Feld P in der Ausgangssituation bzw. nach einem äußeren Schleifendurchlauf.
- In der ersten (linken) Spalte ist das Feld P nach Ausführen des Setzens des aktuellen Wertes (Zeile 3 des obigen Programmcodes) dargestellt. Die Position des separat dargestellten aktuellen Wertes zeigt gleichzeitig die Position der Indexvariablen n dieser äußeren Schleife an. Zu beachten ist, dass im Feld P keine Lücke entsteht. Da der aktuelle Wert in einer extra Variable gehalten wird, kann (und soll) die entsprechende Position des Feldes jedoch ohne Datenverlust überschrieben werden.
- In der mittleren Spalte ist die Situation nach dem Aufrücken (der zu großen Zahlen), also nach Ausführung der inneren Schleife, dargestellt. Zu beachten ist, dass keine Lücke entsteht. Allerdings steht der letzte aufgerückte Wert zweimal im Feld P. Dies gilt nicht in der Zeile, in der die 50 der aktuelle Wert ist. Da hier die 47 nicht aufrückt, weil sie kleiner als 50 ist, sieht man im Feld P keinen Unterschied.

Das dynamische Verhalten des Verfahrens kann in folgender Animation eingehender studiert werden:



In der Online-Version befindet sich an dieser Stelle eine Simulation.

Beispiel Sortieren durch Einfügen (420KB)

Diskussion des Verfahrens "Sortieren durch Einfügen"

Bei der Diskussion eines Sortierverfahrens werden verschiedene Kriterien betrachtet. Beispielsweise lässt es sich iterativ (mit Hilfe von Schleifen) implementieren und dann untersuchen, wie sein Speicherplatz-/Zeitbedarf im schlechtesten Fall oder im Durchschnitt ist.

Implementierung:

iterativ

Geeignet für:

Der Algorithmus Insertionsort wurde hier für Felder vorgestellt, er kann aber auch für Listen angepasst werden.

Zusätzlicher Speicherplatzbedarf (best, average und worst case):

Insertionsort braucht außer den Daten selbst nur genau eine zusätzliche Variable, um den aktuellen Wert zu speichern.

Zeitbedarf (best case):

Im besten Fall läuft der Algorithmus einmal durch das Feld, ohne dass aufgerückt werden muss, also $O(n)$.

Zeitbedarf (average case):

Im Mittel muss das Feld halb durchlaufen und halb aufgerückt werden, also: $O(n^2)$

Zeitbedarf (worst case):

Im schlimmsten Fall ist das Feld falschherum sortiert und daher muss immer komplett aufgerückt werden, also $O(n^2)$

Vergleiche (best case):

$n-1$

Vergleiche (average case):

$$n^2/4$$

Vergleiche (worst case):

$$n^2/2$$

Kopieraktionen (best case):

$$0$$

Kopieraktionen (average case):

$$n^2/4$$

Kopieraktionen (worst case):

$$n^2/2$$

6.4 Aufgaben zu Algorithmen und Datenstrukturen

In diesem Kapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt.



Aufgabe

Aufgabe 6.1

Geben Sie für die Addition von Dezimalzahlen einen intuitiven Algorithmus an. Erklären Sie insbesondere, wie ein Übertrag abgewickelt wird.



Aufgabe

Aufgabe 6.2

Definieren Sie einen einfachen *abstrakten Datentyp* Ihrer Wahl. Bilden Sie vorzugsweise Vierergruppen mit Ihren Mitstudierenden und diskutieren Sie Ihren Datentyp mit Ihren Mitstudierenden in einem geeigneten Fachforum im Kurs im Lernraumsystem.



Aufgabe

Aufgabe 6.3

1. Definieren Sie in Java eine Struktur für ein Spielzeugauto. Das Auto soll die Elemente Farbkennziffer (Ganzzahl), Gewicht (Fließkommazahl) und Modellname (Zeichenkette) haben.
2. Speichern Sie viele Modellautos (beispielsweise 263) so ab, dass immer mit dem gleichen Zeitaufwand schnell auf ein beliebiges n-tes Element zugegriffen werden kann (also nicht Baum und auch nicht Liste).



Aufgabe

Aufgabe 6.4

Finden sie für *einfach verkettete Listen* Lösungen für folgende Fragestellungen:

1. Schreiben Sie in Anlehnung an die in der Vorlesung verwendeten Darstellungsweise (Grundidee und Pseudocode) einen Algorithmus für das Ändern der Daten des ersten Listenknotens auf.
2. Schreiben Sie in Anlehnung an die in der Vorlesung verwendeten Darstellungsweise (Grundidee und Pseudocode) einen Algorithmus für das Ändern der Daten des letzten Listenknotens auf.
3. Der Datentyp des Listenknotens einer einfach verketteten Liste sei folgendermaßen modifiziert. Ausser den Attributen data und next für die (Nutz-)Daten bzw. den Nachfolgeknoten gebe es noch das Attribut name, eine Zeichenkette, die für jeden Knoten ein Unikat ist und daher zur Identifikation eines Knoten verwendet werden kann. Sie können annehmen, dass es einen Operator **equals** gibt, der zwei Zeichenketten vergleicht und true liefert, wenn Sie die gleiche Buchstabenfolge enthalten, false sonst (Beispiel: "Anna" **equals** "Otto" liefert false).

In einer Liste habe einer der Knoten den Namen "Hugo". Dieser Knoten habe mindestens einen Vorgängerknoten und mindestens einen Nachfolgeknoten. Schreiben Sie in Anlehnung an die in der Vorlesung verwendeten Darstellungsweise einen Algorithmus für das Ändern der Daten des Listenknotens mit dem Namen "Hugo" auf.



Aufgabe

Aufgabe 6.5

1. Stellen Sie analog zu den Abbildungen im Abschnitt Listen eine zweifach verkettete Liste mit insgesamt 4 Knoten zeichnerisch dar.
2. Stellen Sie grafisch dar, wie nach dem 3. Knoten ein weiterer Knoten eingefügt wird.



Aufgabe

Aufgabe 6.6

Es geht um das Sortierv Verfahren Bubblesort, welches Sie in der Literatur oder im Internet finden können.

1. Geben Sie wie im Unterkapitel Sortieren eine genaue Beschreibung des Verfahrens Bubblesort für ein Feld (Array), sowie eine Darstellung in Pseudocode an.
2. Überlegen Sie sich Beispiele für den besten und den schlechtesten Fall einer Initialbelegung eines Vierer-Feldes von positiven Zahlen und geben Sie dafür den Aufwand an notwendigen Vergleichen und Vertauschungen an.
3. (schwierig und daher optional): Schätzen Sie ab, wie im Mittel (average case) die Anzahl der Vergleiche und Vertauschungen aussieht. Geben Sie eine kurze Begründung dafür. Eine mathematische Herleitung ist nicht erforderlich.

7 Aufbau eines Rechnersystems



Gliederung

- 7 Aufbau eines Rechnersystems
- 7.1 Begriffserklärung
- 7.2 Das Schichtenmodell eines Rechnersystems
- 7.3 Die Struktur der von Neumann-Maschine
- 7.4 Prozessorarchitekturen
- 7.5 Maschinenbefehle und Mikroprogrammierung
- 7.6 Ein-/Ausgabeorganisation
- 7.7 Multimedia-Peripherie
- 7.8 Bussysteme
- 7.9 Speichertechnologien
- 7.10 Leistungsgrößen und Leistungsbewertung
- 7.11 Konzepte der Parallelverarbeitung
- 7.12 Aufgaben zum Thema Rechnerhardware
- 7.13 Fragen mit Musterlösungen: Rechnerhardware

7.1 Begriffserklärung

Die Rechnerarchitektur besteht aus den Gebieten Struktur, Organisation, Implementierung und Leistung. Unter der Rechnerstruktur versteht man das physikalische Zusammenwirken der einzelnen Hardwarekomponenten eines Rechners. Dabei müssen die zeitabhängigen Wechselwirkungen und die Steuerung der Komponenten organisiert werden. Hierin besteht eine der Hauptaufgaben des Betriebssystems. Die Implementierung legt die konkrete Ausgestaltung von Bausteinen bzw. Baugruppen fest und die Leistung beschreibt das nach außen sichtbare Systemverhalten.

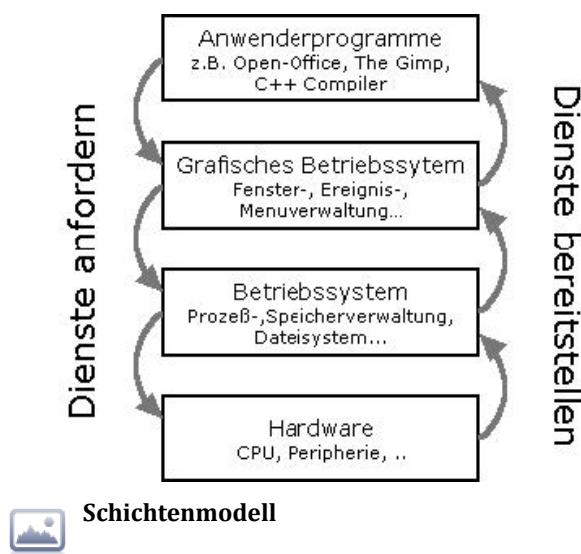
7.2 Das Schichtenmodell eines Rechnersystems

Betrachtet man einen einfachen Computer mit einer CPU (Central Processing Unit) auf einem Motherboard mit Speicher und angeschlossenen Peripheriegeräten wie Drucker und Festplatte, aber ohne jegliche Software, so beherrscht die CPU nicht viel mehr als

- Speicherinhalte in Register bzw. Registerinhalte im Speicher abzulegen

- Registerinhalte arithmetisch oder logisch zu verknüpfen
- mit IN- oder OUT-Befehlen auf Register in Controllern von Peripheriegeräten zuzugreifen

Immerhin kann man hiermit schon ein Zeichen von der Tastatur einlesen und auf dem Bildschirm ausgeben oder einen bestimmten Sektor auf der Festplatte lesen oder beschreiben usw.. Wollte man auf dieser niedrigen Ebene mit dem Computer arbeiten, so müsste man sich mit den technischen Details aller Peripheriegeräte und der CPU auskennen. Außerdem ließe sich portable Software, d.h. Software, die auch auf Computern mit anderer Hardware läuft, nur schwer erstellen, da sich die Hersteller in der Hardware und deren Ansteuerung oft unterscheiden. Es klafft also eine große Lücke zwischen einem vom Anwender intuitiv zu bedienenden Rechner und den Fähigkeiten der Hardware. Um diese Lücke auszufüllen, müssen zwischen Hardware- und Anwenderebene Schichten eingefügt werden. Die nächste Abbildung verdeutlicht den Sachverhalt:



Jede Schicht fordert von der darunterliegenden Schicht Dienste an. Eine Anforderung des Anwenders pflanzt sich von oben nach unten fort, bis sie auf der Hardware-Ebene angekommen ist, dort die entsprechende Aktion auslöst und das Resultat an die höheren Ebenen zurückmeldet. Andersherum gesehen, bietet jede Schicht der nächst höheren Dienste an, so dass ein Programmierer lediglich die **Schnittstelle** zur nächst tieferen Ebene kennen muss.

Bei den Schnittstellen wird zwischen Hard- und Softwareschnittstellen unterschieden. **Hardware- Schnittstellen** sind Konventionen, welche die Verbindung verschiedener Bauteile festlegen. Eine Schnittstellendefinition muss folgende Punkte beinhalten:

- Den mechanischen Aufbau der Steckverbindungen
- Die Zuordnung der elektrischen Kontakte zu den elektrischen Signalen (z.B. Steckerbelegung)
- Die statischen und dynamischen Werte der elektrischen Signale (z.B. Höhe und Polarität der

Signalspannung, die Frequenz, die Kurvenform)

- Die elektrischen Eigenschaften der Treiber und Empfänger (wird z.B. die absolute Höhe der Spannung ausgewertet oder die Potentialdifferenz zwischen zwei Leitungen → "**Differenztechnik**")
- Das Übertragungsprotokoll für die Daten (Sender und Empfänger müssen ständig über den

Zustand der Übertragung informiert sein)

- Die elektrischen Eigenschaften des Kabels (Dämpfung, **Wellenwiderstand** i.e.S. das Verhältnis von Spannungs- zu Stromamplitude an jedem Ort der Leitung) Die maximale Kabellänge Die Anzahl der anschließbaren Geräte

Software-Schnittstellen oder **Treiber** sind Übersetzungsprogramme zur Ansteuerung von Hard- oder Software-Komponenten. Der Treiber ermöglicht die Benutzung einer Komponente, ohne deren inneren Aufbau zu kennen. Treiber sind auf der Betriebssystemebene angesiedelt und müssen dem Betriebssystem per Anmeldung bekannt gemacht werden. Erfolgt dann z.B. eine Ausgabe auf die entsprechende Komponente, so "weiß" das Betriebssystem, an welchen Treiber die Daten zu übergeben sind. Der Treiber steuert dann die Hardware entsprechend an. Treiber werden aber auch verwendet, um virtuelle Geräte, wie z.B. eine RAM-Disk, zu bedienen. Virtuell heißt hier, dass das Gerät physikalisch nicht vorhanden ist, für den Benutzer der Zugriff aber transparent geschieht, d.h. nach außen hin verhält sich die RAM-Disk wie eine tatsächlich vorhandene Festplatte.

7.3 Die Struktur der von Neumann-Maschine

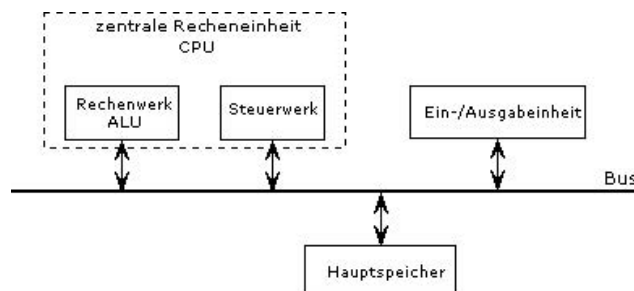
Das "von Neumann'sche" Rechnermodell wurde 1946 von John von Neumann sowie seinen Mitarbeitern Arthur W. Burks und Herman H. Goldstine als theoretisches Konzept im Rahmen der Entwicklung des US-amerikanischen Kernwaffenprogramms vorgestellt. Es war richtungsweisend für die Rechnerentwicklung der ersten 40 Jahre und wird heute in abgewandelter Form immer noch realisiert. Das "von Neumann'sche" Rechnermodell stellt eine universelle Rechnerarchitektur zur

Lösung aller Arten berechenbarer, insbesondere arithmetischer, Probleme dar, d.h. Rechnerstruktur und Problemstruktur sind voneinander unabhängig. Das Konzept wurde später als Princeton-Rechner verwirklicht. Die wesentlichen Eigenschaften des von Neumann'schen Modells bestehen aus folgenden Punkten:

Hauptkomponenten des Rechners sind Rechenwerk (ALU: Arithmetic Logical Unit), Steuerwerk, Hauptspeicher, Eingabe-/Ausgabeeinheit und die Datenverbindungen (Busse). Steuerwerk und Rechenwerk bilden zusammen die zentrale Recheneinheit (CPU, Prozessor). Die Steuerung erfolgt über binär codierte Anweisungen (Befehle). Die Daten sind unabhängig von den Speicherplätzen. Anweisungen und Daten befinden sich gemeinsam im Speicher. Die Befehle werden streng sequentiell abgearbeitet und die Abarbeitung kann durch bedingte oder unbedingte Sprungbefehle unterbrochen bzw. durch eine STOP-Anweisung gestoppt werden.

Die untenstehende Abbildung zeigt den prinzipiellen Aufbau des von Neumann'schen Rechnermodells. Der Befehlszyklus besteht aus zwei sich zyklisch wiederholenden Phasen:

1. Befehl aus dem Hauptspeicher holen und im Steuerwerk interpretieren (decodieren)
2. Operand aus dem Hauptspeicher holen, Befehl ausführen und das Resultat wieder in den Hauptspeicher zurückschreiben



von Neumann'sches Rechnermodell

Die Aufgaben der einzelnen Komponenten ist folgendermaßen verteilt:

- Das Steuerwerk steuert den Ablauf der Anweisungen.
- Das Rechenwerk führt arithmetische Operationen (zumindest die Grundrechenarten für Fest- und Gleitkommaarithmetik) sowie logische und arithmetische Bitoperationen durch.
- Im Speicherwerk werden Daten und Anweisungen gespeichert.

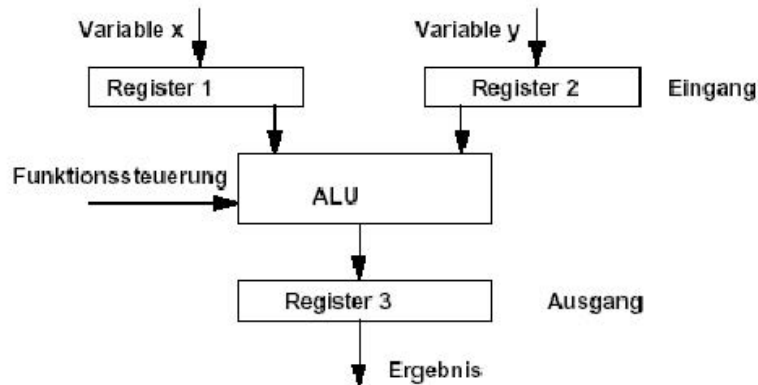
- Die Ein-/Ausgabeeinheit übernimmt die Ein- und Ausgabe von Daten und Anweisungen Rechner, die streng nach diesem Konzept aufgebaut würden, hätten einige gravierende Nachteile:
 1. Befehle und Daten stehen gleichzeitig nebeneinander im Hauptspeicher und müssen über dieselbe Speicherschnittstelle gelesen und geschrieben werden, behindern sich somit gegenseitig.
 - Ausweg: Getrennte Bussysteme und Speicher für Daten und Befehle (**Harvard Architektur**, Princeton-Rechner)
 2. Sequentielle Abarbeitung der einzelnen Programmschritte in den zwei Phasen. Es besteht keine Möglichkeit zur gleichzeitigen Ausführung mehrerer Schritte.
 - Ausweg: Nutzung von Parallelität z.B. Multiprozessorsysteme, Multithreading (parallele Ausführung mehrerer Programmfäden)
 3. Befehlsabarbeitung und Steuerung laufen nur in einer Ebene ab. Dadurch entstehen Probleme bei der Unterprogrammtechnik, beim Multitasking und beim Multiuserbetrieb.

Beispiel von-Neumann-Rechner zum download :[File:unirechn.zip](#)

7.4 Prozessorarchitekturen

Der **Prozessor** besteht im Wesentlichen aus Steuerwerk und Rechenwerk. Auf Spezialprozessoren (wie z.B. Signalprozessoren) soll hier nicht eingegangen werden.

Das **Rechenwerk** (**ALU Arithmetic Logical Unit**) führt in einem Prozessor die anstehenden Rechenoperationen auf binärer Ebene aus, z.B. Additionen und Multiplikationen von Binärzahlen. Die Operanden dieser Operationen sind dazu aus dem Hauptspeicher in spezielle Speicherstellen des Prozessors zu übertragen, die Register. Wiederum spezielle Register, die **Akkumulatoren**, können die Ergebnisse von Operationen aufnehmen. Von dort werden Ergebnisse weiterverarbeitet oder zurück in den Hauptspeicher übertragen.



ALU mit Registern

Im Takt n werden die Eingänge der Register 1 und 2 mit Variablen, z. B. aus dem Speicher oder auch aus dem Ergebnisregister 3, geladen. Mit der fallenden Taktflanke von n liegen diese Daten an den Ausgängen der Register 1 und 2 an und werden durch die ALU verarbeitet. Mit der steigenden Flanke von Takt $(n+1)$ werden sie ins Register 3 eingelesen und stehen nach der fallenden Flanke von $(n+1)$ am Ausgang zur Verfügung. (Unter einer Taktflanke versteht man den Bereich, in dem der Takt seinen Zustand wechselt. Steigende Taktflanke: Wechsel von 0 V Spannung auf maximale Taktspannung; fallende Taktflanke: Wechsel von maximaler Taktspannung auf 0 V.

Haben die Register in obiger Abb. eine Breite von 32 Bit, spricht man von einem 32-Bit-Prozessor. Allerdings wird dann stillschweigend davon ausgegangen, dass die Bussysteme ebenfalls mit 32 Bit arbeiten. Es gibt jedoch Mischformen, wie z.B. den 68000-Prozessor von Motorola. Dieser verfügt zwar über 32-Bit Register, jedoch arbeiten die Bussysteme nur mit 16 Bit.

Das **Steuerwerk** steuert die Abarbeitung des Maschinenprogramms, das in Maschinensprache vorliegt. Als Maschinensprache versteht man hierbei den hardwareabhängigen Befehlssatz des Prozessors (nicht mit "Assembler" zu verwechseln! Siehe Kapitel Addition und Subtraktion). Es interpretiert (decodiert) die einzelnen Befehle und steuert ihre Ausführung. Ein Beispiel hierfür wäre die Beauftragung des Rechenwerks, die Binärzahlen der Register A und B miteinander zu addieren. Dazu bedient es sich neben der implementierten Logik zweier Register: dem Befehlszähler und dem Befehlsregister. Der **Befehlszähler** enthält die Adresse des nächsten auszuführenden Befehls des im Hauptspeicher befindlichen Programms. Das **Befehlsregister** enthält den bereits decodierten auszuführenden Befehl. Der eigentliche Befehlscode (Opcode) steht binär codiert im Speicher (z.B. könnte ein Opcode 0 dem Maschinenbefehl NOP, i.e. No Operation, entsprechen). Ein **Befehlszyklus** des Steuerwerks besteht somit aus drei Schritten:

1. Hole den Befehl, dessen Adresse im Befehlszähler steht, in das Befehlsregister.
2. Schreibe die Adresse des nächsten Befehls in den Befehlszähler.
3. Führe den im Befehlsregister stehenden Befehl aus.

Neben den oben angesprochenen arithmetischen Befehlen gibt es Transportbefehle sowie bedingte

und unbedingte Sprungbefehle.

Im Regelfall wird also die Abarbeitung eines Programms gestartet und es wird der Befehlszyklus so lange durchlaufen, bis ein STOP-Befehl kommt. Es gibt darüber hinaus aber einen Mechanismus, mit dem in diesen Ablauf eingegriffen werden kann, die **Programmunterbrechung auf Grund von Ereignissen (Interrupt, Trap)**. Ereignisse sind zum Beispiel das Drücken einer Taste oder der Klick auf eine Maustaste.

Der Prozessor ist für die Ereignisbearbeitung zuständig:

- Registrieren eines Ereignisses
- Unterbrechen eines Programms
- Reaktion auf das Ereignis (durch Ausführung eines dafür vorgesehenen Programms)
- Fortführung des unterbrochenen Programms.

An grundlegenden **Prozessorarchitekturen** sind im Wesentlichen zwei Architekturen zu betrachten:

- Die CISC-Architektur (Complex Instruction Set Computer)
- Die RISC-Architektur (Reduced Instruction Set Computer)

Allerdings ist für heutige Prozessoren diese Einteilung überholt, da beide Architekturen sich mehr und mehr annähern. Die Ausgangssituation soll aber dennoch aufgezeigt werden.

Die CISC-Architektur

Charakteristika:

- Relativ schnelle Arbeitsweise der CPU
- Sehr viel und teilweise sehr komplexe Maschinenbefehle
- Sehr langsamer Arbeitsspeicher
- Sehr kleiner und teurer Arbeitsspeicher
- Relativ wenige Register im Prozessor

Probleme:

- Ein großer Teil der verfügbaren Befehle wurde (zumindest beim vom Compiler erzeugten Code) nicht verwendet
- Die verwendeten komplexen Befehle trugen nur 20% zur Laufzeit des generierten Codes bei
- Die übrigen hauptsächlich verwendeten Befehle sind sehr einfach

Computerfamilien:

- zSeries Architektur von IBM
- VAX von DEC (auslaufend)
- x86 bis 80386 von Intel (auslaufend)
- 680x0 - Serie von Motorola (auslaufend)

Die RISC-Architektur

Die RISC-Prozessorarchitekturen kamen Anfang der 80er Jahre auf, als es aufgrund der VLSI- Entwurfstechnologie möglich wurde, Maschinenbefehle durch einen einzigen Mikrobefehl direkt auf einem Chip zu realisieren, also durch fest verdrahtete Schaltwerke, anstatt durch ein Mikroprogramm, wie bei den CISC-Prozessoren. Allerdings musste man sich dabei auf die wichtigen Maschinenbefehle beschränken.

Charakteristika:

- Wenige, einfache Befehle, die sehr schnell ausgeführt werden
- Wenige Befehlsformate
- Feste Länge von Operationscode und Operanden
- Sehr viele Mehrzweckregister im Prozessor
- Operationen können ausschließlich in Registern ausgeführt werden Pipelining
- zwischen Register und Hauptspeicher ist ein schneller CACHE-Speicher platziert.

Probleme:

- Durch das Pipelining ist bei Bearbeitungsbeginn des nächsten Befehls dieser noch gar nicht definitiv bekannt (ggf. muss dieser wieder verworfen werden, bei Sprungbefehlen z. B. ist das nicht der nächste in der Programmsequenz)

Computerfamilien:

- pSeries Architektur von IBM
- Power PC von IBM/Motorola
- Sparc von SUN
- Indigo von SGI

Beim Pipelining werden die aufeinanderfolgenden Phasen des Befehlszyklus versetzt parallel bearbeitet:

Takt:	t+1	t+2	t+3	t+4	t+5	t+6	t+7	
	BH	DE	OP	AU	RS			1. Befehl
		BH	DE	OP	AU	RS		2. Befehl
			BH	DE	OP	AU	RS	3. Befehl

Die einzelnen Phasen eines Befehlszyklus werden dabei von unabhängigen Teilwerken des Prozessors bearbeitet. Der Befehlszyklus ist hier etwas detaillierter dargestellt als im vorangegangenen Abschnitt. Dargestellt ist das Pipelining für Programmbefehle, die keine Änderung des Kontrollflusses zur Folge haben. Die Befehle werden also jeweils nur abgearbeitet und weiter geht es mit dem nächsten Programmbefehl.:

- BH Befehlsholphase
- DE Decodierphase
- OP Operandenholphase
- AU Ausführungsphase
- RS Rückschreibphase

Bei Befehlen, die eine Änderung des Kontrollflusses, also der schrittweisen Abarbeitung des Programmes, zur Folge haben (z. B. bedingte oder unbedingte Sprünge, Befehle die auf Ressourcen warten, oder andere Vorkommnisse, können zu Konflikten im Pipelining führen und dieses stören. Auf Möglichkeiten zur Vermeidung oder Vorhersage solcher Konflikte wird hier nicht weiter eingegangen.

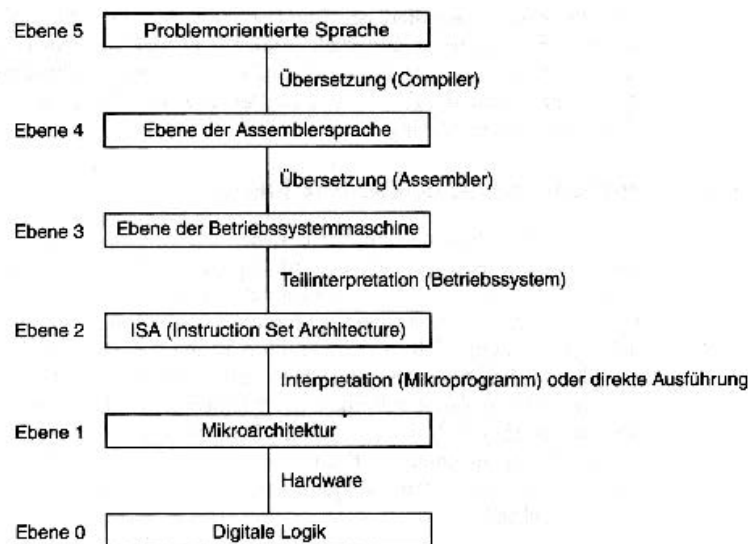
Die Pentium Prozessorfamilie von Intel wird zu den CISC-Architekturen gezählt. Es sind zwar weitgehend die Charakteristika der RISC-Architektur verwirklicht, aber es überwiegen die CISC- Kriterien:

- Relativ wenige Register
- Viele Befehlsformate, viele komplexe Befehle
- Operationen mit Speicheroperanden

7.5 Maschinenbefehle und Mikroprogrammierung

Ein Maschinenbefehl ist eine eindeutig spezifizierte Arbeitsanweisung an den Prozessor, in maschinenlesbarer Form. Er ordnet eine Operation an, die in der Regel mit spezifizierten Daten (Operanden) mit entsprechendem Befehlsformat (i.e. die innere Struktur des Befehls) durchzuführen ist. Die Summe aller in einem Prozessor verfügbaren Maschinenbefehle heißt Befehlssatz oder Befehlsvorrat, ist hardwareabhängig und wird als **Maschinensprache** bezeichnet. Programme, die mit Hilfe von Maschinenbefehlen erstellt werden, heißen Maschinenprogramme.

Unterhalb dieser Maschinenebene, auch ISA-Ebene (**ISA** = Instruction Set Architecture) genannt, schließt sich die Mikroarchitekturebene an. Auf dieser Ebene ist die ALU angesiedelt. Als Verbindung zur ISA-Ebene, als Datenweg, dienen die Register der ALU.



Struktur eines Computers mit sechs Ebenen
Tanenbaum

Hinsichtlich der Steuerung der Operationen auf dem Datenweg muss wieder zwischen RISC- und CISC-Prozessoren unterschieden werden. Bei den CISC-Prozessoren wird ein Befehl, der von Ebene 2 (siehe obige Abbildung) kommt, durch ein Mikroprogramm geprüft, interpretiert, über die Register an die ALU weitergegeben und dort ausgeführt. Beispielsweise würde eine ADD-Anweisung eingelesen, die Operanden gesucht, in die Register gebracht, die Summe von der ALU berechnet und das Ergebnis von der ALU wieder in ein Register geschrieben werden, von wo es dann gelesen werden kann. Ein RISC-Prozessor führt dieselben Schritte aus, jedoch erfolgt die Steuerung des Datenweges nicht durch ein Mikroprogramm, sondern ist festverdrahtet. Dieser Weg ist etwa um den Faktor 10 schneller.

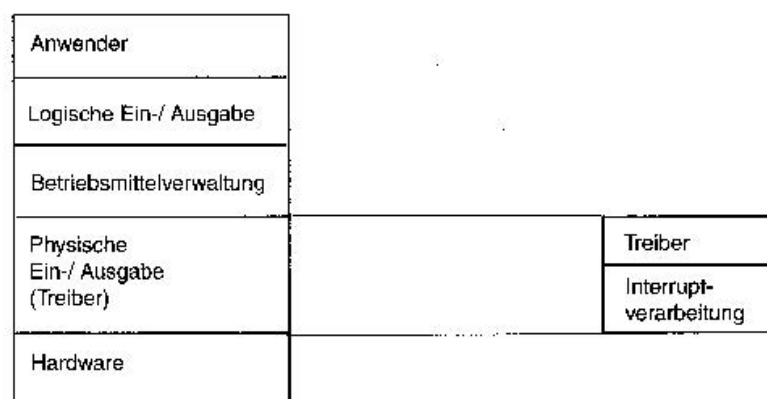
Auf Ebene 0 ist die eigentliche Hardware angesiedelt. Hier befinden sich die Schaltungen, mit denen die Maschinenbefehle aus Ebene 1 ausgeführt werden.

Die Ebene der **Betriebssystemmaschine** ist eine hybride Ebene, d.h. Anweisungen dieser Ebene werden teilweise vom Betriebssystem und teilweise vom Mikroprogramm ausgeführt. Es sind also z.T. dieselben Instruktionen vorhanden wie auf der ISA-Ebene. Allerdings können auf dieser Ebene mehrere Programme parallel ausgeführt werden. Hier laufen die virtuellen Maschinen, welche die Weiterverarbeitung der Anweisungen für die tiefer liegenden Schichten mit Hilfe von Interpretern übernehmen.

Für den "normalen" Programmierer verläuft zwischen Ebene 3 und Ebene 4 die Trennlinie. Die Software für die Ebenen 1 bis 3 wird von Systemprogrammierern geschrieben. Die Ebenen 4 und 5 dienen der Anwendungsprogrammierung in Assembler und Hochsprachen.

7.6 Ein-/Ausgabeorganisation

Die Organisation bzw. Steuerung der Ein-/Ausgabe (E/A) erfordert Kenntnisse der technischen Details und Eigenschaften der E/A-Geräte, um geeignete Schnittstellen erstellen zu können. In vielen Fällen weist das E/A-System eine Schichtenstruktur wie in untenstehender Abbildung auf.



Physisches Ein-/Ausgabesystem

Das physische E/A-System steuert die E/A von Daten zu physisch angeschlossenen Peripheriegeräten. Diese gerätespezifischen Vorgänge werden im Allgemeinen über Gerätetreiber (device driver) abgewickelt. Die Steuerfunktionen zum Ansprechen der Peripheriegeräte sind meist auf E/A-Prozessoren realisiert. Der Zugriff erfolgt

über spezielle E/A-Befehle und Port- Adressen oder über Hauptspeicheradressen (Memory Mapped I/O). Intel-Prozessoren (80x86) verfügen über die Befehle IN und OUT, um E/A-Operationen auszuführen. Diese Befehle senden ein bestimmtes Signal über den Steuerbus, das die Ports empfangsbereit schaltet. Der Hauptspeicher wird dabei ausgeschaltet. Diese Art der Adressierung wird isolated I/O genannt. Motorola-Prozessoren (680x0) verwenden einen definierten Teil des Hauptspeichers für die E/A- Operationen. Bei modernen PC-Steckkarten werden die Portadressen und Interrupts über Software eingestellt (Plug & Play). Die Ports fungieren also als Umsetzer zwischen dem Bussystem und den Peripheriegräten.

Für die Abarbeitung einer E/A-Anforderung durch die CPU gibt es drei grundlegende Mechanismen. In den meisten Fällen geschieht dies durch einen Interrupt, also eine Unterbrechungsanforderung, an die CPU. Die CPU stoppt dabei die aktuelle Programmabarbeitung, sichert ihre Register und springt in die Interruptbehandlungsroutine, auch Interrupthandler oder Interrupt-Service-Routine (ISR) genannt. Wird der Interrupt durch ein Peripheriegerät ausgelöst (z.B. einer seriellen Schnittstelle, weil dort Daten angekommen sind), spricht man von einem Hardwareinterrupt. Registriert werden diese Interrupts von einem speziellen Interruptcontroller auf dem PC-Motherboard. Im Beispiel der seriellen Schnittstelle hätte die ISR die Aufgabe, die Daten von der seriellen Schnittstelle abzuholen. Ist das erfolgt, wird die ISR beendet und die CPU nimmt die normale Programmabarbeitung auf. Die Einsprungsadresse in die ISR ist in der Interrupt-Vektortabelle eingetragen. Die Reihenfolge der Einträge in der Interruptvektortabelle entspricht den Interruptnummern. Wird "Hardwareinterrupt Nr. 3" aufgerufen, weiß die CPU, wo die Einsprungsadresse in der Tabelle steht. Diese berechnet sich nach der Formel $\text{Tabellenplatz} = \text{Offset} + \text{Interruptnummer}$. Der Vorteil dabei ist, dass nur die Interruptnummer, die vom Interruptcontroller ermittelt wird, bekannt sein muss. Da die Interruptvektortabelle im Hauptspeicher steht, kann sie auch geändert werden. Manche Treiber und Dienstprogramme nutzen diesen Umstand, um sich in das System einzuklinken.

Interrupts können aber auch von der Software ausgelöst werden. Diese Softwareinterrupts bilden das zentrale Hilfsmittel zur Kommunikation zwischen Programmen und Betriebssystemfunktionen.

Sollen z.B. Daten an eine serielle Schnittstelle geschickt werden, erfolgt dies über einen Softwareinterrupt. Eine komplette Interruptliste für IBM-kompatible PC's ("Ralf Brown's Interrupt List") gibt es im Internet z.B. unter

<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/ralf/pub/WWW/files.html>

Hinweis

Auf dieser Seiten finden Sie auch zwei Links zu in HTML aufbereiteten Versionen von "Ralf Brown's Interrupt List". Eine dieser Versionen wird in einem bekannten Internetlexikon verlinkt. Dennoch ist derzeit (2010) obiger Link die Seite, die von Ralf Brown selbst gepflegt wird.

Die zweite Möglichkeit, um E/A-Anforderungen zu bearbeiten, ist das zyklische Abfragen einer der der E/A-Einheit zugeordneten Schnittstellen (Register oder Speicherbereiche), das so genannte Polling. Dieses Verfahren ist wegen des aktiven Wartens oft unbefriedigend. Polling eignet sich nur für "langsame" Geräte, wie z.B. dem Drucker.

Eine weitere Möglichkeit zur Bearbeitung von E/A-Anforderungen ist die Befehlssteuerung. Hierbei kommuniziert die CPU direkt mit der Peripherie über spezielle Befehle, die keine E/A- Routinen darstellen. Die CPU ist dabei direkt mit der Peripherie verbunden. Diese Möglichkeit eignet sich nur für sehr schnelle Peripherie, wie z.B. den Gleitkocommacoprozessor.

Logisches Ein-/Ausgabe-System

Das logische E/A-System stellt für die Anwendungsprogrammierung auf der oberen Systemschicht (siehe Kapitel Addition und Subtraktion) Funktionen zur geräteunabhängigen E/A bereit. Es verwendet dazu Funktionen des physischen E/A-Systems. Typische Funktionen des logischen E/A-Systems sind:

Verwaltung logischer Geräte (Festplatte, Floppy, Konsole) Bereitstellung geräteunabhängiger Dateneinheiten und Pufferung (Cluster, Pufferung) Dateiverwaltung auf externen Speichermedien (Dateihandling)

Weiterhin stellt das logische E/A-System spezielle E/A-Dienste außerhalb des Betriebssystem- Kerns zur Verfügung. Virtuelle Laufwerke ("RAMDisks") oder Drucker-SPOOLER (Simultaneous Peripheral Operations Online) sind solche Dienste. Bei langsamen Peripheriegeräten wie z. B. dem Drucker ist es sinnvoll, die auszugebenden Daten zunächst auf einem schnellen Zwischenspeicher, beispielsweise der Festplatte, zwischenzulagern und dann mit Hilfe eines Hintergrundprozesses, dem Druckerspooler, nach und nach auszugeben.

Normalerweise verwendet ein Anwendungsprogrammierer die komfortablen E/A-Funktionen des logischen E/A-Systems und nicht die des physischen E/A-Systems.

Betriebsmittelverwaltung

Unter Betriebsmitteln versteht man alles, was ein Prozess (siehe Kapitel Begriffserklärung) zur Laufzeit benötigt. Dabei wird zwischen Hard- und Softwarebetriebsmitteln unterschieden. Hardwarebetriebsmittel sind z.B. die CPU, Sektoren auf der Festplatte oder auch Nachrichtenpuffer. Unter Softwarebetriebsmitteln versteht man z.B. Dateien, Botschaften oder Ereignisse, auf die der Prozess wartet. Die Betriebsmittelverwaltung ist Aufgabe des Betriebssystems. Beim Aufruf einer logischen E/A- Funktion, beispielsweise einer Druckerausgabe, muss dem SPOOL-Prozess Speicherplatz auf der Festplatte zugeteilt werden. Hierin liegt die Aufgabe der Betriebsmittelverwaltung.

7.7 Multimedia-Peripherie

Medien (engl. media) sind Mittel zur Darstellung und Verbreitung von Informationen. Unter **Multimedia** versteht man eine Informationsumgebung, die mehrere menschliche Sinne gleichzeitig anspricht. Sie dient der Präsentation und Wahrnehmung und wird auf der Basis von Hard- und Software realisiert. Wichtige Merkmale von Multimedia sind:

- Integration verschiedener Medientypen (Text, Grafik, Stillbild, Animation, Applikation zur Ein-/Ausgabe, Video und Ton) auf einem digitalen Träger
- Computergestützte einheitliche Abspieleinheit zur Erzeugung, Bearbeitung, Speicherung und Übertragung von Informationen
- Auf die verschiedenen Informationsanteile kann einzeln und interaktiv zugegriffen werden,

wobei

- mindestens ein Medium als Echtzeit(daten)strom (kontinuierlicher Lieferant von Daten) laufen können muss.

Der ursprünglich von namhaften Hard- und Softwarefirmen konzipierte "Multimedia-PC" (MPC) ist durch das heutige Marktangebot längst überholt. Jeder "Supermarkt-PC" enthält diese Komponenten oder lässt sich leicht um diese erweitern. Zur Ausstattung eines multimediafähigen Rechners gehören:

- Ein schneller Prozessor der jeweils aktuellen Generation, inzwischen (Stand 2009) auch mit mehreren Kernen,
- viel schneller Hauptspeicher und große Festplatte(n), weil Dateien, die Multimedia-Daten

enthalten oft recht groß sind.

- eine Farbgrafikkarte, die möglichst Farben mit mindesten 24 Bit pro Pixel und in möglichst hoher Auflösung anzeigen kann,
- ein oder mehrere schnelle CD/DVD-ROM oder BD-ROM Laufwerke, wobei BD für Blu-ray Disc

™ steht, mindestens einen CD/DVD-oder BD-Brenner für BD-RE (Blu-ray Disc - REwritable) Medien,

- eine gute Audiokarte mit MIDI-Anschluß (Musical Instrument Digital Interface), Synthesizer

sowie Mikrofon und mindestens Stereolautsprechern oder X.1 System, wobei X derzeit (Stand 2009) für 5 oder 7 steht,

- eine Tastatur und Maus, eventuell eine 3D-Maus für CAD-Anwendungen,
- ein oder mehrere große Displays mit hoher Auflösung,
- geeignete Betriebssystemsoftware sowie
- optional eine Videoschnittkarte (Frame-Grabber-Karte) mit analoger oder digitaler Videokamera.

Es handelt sich hierbei um eine Ausrüstung, die auf die Darstellung und Speicherung multimedialer Daten ausgelegt ist. Für die Modifikation oder Produktion multimedialer Daten müssen gegebenenfalls High-End-Komponenten verwendet werden oder Rechnersysteme, die nicht mehr zu den PCs gezählt werden. Im folgenden soll auf die technischen Merkmale einiger Komponenten näher eingegangen werden.

Prozessor

Die CPU muss schnell genug sein, um die Verarbeitung der digitalisierten Daten in Echtzeit zu bewerkstelligen. Die Problematik besteht in der Bewältigung des hohen Datendurchsatzes bei kontinuierlichen Medien, z.B. 22MByte/s bei Video. In modernen Prozessoren sind spezielle Multimedia-Funktionen bereits integriert (MMX).

Audiointerface

Die Aufgabe der Audiokarte besteht in der Analog-Digital-Wandlung und umgekehrt von Eingangssignalen, je nachdem ob eine Aufnahme oder Wiedergabe erfolgt. Das Midi-Interface dient zum Anschluss geeigneter Musikinstrumente. Der Synthesizerchip kann direkt oder über die MIDI- Schnittstelle Instrumentenklänge mehrstimmig erzeugen. Wichtige technische Kenngrößen für Audiokarten sind:

- Die Abtastrate, d.h. die Frequenz, mit der das analoge Signal abgetastet wird. Dabei muss

die Abtastfrequenz etwas größer sein als das doppelte der höchsten Frequenz im abgetasteten Signal (Abtasttheorem von Shannon). Eine Abtastfrequenz von 44,1KHz ist ausreichend (CD-Spieler)

- Die Abtasttiefe oder Auflösung, d.h. wie viele Bits werden verwendet um den Funktionswert

des abgetasteten Signals angenähert darzustellen (z.B. 8 Bit oder besser 16 Bit)

Bildschirm

Heutige Monitore werden teils als zentrale analoge Ausgabegeräte nach Umsetzung der digitalen Daten in der Grafikkarte über analoge RGB-Signale (Rot, Grün, Blau) angesteuert, teils auch als digitale Ausgabegeräte direkt mit digitalen Daten angesteuert, z. B. über den DVI oder den HDMI Anschluss. Wichtige Kenngrößen sind:

- Der Abstand der Bildpunkte oder die Feinheit der Lochmaske.
- Die Bildwiederholfrequenz oder Vertikalfrequenz, d.h. die Frequenz, mit der die Bildschirmseite komplett aufgebaut wird, und zwar ohne Zeilensprung (non-interlaced). Sie sollte mindestens 75Hz betragen, damit ein flimmerfreies Bild erzeugt wird (Arbeitsschutz- Richtlinien für Bildschirmarbeitsplätze)
- Die Bildschirmauflösung, also die maximale Anzahl der auf dem Monitor darstellbaren Bildpunkte.
- Die Farbtiefe oder die Anzahl der verwendeten Bits pro Bildpunkt. Das Minimum ist hier True-Color mit 2^{24} Farben (entsprechend 4 Mio. Farben). Damit werden pro Bildpunkt (Pixel) 24 Bit für die Speicherung seiner Farbe verwendet.
- Ergonomie und hier insbesondere der Strahlenschutz (TCO 95/99, MPR II, TÜV)

Videoerfassungskarte

Die Hauptaufgabe der Videoschnittkarte besteht in der Abtastung des Videosignals in Echtzeit sowie in der Komprimierung der Daten. Die meisten Karten sind in der Lage, die komprimierten Daten zur Darstellung wieder zu dekomprimieren. Außerdem wird die Tondigitalisierung von der Karte mit übernommen.

Digitale Videokamera und Fotokamera

Digitale Video- und Fotokameras haben in den letzten Jahren eine rasante Entwicklung erfahren. Die Bildqualität hat in den letzten 10 Jahren um das zehnfache zugenommen. Das ist mehr, als es bei der analogen Fotografie in 150 Jahren der Fall war. Beispielsweise sind bei Bildformaten bis 18 x 13 Zentimeter und 4 Megapixeln digitale Kameras gleichwertig mit analogen Kameras. Eine analoge Fotokamera erreicht maximal eine Auflösung von 100 Pixel pro Millimeter.

Das wichtigste Bauelement in den Digitalkameras ist die Bildabtasteinheit. Technologisch existieren zwei Ansätze, CCD und CMOS. **CCD** steht für Charged Coupled Device. Der rechteckige CCD-Chip ist in viele Pixel (Picture Elements) aufgeteilt. Jedes dieser Pixel besteht aus einer Photodiode, die in ihrer Gesamtheit auf dem Chip eine Matrix bilden. Fällt Licht auf eine Diode, werden elektrische Ladungen erzeugt. Dabei ist die Anzahl der aufgetroffenen Photonen ein Maß für die sich aufbauende Spannung. Die Höhe der Spannung wird in einen Zahlenwert umgesetzt und kann so weiter verarbeitet werden. Der Wertebereich dieser Zahlen hängt von der Dynamik der Kamera ab. Hat die Kamera eine Dynamik von 12-Bit, so können 4096 Werte unterschieden werden. Farbsensitivität kann z.B. durch Filter erreicht werden. Da CCD-Chips auch infrarotes Licht empfangen können, finden sie Verwendung in der Astronomie. Die Spannung der Sensorelemente wird beim Auslesen mit einer geschalteten Kette von Kondensatoren (Charge Coupling) an den Ausgang weitergereicht.

Die **CMOS**-Bildsensoren arbeiten nach demselben Prinzip wie CCD-Sensoren. Jedoch können bei den CMOS-Sensoren einzelne Bereiche direkt adressiert und ausgelesen werden, um z.B. ein Bild mit geringerer Auflösung zu erhalten. Daraus ergibt sich ein Geschwindigkeitsvorteil, da bei den CCD-Chips immer die kompletten Zeilen oder Spalten der Matrix ausgelesen werden müssen. Weitere Vorteile von CMOS-Sensoren liegen in den geringeren Produktionskosten und der niedrigeren Leistungsaufnahme. Ein gravierender Nachteil der CMOS-Sensoren liegt in den langen Datenleitungen zu den einzelnen Dioden. Dadurch werden diese Bauteile anfällig für Rauschen und weisen einen geringeren Dynamikumfang aus.

Die genannten digitalen Videoekameras nehmen die Bilddaten entsprechend eines **Standards für digitales Video** auf Magnetband, Festplatten, optischen Platten oder Halbleiterspeicher auf. Es werden die Bilddaten der Bildpunkte (Pixel) bei Amateurgeräten meist mit 8 Bit quantisiert (die

Helligkeit wird dabei mit räumlicher höherer Auflösung abgetastet als die Farbigkeit, da das menschliche Auge empfindlicher auf Helligkeitsunterschiede reagiert als auf Farbunterschiede), anschliessend komprimiert und dann gespeichert bzw. übertragen. Einige Fakten:

- Gängige Bildgrößen, waren/sind 720×576 (PAL, 50Hz) oder 720×480 (NTSC, 60Hz).
- Aufzeichnungsmodi sind z. B. 50i (interlaced, also 50 Halbbilder pro Sekunde für das PAL-Signal), 25p (progressive, also 25 Vollbilder pro Sekunde; aus diesem Modus kann leicht das PAL-Signal erzeugt werden oder er kann auf digitalen Bildschirmen direkt angezeigt werden. (Es gibt auch noch weitere: 24p, 50p, und andere).

- MJPEG ist eine für die Aufzeichnung von Videodaten gut geeignete Kodierung, da jedes

einzelne Bild mit JPEG kodiert wird, und somit die Videobildfolge nach jedem Bild geschnitten werden kann; das DV-Format mit einem Datenstrom von 25 MBit/s bei einem Komprimierungsfaktor von ca. 1:5 ähnelt sehr dem MJPEG-Verfahren. Hierbei fallen jedoch noch relativ viele Bilddaten an. Dagegen eignet sich z. B. MPEG-2 wegen der höheren Komprimierung (Grundprinzip: Es werden Schlüsselbilder komprimiert und bei Folgebildern nur die Bildunterschiede komprimiert gespeichert.) besser für die reine Speicherung oder Übertragung von Daten (typische Datenraten sind 5 bis 10 MBit/s). Das Schneiden der Videobildfolge ist allerdings nur direkt vor dem nächsten Schlüsselbild ohne Qualitätsverlust möglich; jedoch sollte dieser Nachteil bei ausreichender Bildqualität unerheblich sein (hier sind Datenraten bis zu 35 MBit/s für HD-Video üblich).

Digitales Fernsehen legt nur die Abtastung und Quantisierung fest, auf die Komprimierung müssen sich die Anbieter untereinander verständigen.

Bei Komprimierung und Speicherung wiederum wird nur die Dekomprimierung genormt, somit bleibt bei der Komprimierung und den Datenformaten noch Innovationsspielraum.

Auf hochauflösendes Digitales Video wird an dieser Stelle (noch) nicht eingegangen.

7.8 Bussysteme

Unter einem **Bus** versteht man Verbindungsstrukturen zur Kommunikation mehrerer Busteilnehmer untereinander über Sammelleitungen, dem so genannten **Bussystem**. Im Gegensatz dazu stehen direkte Verbindungen von einzelnen Komponenten, z.B. über eine serielle Schnittstelle (RS-232). Der Vorteil der Bussysteme liegt u.a. im erheblich geringeren Aufwand für die Hardware, insbesondere der Verdrahtung. Allerdings ist der Softwareaufwand zur Datenübertragung bei einem Bussystem höher als bei den direkten Verbindungen. Bei mehreren Busteilnehmern muss eindeutig festgelegt sein, welcher Teilnehmer senden darf, wer der Empfänger ist usw.. Zugriffskonflikte müssen vermieden oder aufgelöst werden. Der Teilnehmer, der Senderechte erhält, ist der **(Bus-)Master**, die übrigen Teilnehmer sind die **Slaves**. Die Strategie, mit der Senderechte verteilt werden, heißt Bus-Arbitrierung (engl. Arbitration: Konkurrenzbereinigung, Priorität). Außerdem müssen feste Regeln bezüglich der (zeitlichen) Funktionsweise des Bussystems eingehalten werden. Diese

Regeln nennt man **Busprotokoll**. Bezüglich der Bustaktung muss zwischen synchronen und **asynchronen Bussen** unterschieden werden. Die synchronen Busse arbeiten synchron zum CPU-Takt, während die asynchronen Busse unabhängig vom CPU-Takt mit variabler Frequenz arbeiten können und somit z.B. für Mehrprozessorsysteme geeignet sind. Außerdem bremst bei einem asynchronen Bus ein langsamer Teilnehmer schnellere nicht aus. Als **Busbreite** bezeichnet man die Anzahl der Bits, die pro Takt parallel übertragen werden können. Die Geschwindigkeit und Effizienz eines Bussystems beeinflusst das Leistungsverhalten eines Rechners maßgeblich.

Ein wesentlicher Grund für den Siegeszug der PC's und einiger Workstations in den letzten Jahrzehnten liegt in der Standardisierung der Erweiterungsslots. Dadurch wurde die preiswerte Massenproduktion von Steckkarten für diese Bussysteme möglich. Ein IBM-kompatibler PC enthält folgende Bussysteme:

- Interner Bus der CPU.
- Systembus (s.u.) oder Memory-Bus, der eine schnelle Verbindung von der CPU zum Hauptspeicher herstellt (**Front Side Bus**)
- Grafik-Bus (**AGP**: Accelerated Graphics Port)
- **PCI-Bus** (PCI: Peripheral Component Interconnect) zur schnellen Verbindung mit den Erweiterungsslots. Der PCI-Bus hat eine Busbreite von 32 oder 64 Bit (PCI-2) für Adressen und Daten. Er wird mit 33 oder 66 MHz (PCI-2) getaktet und erreicht einen Datendurchsatz von 132 MByte/s bzw. 528 MByte/s (PCI-2).
- **ISA-Bus** (Industry Standard Architecture). Ein veralteter und aussterbender 16 Bit-Bus mit 8 MHz Taktfrequenz.
- Gerätebus als Verbindung zu Peripheriegeräten, wie der **IDE-Bus** zur Verbindung mit Festplatten und CDRom, oder der **USB** (Universal Serial Bus) zur Verbindung mit externen Geräten wie Scanner, Drucker, Fotokamera.

Systembus

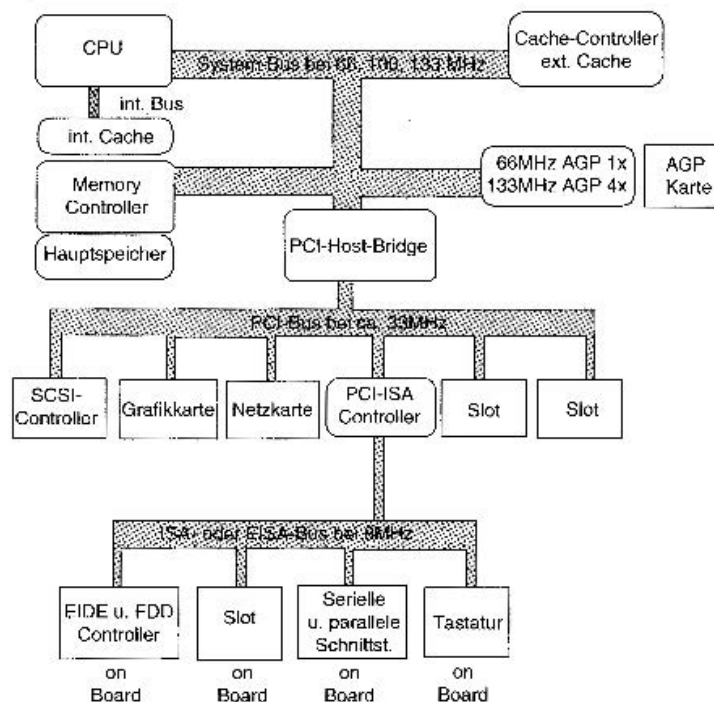
Über den Systembus läuft die gesamte rechnerinterne Kommunikation der daran angeschlossenen Hardware - Komponenten (Omnibus; lat. "Bus für alle"). Ein Bus besteht aus parallelen Leitungen, über die jeweils einzelne Binärstellen parallel übertragen werden. Die Leitungen werden zu Adress- , Daten- und Steuerleitungen gruppiert.

Adressleitungen: Hier wird die Adresse des Hauptspeichers oder der E/A - Schnittstelle binär codiert angelegt, zu dem/der oder von dem/der ein Datum übertragen werden soll. (Adressbreite / Anzahl der Leitungen : z.B. 16, 32)

Datenleitungen: Übertragung der binären Nutzdaten zu oder von der angelegten Adresse. (Datenbreite : z.B. 8, 16, 32)

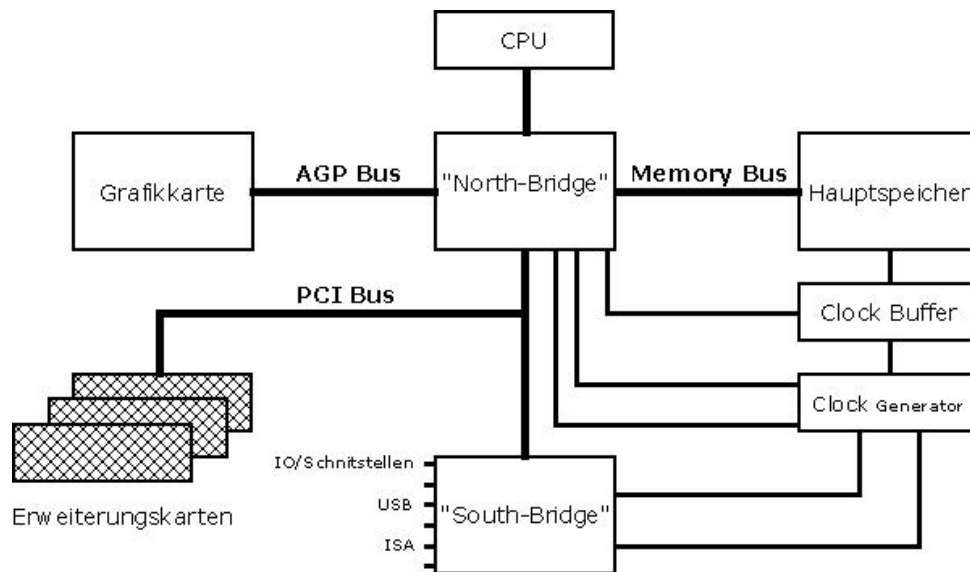
Steuerleitungen, z.B. Taktleitung, Lese-, Schreib- und Steuerleitung (Anzahl der Steuerleitungen z.B. 4)

Die untenstehende Abbildung zeigt die Bussysteme in einem älteren IBM-kompatiblen PC. Allerdings fehlen in dieser Abbildung die Gerätebusse.



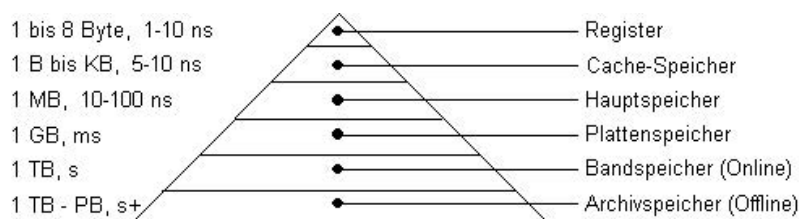
Bussysteme im IBM-kompatiblen PC

Um verschiedene oder gleiche Busse miteinander zu verbinden, werden Brücken ("Bridges") verwendet. Auf modernen Motherboards sind meist zwei dieser Brücken auf zwei Chips enthalten. Hierfür haben sich die Namen North- und South-Bridge eingebürgert. Die North-Bridge verwaltet die Datenübertragung zwischen CPU, Hauptspeicher und AGP-Port, während an der South-Bridge die Ein-/Ausgabeschnittstellen, USB, Keyboard-Controller, Floppy und IDE-Controller angeschlossen sind.


North- und South-Bridge

7.9 Speichertechnologien

Auf der Ebene der Modellierung eines Computers als von Neumann-Rechner gibt es ein hier nicht näher spezifiziertes Speicherwerk. In realen Computern gibt es verschiedene interne und externe Speicher, die bezüglich ihrer Zugriffszeit und ihrer Größe in einer Speicherhierarchie angeordnet werden (untenstehende Abbildung).


Speicherhierarchie

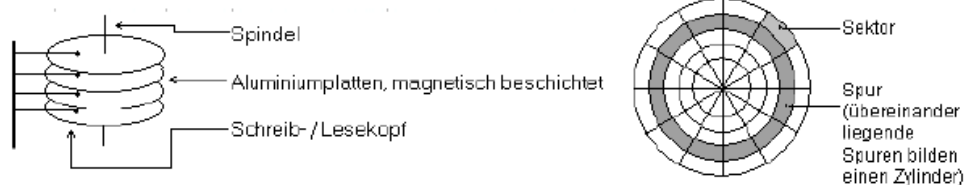
Charakteristische Merkmale:

Bezeichnung	Kapazität	Zugriffszeit	Bauteil	int./ext.
Register	1 bis 8 Byte	1-10 ns	Prozessor	intern
Cache	bis 1 KB	5-10 ns	Prozessor / RAM	intern
Hauptspeicher	1 MB	10 - 100 ns	ROM / RAM	intern

Plattenspeicher	1 GB	ms	Festplatte	extern
Bandspeicher	1 TB	sec.	Bandlaufwerk	extern
Archivspeicher	1 TB - PB	sec. - min.	diverse	extern

Die internen Speicher sind Halbleiterspeicher. Es handelt sich dabei um elektronische Speicher, im Endeffekt um Kondensatoren, deren Ladungszustand periodisch aufgefrischt werden muss (Refresh). Mit Ausnahme des ROM's gehen bei diesen Speichern bei Spannungsverlust auch die gesamten Daten verloren. Man spricht von flüchtigen Speichern.

Die externen Speicher sind optische oder magnetische Speichermedien, welche als Platten, Plattenstapel (untenstehende Abbildung) oder Bänder organisiert sind. Den externen Speichern ist im Gegensatz zu internen Speichern gemein, dass Speicherwörter nicht direkt adressiert werden können. Bei Platten ist der Schreib-/Lese-Kopf auf der Spur zu positionieren und dann aus dem Dateninhalt der gesamten Spur sequentiell das gesuchte Datum herauszufiltern. Bänder können nur sequentiell beschrieben und gelesen werden. Hieraus resultiert die um mehrere Größenordnungen höhere Zugriffszeit.



Plattenstapel, Struktur einer Festplatte

Handelsübliche **Festplatten** erreichen Drehzahlen bis 7200 Umdrehungen pro Minute. Aufgrund der auftretenden Fliehkräfte besteht das Trägermaterial der magnetisch beschichteten Platten aus Leichtmetall. Die Arme der Schreib-/Leseköpfe werden mit Schrittmotoren positioniert. Der Kopf besteht aus einer kleinen Spule mit Eisenkern und besitzt ein flugfähiges Profil. Sobald die Platten

nach dem Einschalten ihre Nenndrehzahl erreicht haben, wird die mechanische Verriegelung der Arme gelöst, und die Köpfe "fliegen", im Bereich von Mikrometern, über den Platten. Die magnetisierten Bereiche der Platte, nämlich die Sektoren, kann man sich als Aneinanderreihung von kleinen Stabmagneten vorstellen. Läuft ein solcher Bereich am Kopf vorbei, wird in den Spulenwindungen ein Strom induziert, dessen Richtung

davon abhängt, ob gerade ein Nord- oder ein Südpol "vorbeifliegt". Zum Schreiben wird ein höherer Strom durch die Spulenwindung geschickt und über das entstehende Magnetfeld der Plattenbereich unter dem Spulenkopf aufmagnetisiert. Somit sind die logischen Schaltzustände "0" und "1" darstellbar.

Der früher gefürchtete **Headcrash**, d.h. ein Aufsetzen der Köpfe auf die drehenden Platten mit fatalen Folgen, tritt bei modernen Festplatten sehr selten auf. Wird die Platte von der Versorgungsspannung abgetrennt, läuft der Antriebsmotor der Platten generatorisch und liefert so die Energie, die benötigt wird, um die Arme in die mechanische Verriegelung zu fahren. Moderne Festplatten halten Beschleunigungen bis ca. 50g (g: Erdbeschleunigung= $9,81 \text{ m/s}^2$) stand. Dabei wird eine Anregung mit einer Sinushalbwellenlänge für die Dauer von 2ms zu Grunde gelegt. Zum Vergleich, ein kräftigerer Faustschlag auf den PC bewirkt ca. 20g.

Diskettenlaufwerke arbeiten nach demselben Prinzip wie Festplatten. Allerdings liegen hier die beiden Schreib-/Leseköpfe auf der Oberfläche auf. Die oberste Schicht besteht aus einem reibungsarmen Material, das zusätzlich in der Lage ist, Staubpartikel aufzunehmen. Die Drehzahl liegt bei 360 Umdrehungen pro Minute.

Als weitere Massenspeicher haben sich optische Speichermedien, also **CD** und **DVD** (Digital Versatile Disk), etabliert. Die Informationen werden entlang einer spiralförmigen Spur eingetragen, die von innen nach außen verläuft. Beim Lesen tastet ein fokussierender Laserstrahl die Spur mit konstanter Lineargeschwindigkeit ab. Je nach CD-Typ werden Phasenverschiebungen im Strahlengang, Änderung des Reflexionsfaktors oder unterschiedliche Dämpfung bezüglich der Helligkeit ausgenutzt. Die industriell gefertigten **Audio-CD's** oder **CD-ROM's** enthalten Vertiefungen (engl. Pits) entlang der Spur. Der Laser fokussiert auf die höhere Ebene (engl. Lands), die $1/4$ der Wellenlänge höher liegt als die Pits. Trifft der Laserstrahl auf ein Pit, so erfolgt in der Optik des CD-Laufwerkes eine Auslöschung des Strahls. Trifft der Strahl auf ein Land, so wird er normal reflektiert. Bei den einfach beschreibbaren **CD-R's** erhitzt der Laser eine über der Reflektionsschicht liegende Farbschicht und macht diese undurchsichtig. Beim Lesen wird der Laserstrahl an dieser Stelle in seiner Helligkeit gedämpft. Wiederbeschreibbare CD's, **CD- RW's**, besitzen eine spezielle Schicht, die je nach Erhitzungstemperatur in einen amorphen (d.h. es gibt keine regelmäßigen Kristallstrukturen) oder kristallinen Zustand abkühlt. Der unterschiedliche Reflektionsgrad der beiden Zustände wird beim Lesen ausgewertet. Zwischen den Zuständen kann mehrfach gewechselt werden.

7.10 Leistungsgrößen und Leistungsbewertung

Die Leistungsbewertung eines Rechners kann grundsätzlich über den Zeitbedarf für die Erledigung einer bestimmten Aufgabe erfolgen. Allerdings muss hierbei berücksichtigt werden, welcher Art die Aufgabe war bzw. wie stark die verschiedenen Funktionseinheiten des Rechners mit der Aufgabe belastet wurden. Die einzelnen Funktionseinheiten sind hierbei:

- CPU
- Hauptspeicher und Cache
- Ein-/Ausgabesystem, insbesondere die Massenspeicher Interne und externe
- Bussysteme

Landläufig trifft man oft auf die Ansicht, dass die Rechnerleistung im Wesentlichen von der Taktfrequenz der CPU abhinge, nach dem Motto "viel hilft viel". Bei einem handelsüblichen PC kann man die Faustformel angeben, dass eine Verdopplung der CPU-Taktfrequenz ca. 15-30% mehr Systemleistung bringt, die dann nach außen wirklich sichtbar wird. Die Größe und die Zugriffszeiten von Hauptspeicher und Cache sowie die Leistungsfähigkeit des Ein-/Ausgabesystems und der Bussysteme sind meist viel entscheidender.

Um Rechner mit verschiedenen Architekturen untereinander bezüglich ihrer Leistung vergleichen zu können, werden Benchmark-Programme verwendet. Die wichtigsten Anforderungen an **Benchmark**-Programme sind:

- Relevanz: Es sollen typische Operationen gemessen und spezielle Eigenschaften vermieden werden
- Portabilität: Leichte Implementierung auf verschiedenen Systemen
- Einfachheit: Das Programm sollte für alle verständlich sein
- Skalierbarkeit: Das Programm sollte auf kleinen und großen Rechnern anwendbar und vergleichbar sein
- Verifizierbarkeit: Die Ergebnisse sollen für Dritte nachvollziehbar und überprüfbar sein

Weiterhin muss beachtet werden, dass Benchmarks nur einzelne Funktionseinheiten des Rechners auf ihre Leistungsfähigkeit überprüfen, jedoch nie das gesamte System. Aber selbst hierbei können widersprüchliche Ergebnisse auftreten. Beispielsweise wird die **CPU-Leistung** mit dem heute nicht mehr so gebräuchlichen **MIPS**-Benchmark (Million Instructions Per Second) angegeben. Die Widersprüchlichkeit dieser Größe soll in zwei Beispielen verdeutlicht werden.

1. Gegeben sei die Formel $y=mx+b$. Drei Rechner A, B und C benötigen für diese Aufgabe genau 1ns. Der Rechner A hat eine typische RISC-Architektur und benötigt 10 Maschinenbefehle zur Lösung. Rechner B benötigt 5 Maschinenbefehle und Rechner C 2 Maschinenbefehle. Somit hätte A eine MIPS-Rate von 10, B von 5 und C von 2, obwohl alle Rechner die Aufgabe in derselben Zeit gelöst haben. Das Ergebnis suggeriert also, dass Rechner A eine 5-mal höhere Leistung als C hätte.

2. Es sollen 16 Bytes im Hauptspeicher verschoben werden. Es stehen wieder 3 Rechner zur Verfügung, mit drei unterschiedlichen Prozessoren. Der erste Rechner mit einem 6502-Prozessor benötigt 65 Maschinenbefehle, der zweite Rechner mit einem 68000-Prozessor benötigt 35 Maschinenbefehle und der dritte Rechner mit einem /370-Prozessor (IBM) benötigt einen Maschinenbefehl. Alle Rechner benötigen wieder 1ns zur Lösung. Somit hätte der 6502-Rechner 65-MIPS, der 68000-er 35-MIPS und der /370 1 MIPS. Dabei ist der /370-Prozessor der absolut leistungsfähigste.

Nach verbesserter Definition wird bei dem MIPS-Benchmark noch die mittlere Anzahl der Taktzyklen pro Befehl, die mittlere Speicherzugriffszeit und der Speicherbedarf pro Durchschnitts- Befehl berücksichtigt. Er bleibt aber trotzdem eher ein Maß für die maximale als für die reale CPU- Leistung.

Der zweite bedeutende Gütefaktor für einen Rechner ist die **Hauptspeicher-Effizienz**. Wird ein Programm in den Speicher geladen, stellt sich die Frage, wie viele Bytes das Programm dort belegt. Obwohl Rechner heutzutage meist üppig mit Speicher ausgestattet sind, ist die Frage der Hauptspeicher-Nutzung immer noch aktuell, da zwischen CPU und Hauptspeicher die Cache- Speicher geschaltet sind. Die Schnelligkeit des Datenverkehrs zwischen CPU und Cache einerseits

und zwischen Cache und Hauptspeicher andererseits sind die kritischen Stellen. Weiterhin spielen die Struktur und Breite der Maschinenbefehle eine wichtige Rolle, also die Frage nach RISC oder CISC.

Der dritte und oft unterschätzte Gütefaktor ist die **Ein-/Ausgabeleistung**. Hierbei ist die Antwortzeit oder Latenzzeit (Festplatten und CD-ROM- Hersteller geben die mittlere Zugriffszeit an) und der Datendurchsatz, auch E/A-Bandbreite genannt, von entscheidender Bedeutung. Bei steigendem Datendurchsatz nimmt die Latenzzeit exponentiell zu.

Da sich der MIPS-Benchmark als untauglich erwiesen hatte und Hardware-Hersteller dazu übergingen, speziell auf ihre Hardware zugeschnittene Benchmark-Programme zu präsentieren, wurden Rufe nach standardisierten Verfahren laut. Nach großen Streitigkeiten mussten die Hersteller diese Verfahren akzeptieren. Das erste

standardisierte Verfahren war der **Whetstone- Benchmark**. Er wurde 1976 zunächst als ALGOL-60-, dann als FORTRAN-Version veröffentlicht. Das Programm enthält genau eine Million maschinenunabhängiger so genannter Whetstone- Befehle. Gemessen wurde die Zeit, die ein Rechner benötigt, um diese Befehle abzuarbeiten. Die Leistung wird dann in MWIPS (Mega Whetstone Instructions Per Second) angegeben. Ein Rechner, der 5 Sekunden zur Abarbeitung benötigt hätte, hätte also 0.2 MWIPS erzielt. Da der Whetstone- Benchmark einen relativ hohen Anteil an Gleitkomma-Berechnungen beinhaltet, ist er nicht zur Leistungseinschätzung jeder Rechnerarchitektur geeignet. Für Rechner ohne Fließkommaeinheit wurde 1984 der Dhrystone-Benchmark entwickelt. Die Urversion war in 'Ada' geschrieben, heute wird eine C-Version verwendet. Das Programm besteht aus lediglich 100 Befehlen und passt bei 2- 3 kbyte-Größe in jeden Cache-Speicher. Der **Dhrystone-Benchmark** besteht zu 50% aus Zuweisungen und Ausdrücken mit maximal 3 Operanden, zu 33% aus Fallunterscheidungen, Schleifen und Kontrollstrukturen und zu 17% aus Funktionsaufrufen. Infolge der unterschiedlichen Rechnerstrukturen bereitet eine Umcodierung des Programms in Assembler einige Schwierigkeiten. Aus diesem Grund werden Hochsprachen verwendet. Bei der Ausführung des Programms wird also immer der Rechner mit dem zugehörigen Compiler getestet. Das führt dann dazu, dass eigentlich die Leistungsfähigkeit bzw. die Optimierungsfähigkeiten des Compilers bewertet werden. Hardware- Hersteller sehen das aber nicht ungern. Weitere gebräuchliche Benchmark-Programme sind **Linpack**, für Supercomputer mit Vektor-Funtionseinheiten, die **SPEC-Benchmarks** für unterschiedliche Rechnerarchitekturen und die **TPC-Benchmarks** als Testprogramme für die Leistungsfähigkeit von Datenbanken. Des weiteren wurden spezielle Hard- und Softwaremonitore entwickelt, mit denen eine Leistungsüberwachung spezieller Funktionseinheiten möglich ist.

7.11 Konzepte der Parallelverarbeitung



Gliederung

7.11 Konzepte der Parallelverarbeitung

7.11.1 SIMD-Architekturen

7.11.2 MIMD-Architekturen

Prozessoren werden zwar immer schneller, jedoch rücken die physikalischen Grenzen, nämlich die Lichtgeschwindigkeit und quantenmechanische Effekte, auch immer näher. Extrem rechenintensive Simulationen, wie z.B. die Wettervorhersage, Crash-Tests in der Kfz-Industrie, Design von neuen Medikamenten usw. sind mit einem Ein-Prozessor-Rechner, wie einem IBM-kompatiblen PC, allein in einer vernünftigen Zeitspanne unmöglich. Das wird sich in absehbarer Zukunft auch nicht ändern. Zur Umgehung des Problems bleibt derzeit nur die Möglichkeit, mehrere CPU's zu **Mehrprozessorsystemen** (s.u.) oder mehrere Rechner zu **Mehrrechnersystemen** (s.u.) zusammenzuschalten oder Mischformen zu bilden. Das Ziel ist natürlich, den Rechner an eine gegebene Aufgabenstellung bezüglich der Gesamtleistung und/oder der Kosten optimal anzupassen. Die Verbindungsmethode der CPU's unterscheidet sich in statische und dynamische Verbindung. Bei der statischen Methode werden alle Komponenten fest mit einander verbunden, z.B. als Ring, Kreis, Gitter usw.. Bei der dynamischen Methode werden alle Komponenten über ein Vermittlungsnetz zusammengeschaltet und die Daten innerhalb des Netzes dynamisch (d.h. die Wege können unterschiedlich sein) weitergeleitet. Die statische Verbindungsmethode stellt im Prinzip eine Punkt-zu-Punkt Verbindung her, ähnlich dem Telefonfestnetz, während bei der dynamischen Verbindung die Daten in Pakete aufgeteilt werden, die dann einzeln und evtl. auf unterschiedlichen Wegen durch das Vermittlungsnetz geschickt werden, ähnlich dem Mobilfunknetz oder dem Internet.

Bezüglich der konkreten Softwareverarbeitung wird bei Parallelrechnern zwischen drei Grundtypen unterschieden. Zum einen können sie aus einem Zentralrechner mit vielen angeschlossenen Benutzerstationen bestehen. Dabei kann der Zentralrechner aus einem Computer mit mehreren CPU's bestehen und die Benutzerstationen können ebenfalls eine eigene CPU und lokalen Speicher besitzen. Bei einem solchen System führt jede Benutzerstation ein eigenes Programm auf dem Zentralrechner aus. Die Benutzerstationen teilen sich also die Rechenzeit des Zentralrechners. Solche Systeme werden **Timesharing-Systeme** genannt. Typische Anwendungsbeispiele sind z.B. Buchungssysteme von Fluggesellschaften, Bankterminals oder große Webserver. Ein weiterer Typ ist ein Parallelrechner, auf dem ein einziges Programm abläuft, das aus mehreren parallelen Prozessen besteht, die über die CPU's verteilt werden. Hier steht die Beschleunigung der Problemlösung im Vordergrund. Der dritte Typ sind numerische Supercomputer. Sie bestehen aus vielen ALU's, die alle mit demselben Instruktionsstrom arbeiten. Die letzten beiden Typen setzen natürlich eine parallelisierbare Aufgabe voraus. Die Unterschiede zwischen den drei Grundtypen sind in der Praxis fließend.

Mehrprozessorsysteme

Hierbei teilen sich alle CPU's einen gemeinsamen Speicher. Bei einem Mehrprozessorsystem werden mehrere Prozessoren von einem einzigen Betriebssystem gesteuert. Alle Prozesse können auf den zentralen Speicher zugreifen und untereinander kommunizieren. Prozess A kann Prozess B auffordern, die von A an einer bestimmten Stelle im Speicher abgelegten Daten zu lesen. Solche Systeme sind bei Programmierern wegen ihrer Überschaubarkeit und der Möglichkeit der Anwendung auf zahlreiche Probleme beliebt.

Mehrrechnersysteme

Mehrrechnersysteme bestehen aus weitgehend autonomen Rechnern mit eigenen Betriebssystemen. Hierbei besitzt jede CPU einen lokalen Speicher, auf den nur sie zugreifen kann. Ein globaler Speicher fehlt. Diese Tatsache hat erhebliche Auswirkungen auf die Software-Struktur. Ein Beispiel: Der Prozess in CPU A benötigt Daten von einem anderen Prozess in einer anderen CPU. CPU A muss also feststellen, welche CPU die benötigten Daten enthält, dieser eine Nachricht über das Vermittlungsnetzwerk zur Anforderung der Daten schicken und ihren eigenen Prozess bis zum Eintreffen der Antwort stoppen. Für die Leistung eines Mehrrechnersystems ist die Aufteilung und Bereitstellung der Daten an optimalen Stellen des Vermittlungsnetzes entscheidend. Die Programmierung eines Mehrrechnersystems ist somit erheblich schwieriger als die eines Mehrprozessorsystems. Mehrrechnersysteme besitzen aber Kostenvorteile gegenüber

Mehrprozessorsystemen, da sie aus preisgünstigen Standardbauteilen zusammengesetzt werden können. Die Konstruktion eines Mehrrechnersystems mit 10.000 CPU's und mehr ist kein größeres Problem, ein Mehrprozessorsystem mit einigen hundert CPU's schon.

Um die Stärken von Mehrprozessor- und Mehrrechnersystemen zu vereinen, werden Mischformen konstruiert. Der Ansatz bei diesen hybriden Formen ist, gemeinsam genutzten Speicher auf den verschiedenen Ebenen einzufügen (siehe Kapitel Addition und Subtraktion, oberhalb Ebene 2) und durch spezielle Laufzeitsysteme zu verwalten. Ein weiteres Ziel bei der Entwicklung dieser hybriden Formen ist die Skalierbarkeit, d.h. bei n eingesetzten CPU's sollte auch die n -fache Beschleunigung herauskommen und das System sollte um weitere CPU's erweiterbar sein. Diese lineare Beschleunigung wird aber nur von wenigen Programmen annähernd erreicht. Es existieren hervorragend parallelisierbare Algorithmen, bei denen das Maximum der Beschleunigung bei n CPU's liegt und die Beschleunigung bei Verwendung von mehr als n CPU's sogar wieder abfällt (Skyline-Matrix-Inversion). Die Begründung liegt darin, dass eben fast alle Programme auch sequentielle Anteile besitzen, z.B. Initialisierung, Einlesen der Daten

und Erfassung der Ergebnisse. Hier hilft die Verfügbarkeit beliebig vieler CPU's nicht weiter. Sei T die Ausführungszeit eines Programms auf einem Einprozessorsystem und ein Bruchteil f dieser Zeit der sequentielle Anteil, so wäre der Anteil $(1-f)$ der potentiell parallelisierbare Anteil. Kann dieser Anteil auf n CPU's ohne weitere Verzögerung (Overhead) ablaufen, lässt sich die Ausführungszeit bestenfalls von $(1-f) \cdot T$ auf $(1-f) \cdot T / n$ reduzieren. Damit ergibt sich die gesamte Ausführungszeit aus dem sequentiellen plus dem parallelen Teil, also $f \cdot T + (1-f) \cdot T / n$. Die Beschleunigung errechnet sich somit aus dem Quotienten der ursprünglichen Ausführungszeit T auf dem Einprozessorsystem und der neuen Ausführungszeit auf dem Mehrprozessorsystem. Man erhält somit die Beschleunigung zu $n / (1 + (n-1) \cdot f)$. Für $f=0$ ist die Beschleunigung linear. Für $f>0$ ist keine perfekte lineare Beschleunigung mehr möglich (**Amdahls Gesetz**).

7.11.1 SIMD-Architekturen

Nach einer Klassifikation von Flynn (1972) werden Rechnerarchitekturen durch zwei Unterscheidungsmerkmale eingeteilt:

- Wieviele Befehle werden gleichzeitig bearbeitet?
- Auf wievielen Speicherwörtern wird mit einem Befehl gleichzeitig operiert?

Diese Fragestellungen zielen im Wesentlichen darauf ab, wie viele universelle Prozessoren in einem Rechner parallel arbeiten. Die Betonung liegt auf 'universellen Prozessoren', nicht auf parallel arbeitenden Spezialprozessoren, wie z.B. Grafik- oder Ein- und Ausgabeprozessoren. Nach Flynn werden 4 Klassen unterschieden:

SISD (Single Instruction Stream - Single Data Stream)

Repräsentanten sind Rechner mit von-Neumann-Architektur; z.B. PC, Workstation

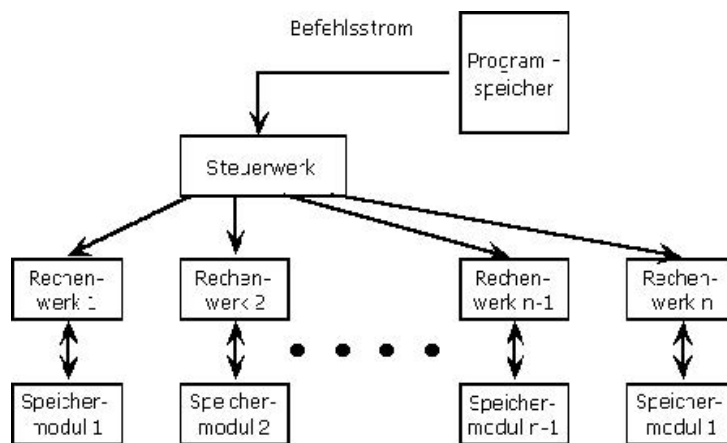
SIMD (Single Instruction Stream - Multiple Data Stream)

Repräsentanten sind Vektor- und Feld-Rechner; mit einem Befehl werden alle Komponenten eines Vektors oder alle Elemente einer Matrix manipuliert, z.B. Cray-Rechner (für komplexe, wissenschaftlich-technische Berechnungen)

MISD (Multiple Instruction Stream - Single Data Stream) Keine Repräsentanten

MIMD (Multiple Instruction Stream - Multiple Data Stream, siehe Kapitel MIMD-Architekturen)

Hier gibt es sehr verschiedenartige, aber immer sehr spezielle Multiprozessorarchitekturen. Hinzuzuzählen sind auch eng gekoppelte SISD-Architekturen. Der Übergang zu Rechnerverbünden wird dann fließend.



Informationsfluss im SIMD-Rechner

Der SIMD-Rechner besitzt ein Steuerwerk zur Befehlsdekodierung und mehrere Rechenwerke mit bidirektionalem Zugriff auf den Hauptspeicher. Die Maschinsprache von SIMD-Rechnern muss die Verteilung der Operanden im Hauptspeicher berücksichtigen und auch deren Kommunikation steuern. Die klassischen SIMD-Rechner sind so genannte Vektor- oder Feld-Rechner. Wenn jedes

Rechenwerk eine Breite von nur einem Bit besitzt, dann spricht man von einem Assoziativrechner. Assoziativrechner werden z. B. zur Vorverarbeitung von Bilddaten verwendet. Rechner mit SIMD-Struktur eignen sich besonders zur Abarbeitung von Schleifen, in denen sequentiell Vektorelemente verarbeitet werden. Beispiel:

```
for I := 1 to N do
```

```
  C[I] := A[I] + B[I];
```

Im Rechenwerk 1 würde z.B. die Zuweisung $C[1] := A[1] + B[1]$, im Rechenwerk 2 die Zuweisung $C[2] := A[2] + B[2]$; u.s.w. bearbeitet werden.

Keine Leistungssteigerung erreicht man dagegen bei Schleifen, in denen Datenabhängigkeiten vorkommen. Beispiel:

```
for I := 2 to N do
```

```
  C[I] := C[I-1] + B[I];
```

Moderne Compiler führen die Transformation geeigneter Schleifen in die entsprechenden Vektorinstruktionen selbstständig durch oder nehmen sogar Änderungen am Programmcode vor, um besser vektorisieren zu können. Beispiel:

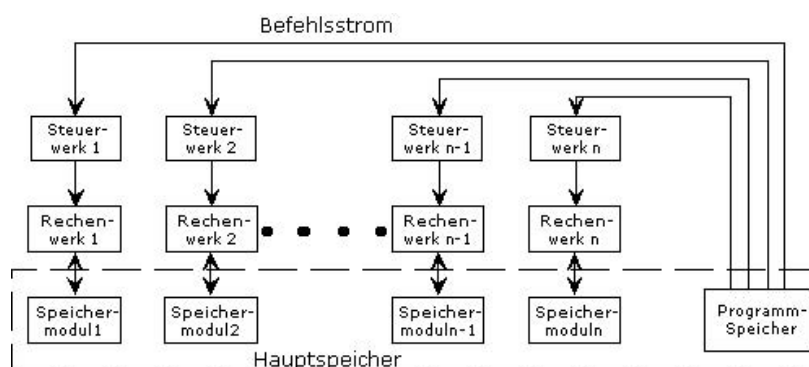
```
for I := 1 to N do
begin
  T := A[I] + B[I];
  if T ≥ 0 then C[I] := C[I-1] . T
end;
```

Die Umwandlung der skalaren Variablen T in ein Array T[] würde eine Vektorisierung der Schleife ermöglichen.

Rechner mit SIMD-Architektur (z.B. CRAY) mit ihren maximal 16 Prozessoren reichen für die Bewältigung von besonders rechenintensive Aufgaben nicht aus. Die hierfür notwendige Leistung kann nur mit Architekturen, die viele tausend Prozessoren umfassen, erreicht werden.

7.11.2 MIMD-Architekturen

Die Vorteile bei der Nutzung von MIMD-Parallelrechnern liegen in der Reduzierung der Laufzeit eines Prozesses und in der Ausfallsicherheit. Zum einen liegen dadurch die Ergebnisse eher vor, zum anderen werden Betriebsmittel, z.B. der Hauptspeicher, schneller freigegeben. Ein anderer Aspekt liegt in der Erhöhung des Durchsatzes. Auf jedem Prozessor arbeitet unabhängig ein Prozess. Im gleichen Zeitintervall können somit mehr Aufträge abgearbeitet werden. Ein Steuerwerk, das mindestens ein Rechenwerk steuert, fasst man unter dem Begriff "Prozessor" zusammen.





Informationsfluss im MIMD-Rechner

Es existieren unterschiedliche Typen von MIMD-Rechnern, deren Eigenarten durch das Flynn- Schema nicht erfasst werden. Die verschiedenen Typen von MIMD-Rechnern unterscheiden sich hinsichtlich der Organisation des Speichers und der Art der Kopplung der Prozessoren an den Speicher und untereinander. Verwenden die Prozessoren einen gemeinsamen Speicher, nennt man diese Systeme **eng gekoppelt**. Der Informationsaustausch erfolgt über Datenstrukturen im Speicher. Allerdings können diese Systeme nicht beliebig um Prozessoren erweitert werden, da allen Prozessoren ein gleich schneller Zugriff auf den gesamten Speicher ermöglicht werden muss. Ein möglicher 64-facher überlappender Zugriff mit 64 Speicherbänken setzt ein theoretisches Limit von 64 Prozessoren.

Verfügt jeder Prozessor über eigenen Speicher, nennt man diese Systeme **lose gekoppelt**. Der Nachrichtenaustausch unter den Prozessoren erfolgt dann über ein Vermittlungsnetzwerk. Derartige Systeme sind beliebig erweiterbar. Rechner mit 65536 Prozessoren (2^{16}) sind schon gebaut worden. Allerdings muss bei diesen Strukturen in Kauf genommen werden, dass die Kommunikation zwischen zwei Prozessoren über andere Prozessoren als Zwischenstationen erfolgen muss, was natürlich Zeit in Anspruch nimmt. Werden viele Prozessoren in einem System verwendet, spricht man von einem **Massiv-Parallel-System (MPR)**. Derartige Systeme können kostengünstig aus Standard-Prozessoren aufgebaut werden.

Bei der Softwareerstellung für derartige Systeme ist die Aufgabe in mehrere unabhängig voneinander ausführbare Teilaufgaben, so genannte Threads (**Threads = Fäden**), zu zerlegen, die dann auf unterschiedlichen Prozessoren und nach Möglichkeit in unterschiedlicher Reihenfolge ausgeführt werden können. Diese Threads sollten aber nicht zu klein sein, man spricht von Grobkörnigkeit, damit die Threadwechsel im Vergleich zur Lösung der Gesamtaufgabe nicht zuviel Zeit beanspruchen. Bei der Softwareerstellung wird nach dem Prinzip "teile und herrsche" vorgegangen, womit die sukzessive Aufteilung in voneinander unabhängige Teilaufgaben mit annähernd gleicher Komplexität (**load balancing**) und anschließender Zusammenfassung der Ergebnisse gemeint ist. Beispiel 1:

$$\text{SUMME} := a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7$$

Für die Berechnung von SUMME benötigt ein sequentiell arbeitender von-Neumann-Rechner 7 Schritte.

Parallelisiert wird daraus:

$SUMME_1 = a_1 + a_2 + a_3$ $SUMME_2 = a_4 + a_5 + a_6 + a_7$ Danach wird die Gesamtsumme gebildet. Führt man die Berechnung von $SUMME_1$ und $SUMME_2$ auf zwei Prozessoren aus, so sind nur 4 Schritte notwendig.

Beispiel 2:

Quick und Heap-Sort Algorithmus. Ein sequentieller Rechner benötigt $n \cdot \log(n)$ Arbeitsschritte für die Sortierung. Schaltet man n Prozessoren parallel, benötigt der Parallelrechner n Schritte. Jeder der Prozessoren kennt einen aktiven und einen passiven Zustand. Im aktiven Zustand wird der Wert vom rechten Nachbarn empfangen und mit dem eigenen gespeicherten Wert verglichen. Der größere Wert wird an den rechten Nachbarn zurückgesandt und der kleinere gespeichert. Im passiven Zustand wird der gespeicherte Wert an den linken Nachbarn gesandt und der zurückgelieferte Wert gespeichert. Nach jedem Verarbeitungsschritt wechselt der Zustand. Zu Beginn befinden sich alle Prozessoren mit ungeraden Nummern im aktiven Zustand.

Prozessor		1	2	3	4	5	6
Wert		5	2	6	1	9	3

Im ersten Schritt empfängt Prozessor 1 den Wert 2 von seinem rechten Nachbarn. Nach Vergleich sendet er den Wert 5 zurück.

1. Schritt		2	5	1	6	3	9
2. Schritt		2	1	5	3	6	9
3. Schritt		1	2	3	5	6	9

Das Ergebnis liegt also in maximal n Schritten vor.

Die Gesamtleistung der Parallelrechner ist abhängig von:

- Der Leistungsfähigkeit der einzelnen Prozessoren
- Der Anzahl der Prozessoren
- Der Kommunikation der Prozessoren untereinander (Netztopologie!)
- Der Parallelisierbarkeit des Algorithmus

7.12 Aufgaben zum Thema Rechnerhardware

In diesem Unterkapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt. Es empfiehlt sich, die Aufgaben selbstständig auf einem Blatt Papier zu lösen. Musterlösungen werden während des Semesters nach und nach freigeschaltet. Aus ähnlichen Aufgaben kann sich die Klausur zusammensetzen.



Aufgabe

Aufgabe 7.1

Gegeben ist das folgende Assembler-Programm zur Addition zweier Zahlen:

```
LDA 100 ; lade aus Speicherzelle 100 in Register A
ADA 101 ; addiere Speicherzelle 101 zu Register A
STA 102 ; speichere Inhalt von Register A in Speicherzelle 102
```

Beschreiben Sie die Aktionen des Steuerwerks zur Abarbeitung dieses Programms.



Aufgabe

Aufgabe 7.2

Erläutern Sie, warum Ein-/Ausgaben die über den "vergleichsweise langsamen" PCI-Bus oder sogar USB-Bus abgewickelt werden, die Anwendungen eines PC's nicht wesentlich verlangsamen.



Aufgabe

Aufgabe 7.3

Erläutern Sie den Übergang vom Assemblerprogramm zum Maschinenbefehlsablauf bei RISC und CISC Architekturen.



Aufgabe

Aufgabe 7.4

Zeichnen Sie das Pipelining für die folgenden Befehle auf:

ADD A, B; Ergebnis in A

ADD C, D; Ergebnis in C



Aufgabe

Aufgabe 7.5

Ermitteln Sie überschlägig die wesentlichen Datenraten im Signalgang von einer Videoschnittkarte mit Komprimierung (1:4) bis auf die Festplatte



Aufgabe

Aufgabe 7.6

Schlagen Sie eine Aufgabenverteilung für ein Parallelrechensystem mit 8 gleichartigen und gleichberechtigten Rechnern vor, das einen Film mit (25 Frames (Bildern)/s) von 12s Dauer erstellen soll. Welche Teilaufgaben gibt es? Wie sollen diese auf die Rechner verteilt werden?

7.13 Fragen mit Musterlösungen: Rechnerhardware

In diesem Kapitel sind Verständnisfragen mit ihren Musterantworten zu dem vorangehenden Lehrstoff zusammengestellt. Solche Fragen werden auch in der Klausur vorkommen.

Im Folgenden sind die Verweise zu den Fragen aufgelistet. Mit einem Klick auf 'Antwort' am Ende der Fragenseite bekommt man die Antwort für das gestellte Problem angezeigt. Ein Klick auf "Info-Basis ..." führt zu dem Kapitel des für die Aufgabe relevanten Lehrstoffes.

Das Textfeld dient z.Z. lediglich dazu, eigene Antworten zu notieren, um sie besser mit der Musterantwort vergleichen zu können. Eine Speichermöglichkeit des Textes ist damit jedoch nicht gegeben.

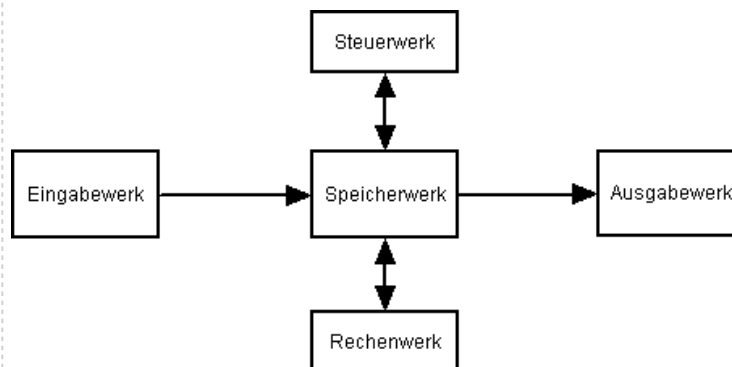


Aufgabe

Frage 7.1

Beschreiben Sie den Aufbau eines von Neumann-Rechners!

Antwort



Vgl. Kap. Begriffserklärung



Aufgabe

Frage 7.2

Woraus ist eine Zentraleinheit aufgebaut?

Antwort

Die Zentraleinheit besteht nach dem von Neumann-Rechnermodell aus Steuerwerk und Rechenwerk.

Vgl. Kap. Begriffserklärung



Aufgabe

Frage 7.3

Was wird unter einem Befehlszyklus verstanden?

Antwort

Ein Befehlszyklus des Steuerwerks besteht aus drei Schritten:

1. Hole den Befehl, dessen Adresse im Befehlszähler steht, in das Befehlsregister.
2. Schreibe die Adresse des nächsten Befehls in den Befehlszähler.
3. Führe den im Befehlsregister stehenden Befehl aus.

Vgl. Kap. Begriffserklärung



Aufgabe

Frage 7.4

Wie ist ein Hauptspeicher organisiert?

Antwort

Ein Hauptspeicher besteht aus binären Speicherzellen, die mit einer festen Anzahl von Bits zu Speicherwörtern strukturiert sind. Die Speicherwörter sind in der Regel die kleinsten adressierbaren Einheiten des Speichers. Typische Wortlängen (**k**) sind **k=8**, **k=16**, **k=32**, **k=64**. Die Adressierung der Speicherwörter erfolgt von Null aufwärts.

Vgl. Kap. Begriffserklärung



Aufgabe

Frage 7.5

Was wird unter Pipeline-Verarbeitung verstanden?

Antwort

Beim Pipelining werden die aufeinanderfolgenden Phasen des Befehlszyklus, zeitlich versetzt, parallel bearbeitet.

Vgl. Kap. Das Schichtenmodell eines Rechnersystems

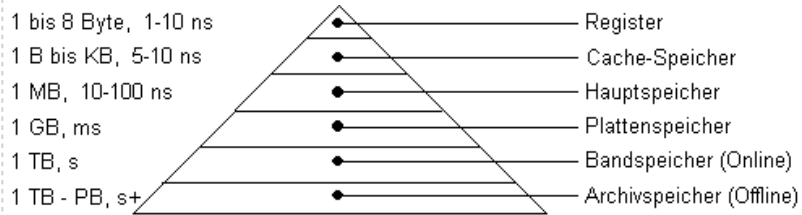


Aufgabe

Frage 7.6

Nennen Sie interne und externe Speicher und ordnen Sie diese nach Kapazität und Zugriffszeit!

Antwort



Vgl. Kap. Die Struktur der von Neumann-Maschine

8 System- und Anwendungssoftware



Gliederung

8 System- und Anwendungssoftware

8.1 Betriebssysteme

8.2 Basissysteme

8.3 Anwendungssysteme

8.4 Aufgaben zum Thema: System- und Anwendungssoftware

8.5 Fragen mit Musterlösungen: System- und Anwendungssoftware

8.1 Betriebssysteme

Das **Betriebssystem (BS)** wird durch die Programme eines digitalen Rechensystems gebildet, die zusammen mit den Eigenschaften der Rechanlage die Basis der möglichen Betriebsarten des digitalen Rechensystems und insbesondere die Abwicklung von Programmen steuern und überwachen.

Die Programmabarbeitung erfolgt dabei im Wesentlichen durch das Zusammenspiel von diversen Verwaltungsprogrammen. Die wichtigsten davon sind:

- Prozessorverwaltung
- Speicherverwaltung
- Dateiverwaltung
- Geräteverwaltung
- Auftragsverwaltung

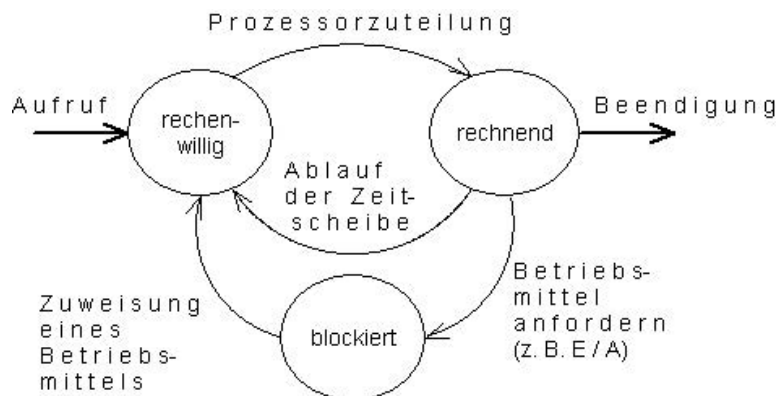
Prozess- und Prozessorverwaltung

Aufgabe der Prozess- und Prozessorverwaltung ist die Einteilung der Befehlsabläufe in Prozesse, die Verwaltung von Prozesszuständen und die Zuordnung von Rechenprozessoren zu rechenwilligen Prozessen.

Ein **Prozess** ist dabei eine Folge von sequentiell abzuarbeitenden Aktionen. Dies kann ein gesamtes zur Ausführung gelangtes Programm sein oder auch einzelne Sequenzen davon. Stehen mehrere Prozessoren zur Verfügung, so können mehrere Prozesse parallel ausgeführt werden. Bei nur einem Prozessor können mehrere Prozesse nur 'quasi parallel' (im Multiplex- Betrieb) ausgeführt werden. Das heißt, dass ein Prozess an einer Stelle unterbrochen und ein anderer vom System wieder aufgenommen wird. Der

unterbrochene Prozess wird später weiterbearbeitet. Ein Prozess kann nicht an einer beliebigen Stelle unterbrochen werden.

In untenstehender Abbildung sind die Übergänge der **Prozesszustände** eines Prozesses graphisch dargestellt. Ein Prozess ist entweder rechnend, oder er wartet. Er wartet entweder, weil der Prozessor belegt ist, oder weil er auf Grund eines benötigten, aber noch nicht zugeteilten, Betriebsmittels blockiert ist.



Prozesszustände

Die Prozessorzuteilung an rechenwillige Prozesse kann nach unterschiedlichen Strategien erfolgen:

- Reihenfolgeverfahren: Die Prozesse werden in der Reihenfolge, in der sie rechenwillig werden, abgearbeitet.
- Prioritätsverfahren: Den Prozessen werden Prioritäten zugeordnet, wobei derjenige mit der

höchsten Priorität jeweils die Zuteilung bekommt.

- Zeitscheibenverfahren: Den Prozessen wird in regelmäßigen Abständen der Prozessor zugeteilt. Eine neue Zuteilung erfolgt, wenn die Zeitscheibe des rechnenden Prozesses abgelaufen ist oder der rechnende Prozess auf ein Ereignis warten muss (z.B. Freigabe eines

Betriebsmittels wie Ausgabe oder Eingabe).

Neben diesen einfachen Zuteilungsstrategien gibt es weitere, insbesondere auch Mischformen. Es wird von **parallelen Prozessen** gesprochen, wenn mehrere (in sich sequentielle) Prozesse parallel abgearbeitet werden oder werden könnten. Dabei ist es unerheblich, ob dies wirklich parallel auf mehreren Prozessoren oder quasi-parallel auf einem Prozessor erfolgt. Bei voneinander unabhängigen Prozessen sind keine Randbedingungen zu beachten. Bei **gekoppelten Prozessen** gibt es Abhängigkeiten,

die zu berücksichtigen sind. Erzeugt ein Prozess z.B. Daten, die von einem anderen Prozess verarbeitet werden sollen, so muss der empfangende Prozess auf die Daten warten, bevor er weiter abgearbeitet werden kann. Die Kopplung in parallelen Prozessen beschränkt sich meist auf kurze Befehlssequenzen, die übrigen können parallel ablaufen.

Unabhängig von solchen Prozesskopplungen gibt es Sequenzen, die lediglich nicht parallel bzw. quasi-parallel von mehreren Prozessen durchlaufen werden dürfen, die sogenannten kritischen Abschnitte. Solche Abschnitte schließen sich wechselseitig aus (mutual exclusion). Das Betreten von **kritischen Abschnitten** wird über **Semaphore** gesteuert, es wird nur maximal einem Prozess das Betreten gestattet. Ein Semaphor ist dabei eine besondere Markierung, so eine Art "Bitte- Nicht-Stören-Schild" bei der Abarbeitung schutzwürdiger Befehlssequenzen. Dabei muss sichergestellt sein, dass das Setzen und Löschen des Semaphors selbst nicht gestört werden kann.

Eine weitere Art von Kopplung sollte möglichst nie eintreten, lässt sich aber leider bei dynamischen Abläufen nicht immer einfach erkennen: Zwei Prozesse warten gleichzeitig auf vom jeweils anderen zu liefernde Daten oder zu erfüllende Bedingungen. Eine solche Situation heißt Verklemmung (**Dead-Lock**).

Speicherverwaltung

Aufgabe der Speicherverwaltung ist die Zuteilung von Adressräumen (Speicherbereichen) im Hauptspeicher an die Prozesse. Kommt ein Prozess zur Ausführung, so wird ihm ein Adressraum zugeordnet. Die Freigabe erfolgt spätestens bei Ende des Prozesses. Wird der Prozess unterbrochen, so kann der Adressraum u.U. bis zur Wiederaufnahme zugeordnet bleiben. Falls jedoch für den dann auszuführenden Prozess nicht genügend freier Speicher zur Verfügung steht, muss dem unterbrochenen Prozess der Adressraum entzogen werden. Der Inhalt dieses Speicherbereichs wird auf einen externen Speicher ausgelagert und bei Wiederaufnahme des Prozesses in einem neu zuzuordnenden Adressraum wieder eingelagert.

Werden Prozessen die Adressräume dynamisch erst zur Laufzeit zugewiesen, so können zur Übersetzungs-/Bindezeit der Programme nicht die realen Speicheradressen generiert werden. Es werden virtuelle Adressen erzeugt. Zur Prozessausführung muß dann eine Adressumsetzung erfolgen.

Programmadressen:

Adressen, die im Prozessor erzeugt und bei der Ausführung eines Maschinenbefehls zur Adressierung von Operanden und Befehlen benutzt werden.

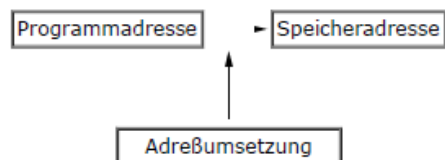
Speicheradressen:

Adressen, die beim (Hardware-) Zugriff auf den Hauptspeicher die angesprochenen Speicherstellen identifizieren.

Reale Adressierung:

Programmadresse = Speicheradresse

Virtuelle Adressierung:



Ein-/Ausgabesystem

Das Ein- /Ausgabesystem läßt sich untergliedern in die Datei- und die Geräteverwaltung.

Die Dateiverwaltung (Dateisystem, File-System) organisiert den Speicherplatz auf den externen Speichermedien in Dateien und hierarchisch angelegten Verzeichnissen (Directories). Dateien und Verzeichnisse können angelegt, gelesen, geändert und gelöscht werden. Die logische Struktur des Dateisystems muss auf die physikalische Struktur des jeweiligen Speichermediums abgebildet werden. Über diese erfolgt dann der eigentliche Zugriff auf die Daten.

Die Geräteverwaltung steuert und überwacht die Ein- und Ausgabe von und an die Ein- und Ausgabegeräte. Das heißt, von und an die jeweiligen Ein- und Ausgabeschnittstellen, einschließlich der Schnittstellen der externen Speichermedien. Auch hier wird die logische Struktur der Geräte auf die physikalische abgebildet. Der eigentliche Transfer der Steuersignale und der Daten wird von Gerätetreibern (Device-Drivers) gesteuert.

In der folgende Tabelle sind die Komponenten eines Ein- und Ausgabe-Systems schematisch dargestellt.

	E/A - Vorgänge	Verwaltungsrouinen	
Logische	Stufe	Logische Zugriffsmethoden (diverse)	Datei- und trägerverwaltung Daten-
	↕		

Physikalische Stufe	Physikalische Zugriffsmethode	Extern-speicher-verwaltung	Geräte-verwaltung
	↕		
	Gerätetreiber		
	↕		
	Hardware		



Komponenten eines Ein- und Ausgabe-Systems

Auftragsverwaltung

Auf einem Rechner laufen i.d. Regel mehrere Programme mehrerer Benutzer. Diese organisieren Ihre Rechenaufträge in Teilaufträge, z.B.

- übersetze Programm
- lade Programm
- starte Programm

Solche Aufträge sind vom Betriebssystem zu verwalten und abzuarbeiten. Die Benutzer beschreiben Ihre Aufträge in einer Steuersprache (Job-Control-Language).

Client - Server Architektur

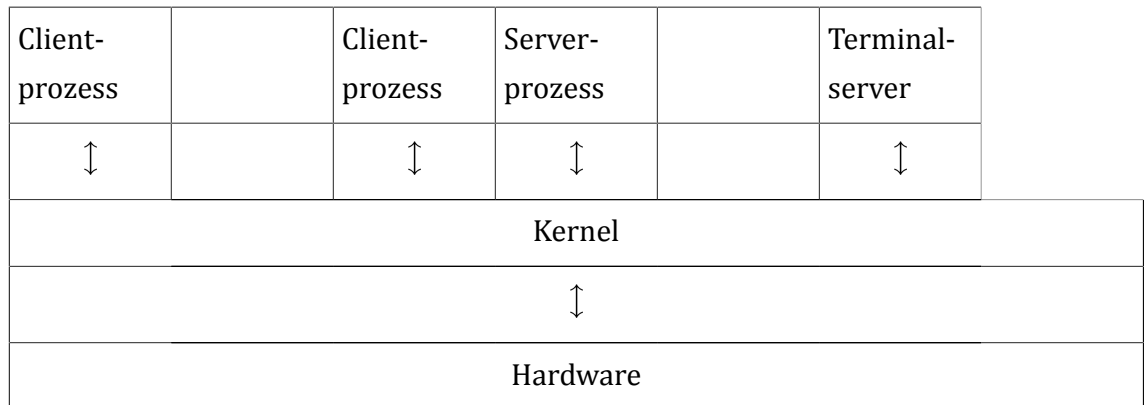
Es gibt diverse Architekturen von Betriebssystemen. Dies ist einmal bedingt durch die unterschiedliche Entwicklung von Rechenanlagen in der Vergangenheit, andererseits gibt es aber auch heute noch verschiedene Anforderungen aus Sicht der zugrunde liegenden Hardware und der Anwendungssysteme.

In einer **Client-Server-Architektur** werden Kunden- und Anbieterprozesse (Clients und Server) um ein Betriebssystemkern (Kernel) angeordnet. Sie kommunizieren über Nachrichten miteinander. Der Betriebssystemkern verteilt diese Nachrichten und stellt die Schnittstelle zur Hardware dar (untenstehende Abbildung).

Es wird zwischen 'Kernel-Modus' und 'User-Modus' unterschieden. Der **Kernel-Modus** ist der privilegierte Modus. Er darf alles, wird als fehlerfrei vorausgesetzt und beinhaltet den Betriebssystemkern. Der **User-Modus** ist der nichtprivilegierte Modus. Er hat nur eingeschränkte Befugnisse. Hier sind die Server und Clients angesiedelt. Auch Teile des Betriebssystems sind nur Server-Prozesse, haben also nur eingeschränkte Befugnisse.

Das System ist dadurch sicherer und flexibler. Ein Beispiel für ein Client-Server-System ist Windows NT.

Die Kommunikation zwischen Client und Server beschränkt sich nicht auf einen Rechner. Client und Server können sich auf verschiedenen Rechnern befinden, welche durch ein Netzwerk verbunden sind (**Verteilte Systeme**). Dies funktioniert auch über verschiedene Hardware- Plattformen hinweg.



Client-Server-Architektur

8.2 Basissysteme

Zu den grundlegenden Programmen eines Rechners gehören, neben dem im vorangegangenen Kapitel beschriebenen Betriebssystem, die folgenden Basissysteme:

- Editoren
- Assembler, Compiler, Binder
- Büroanwendungen
- Datenbanksysteme

Editoren

Editoren sind Programme zur Erstellung und Bearbeitung von beliebigen Texten über Tastatur und Bildschirm. Sie werden insbesondere zur Erstellung von Programmen eingesetzt. Es gibt Editoren, die eine Eingabe von bestimmten Textarten besonders unterstützen.

Assembler, Compiler und Binder

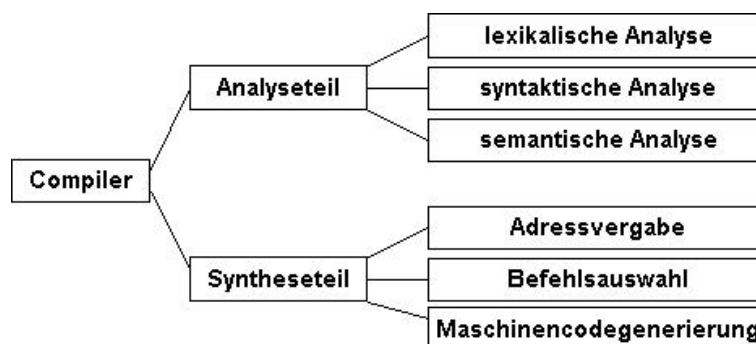
Assembler, Compiler und Binder sind Programme zum Übersetzen von Programmiersprachen in ein vom Rechner ausführbares Programm. Sie werden zusammen mit einem Editor zur Programmentwicklung benötigt.

Ein **Assembler** ist ein Programm, welches ein anderes Programm in so genannten Assemblercode (siehe Beispiel im Kapitel Einführung) in den Maschinencode des Rechners übersetzt. Ein Assemblerprogramm ist hardwareorientiert (hardwareabhängig). Das bedeutet, dass es nur in einer bestimmten Hardwareumgebung funktioniert.

Ein **Compiler** ist ein Programm, welches ein anderes Programm, das in einer problemorientierten, meist hardwareunabhängigen Programmiersprache geschrieben ist, in Maschinencode des Rechners übersetzt, der diesen Code ausführen soll. Dieser Maschinencode ist allerdings noch nicht ausführbar. Nach der Übersetzung muss das Programm noch geladen (gebunden) werden. Man spricht auch von Compilern, wenn ein Programm nicht in Maschinencode sondern in eine andere Programmiersprache oder einen von einer virtuellen Maschine interpretierbaren Bytecode (wie z.B. bei der Programmiersprache JAVA) übersetzt wird.

Ein **Binder** (Linker) ist zuständig für die Endbearbeitung eines von einem Compiler bzw. Assembler übersetzten Programms. Ein Programm besteht in der Regel aus mehreren Prozeduren in verschiedenen Dateien, die unabhängig voneinander übersetzt werden. Diese werden beim Binden zu einem Maschinenprogramm zusammengeführt. Dazubinden sind diverse Systemroutinen, z. B. bezüglich der Ein- und Ausgabe. Den jeweiligen Programmteilen müssen die Speicheradressen der anderen Programmteile bekannt gemacht werden. Erst dann ist ein Programm ausführbar und kann in den Hauptspeicher geladen und von dort gestartet werden.

Im Folgenden erfolgt ein kurzer Überblick über die Struktur von **Compilern** (untenstehende Abbildung)



Struktur eines Compilers

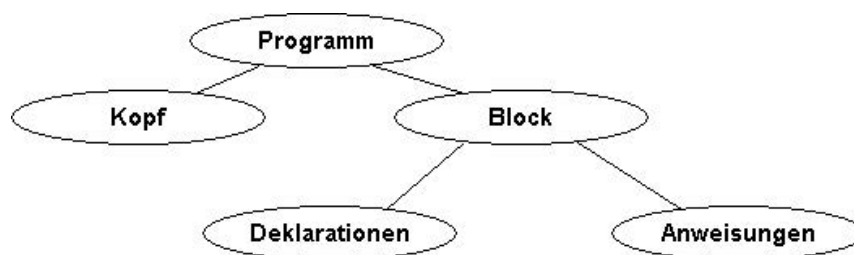
Lexikalische Analyse:

Erkennung der zulässigen Wörter einer Programmiersprache. Folgende Sprachsymbole werden erkannt:

- Bezeichner
- Zahlen
- Zeichenketten
- Spezialsymbole: Schlüsselwörter, Sonderzeichen, Operatoren

Syntaxanalyse:

Untersuchung, ob die Sprachsymbole in einer zulässigen Struktur (Reihenfolge) vorliegen. Die Programmstruktur lässt sich grafisch in einem Syntaxbaum darstellen (untenstehende Abbildung).

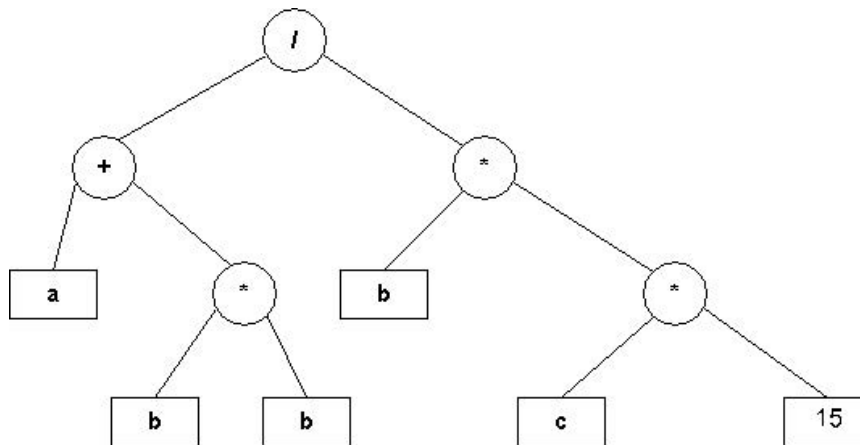


Programmstruktur

Semantische Analyse:

In dieser Phase werden alle die Analysen vorgenommen, die in den vorangegangenen Phasen nicht möglich waren. Dies sind unter anderem:

- Verwaltung der deklarierten Bezeichner in einer Symboltabelle
- Erstellen von Ausdrucksbäumen und Typüberprüfung der Bezeichner (untenstehende Abbildung)

**Ausdrucksbaum**Bezeichnen: a, b und c für den Ausdruck $(a + b \cdot b) / (b \cdot c \cdot 15)$ **Synthese:**

Auf Basis der vorangegangenen Analyse ist der Maschinencode zu generieren. Dazu gehört neben der Auswahl und Anordnung der Maschinenbefehle insbesondere auch die Organisation des Arbeitsspeichers für die Variablen und die Befehle. Bezogen auf den Hauptspeicher werden ausschließlich relative Adressen vergeben.

Büroanwendungen

Hierunter werden Programme subsumiert, die an einem Büroarbeitsplatz zur Standardausstattung gehören, wie z. B. Textverarbeitung, Tabellenkalkulation, Taschenrechner, Terminkalender, Präsentationsgraphik und Programme zur Kommunikation und Information (E-Mail, Browser).

Datenbanksysteme

In vielen Anwendungen (Programmen) werden große Datenbestände dauerhaft auf externen Speichermedien verwaltet und bearbeitet. Stellen solche Datenbestände eine logische Einheit dar, so wird von einer **Datenbank** gesprochen. Meist arbeiten mehrere Benutzer gleichzeitig mit solchen Datenbanken. Ein Datenbanksystem unterstützt die Erstellung und das Arbeiten mit Datenbanken.

Ein **Datenbanksystem** (DBS) besteht aus...

...einer (ggf. mehreren) Datenbank(en):

Integrierter, einheitlich strukturierter Datenbestand.

Die Struktur (Schema) wird über eine Datendefinitionssprache (DDL) definiert.

...einem Datenbankverwaltungssystem (DBVS, auch Datenbankmanagementsystem DBMS):

Software zur Verwaltung von Datenbeständen.

Die Zugriffe und Manipulationen in der Datenbank erfolgen über eine :Abfragesprache (Datenmanipulationssprache (DML)).

Charakteristika:

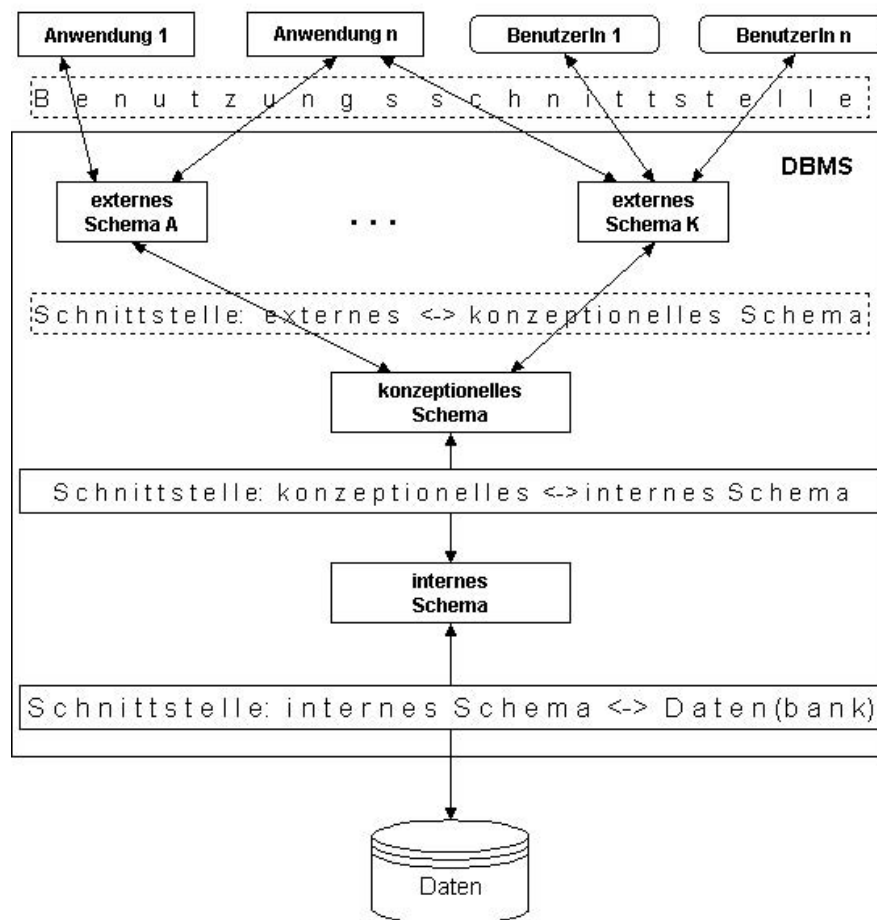
- Jeder Benutzer kann auf dem Datenbestand ohne Kenntnis der internen Struktur zugreifen.
- Jeder Benutzer kann nur kontrolliert auf den Datenbestand zugreifen. Konsistenz und Sicherheit der Daten sind gewährleistet.
- Die interne Struktur der Daten kann ohne Störung der Benutzer geändert werden.

Aufgaben:

- Integration: Einheitliche Verwaltung aller Daten
- Operationen: Datenspeicherung, -suche, -änderung und -löschung
- Katalog: Integrierte Beschreibung der Daten bzw. der Metadaten (Data Dictionary)
- Benutzersichten: verschiedene Benutzerklassen können, bzw. dürfen, die Daten unterschiedlich sehen
- Konsistenz- Erhaltung: Gewährleistung korrekter Datenbankinhalte
- Datenschutz: Ausschluß nicht berechtigter Zugriffe
- Transaktionen: Zusammenfassung von Datenbankzugriffen zu 'atomaren' (zusammengehörigen) Einheiten
- Synchronisation: Verwalten von gleichzeitigen/konkurrierenden Transaktionen mehrerer Benutzer
- Datensicherung: Wiederherstellung von Daten nach Systemfehlern

Zur Unterstützung der Charakteristika und der Aufgaben eines Datenbanksystems werden die Datenbestände logisch in drei Schichten strukturiert. Jede Schicht wird durch ein Schema definiert, welches eine bestimmte Sicht auf die Daten liefert. Es gibt das interne Schema (die physikalische Struktur der Daten auf dem externen Speicher liegt noch darunter), das konzeptionelle Schema und das externe Schema. Das externe Schema kann mehrfach vorkommen. Es definiert die Sichtweisen für die diversen Anwendungen (untenstehende Abbildung).

3-Ebenen-Schemaarchitektur für Datenbanken



3-Ebenen-Schemaarchitektur für Datenbanken

Es gibt verschiedene Datenbankmodelle. Das zur Zeit wichtigste ist das so genannte Relationenmodell (Codd, 1970). Es besteht aus :

- Relation: anschaulich als Tabelle,

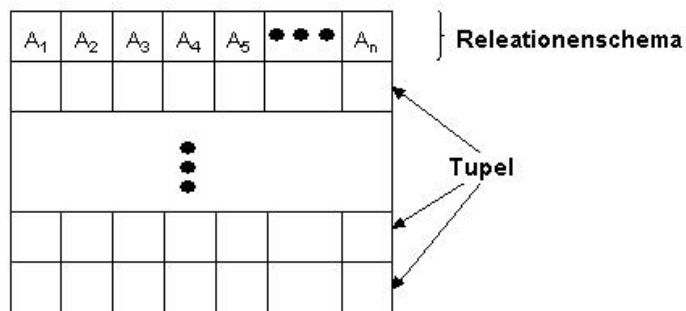
jede Relation hat einen Namen

- Relationenschema: Beschreibung der Relationen, die aus mehreren

Attributen bestehen ($A_1, A_2, A_3, \dots, A_n$)

- Tupel: jede Eintragszeile der Tabelle (Datensatz)
- Datenbank: Zusammenfassung mehrerer Relationen In untenstehender Abbildung ist eine Relation graphisch dargestellt.

Relationenname



Relation des Relationenmodells

Ein Attribut oder eine Attributkombination, dessen Wert bzw. deren Werte die Tupel (Datensätze) in einer Relation eindeutig identifiziert, heißt **Schlüssel**. Es wird/Sie werden zur Kennzeichnung unterstrichen.



Beispiel

Miniwelt "Fachhochschule"

- Informationen über:
 - Studierende
 - Matrikelnummer
 - Name, Vorname
 - Geburtsdatum
 - Studiengang
 - Semester
 - ...
- Lehrende
 - Personalnummer
 - Name, Vorname
 - Titel
 - Fachbereich
 - Lehrgebiet
 - ...
 - Lehrveranstaltungen
 - Lehrveranstaltungsnummer
 - Bezeichnung

- Studiengang
 - Semester
 - ...
- Beziehungen zwischen den "Objekten"
 - Lehrende bieten Lehrveranstaltungen an
 - Studierende belegen Lehrveranstaltungen
 - ...

Studierende						
MATRNR	NAME	VORNAME	GEBURTSDATUM	STUDIENGANG	SEMESTER	...
96407	Meier	Hans	1981	PI	1	...
96373	Schulz	Klaus	1982	PI	2	...
96609	Meyer	Arne	1983	PI	3	...
...						
...						
...						

Lehrende						
PERSONALNUMMER	NAMEN	VORNAME	TITEL	FACHBEREICH	LEHRGEBIET	...
123456007						
...						

Lehrveranstaltungen						
LEHRVERANSTALTUNGSNUMMER	STUDIENGANG	SEMESTER	...			
1234						
...						

Die Realisierung oben genannter Beziehungen, z. B. Lehrende bieten Lehrveranstaltungen an, erfolgt in den Relationentabellen folgendermaßen:

bietet an	
PERSONALNUMMER	LEHRVERANSTALTUNGSNUMMER

123456007	1234
...	

Die Zuordnung der Tupel erfolgt also über eindeutige und identische Werte von Attributen.

8.3 Anwendungssysteme

Anwendungssysteme sind Programme oder Programmsysteme, die bestimmte Aufgaben bearbeiten bzw. bestimmte Probleme lösen (Anwendung, Applikation). Eine Grobeinteilung solcher Systeme kann wie folgt vorgenommen werden:

Technisch-wissenschaftliche Systeme:

Computer Aided Design (CAD) Computer Aided Manufacturing (CAM) Finite Elemente Methode (FEM) ...

Kaufmännisch-administrative Systeme:

Produktionsplanung und Steuerung (PPS) Finanzbuchhaltung Personalverwaltung und Gehalt ... Insbesondere bei den Systemen der 2. Gruppe handelt es sich primär um Datenbankanwendungen. Eine andere Art der Einteilung von Anwendungssystemen lässt sich anhand der verwendeten Datenstrukturen und Algorithmen vornehmen. Hierbei werden konventionelle oder so genannte 'intelligente' Modelle und Methoden der Informatik verwendet. Im ersten Fall wird von konventioneller Datenverarbeitung, im zweiten Fall von Wissensverarbeitung (Expertensystemen, intelligenten Systemen) gesprochen. In der Tabelle werden diese einander gegenübergestellt.

Datenverarbeitung	Wissensverarbeitung
monotone, klar strukturierte und wohldefinierte Datenverarbeitungsprozesse	komplexe Informations- und Wissensverarbeitungsprozesse
algorithmischer Verarbeitungsablauf	heuristischer Verarbeitungsablauf
Verarbeitungsablauf ist explizit festgelegt	Verarbeitungsablauf ist nur implizit oder gar nicht vorgegeben
Verarbeitung von Zahlen und Zeichen	Verarbeitung symbolischer Ausdrücke

wenige Datentypen, aber viele Instanzen eines Typs	viele Datentypen, oft wenige Instanzen eines Typs
Verarbeitung homogen strukturierter Massendaten	Verarbeitung heterogen strukturierten Wissens
unvollständige Eingaben werden zurückgewiesen	Verarbeitung unvollständiger Strukturen ist möglich
System kann den Verarbeitungsprozess nicht erklären oder rechtfertigen	System kann den Verarbeitungsprozess erklären und rechtfertigen
System ist nicht lernfähig	System ist lernfähig



Datenverarbeitung vs. Wissensverarbeitung

Die Verschiedenartigkeit von Anwendungssystemen ist ansonsten so groß, dass hier nicht weiter darauf eingegangen werden kann.

8.4 Aufgaben zum Thema: System- und Anwendungssoftware

In diesem Unterkapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt. Es empfiehlt sich, die Aufgaben selbstständig auf einem Blatt Papier zu lösen. Musterlösungen werden während des Semesters nach und nach freigeschaltet. Aus ähnlichen Aufgaben kann sich die Klausur zusammensetzen.



Aufgabe

Aufgabe 8.1

Ein Programm einer höheren Programmiersprache ist übersetzt und zu einem ausführbaren Programm gebunden worden. Alle für die Befehle und Operanden benötigten Adressen sind als Programmadressen ausgedrückt. Zur Ausführung wird dieses Programm in einen beliebigen Adressbereich des Hauptspeichers geschrieben. Geben Sie eine Funktion zur Adressumsetzung der Programmadressen in die Speicheradressen an.



Aufgabe

Aufgabe 8.2

Geben Sie den Ausdrucksbaum für den arithmetischen Ausdruck

$$a + 2 \cdot b - c \cdot (n - 1)$$

an.



Aufgabe

Aufgabe 8.3

Die nachfolgend aufgeführten Aussagen aus dem Hochschulbereich sind in einem Relationenmodell darzustellen:

Eine Hochschule ist gegliedert in mehrere Fachbereiche. Ein Fachbereich bietet mehrere Studiengänge an. Dozenten gehören zu jeweils einem Fachbereich. Dozenten halten Vorlesungen.

Studierende schreiben sich für einen Studiengang ein. Studierende hören Vorlesungen.

Fügen Sie den Relationen jeweils zwei bis drei sinnvolle Attribute hinzu.

8.5 Fragen mit Musterlösungen: System- und Anwendungssoftware

In diesem Kapitel sind Verständnisfragen mit ihren Musterantworten zu dem vorangehenden Lehrstoff zusammengestellt. Solche Fragen werden auch in der Klausur vorkommen.

Im Folgenden sind die Verweise zu den Fragen aufgelistet. Mit einem Klick auf 'Antwort' am Ende der Fragenseite bekommt man die Antwort für das gestellte Problem angezeigt. Ein Klick auf "Info-Basis ..." führt zu dem Kapitel des für die Aufgabe relevanten Lehrstoffes.

Das Textfeld dient z.Z. lediglich dazu, eigene Antworten zu notieren, um sie besser mit der Musterantwort vergleichen zu können. Eine Speichermöglichkeit des Textes ist damit jedoch nicht gegeben.



Aufgabe

Frage 8.1

Welche Aufgaben hat ein Betriebssystem?

Antwort

Die wichtigsten Programme eines Betriebssystems sind:

- Prozessorverwaltung
- Speicherverwaltung
- Dateiverwaltung
- Geräteverwaltung
- Auftragsverwaltung

Vgl. Kap. Betriebssysteme



Aufgabe

Frage 8.2

Was ist ein Prozess?

Antwort

Ein Prozess ist eine Folge von sequentiell abzuarbeitenden Aktionen. Eine solche Aktionsfolge kann ein gesamtes zur Ausführung gelangtes Programm oder auch einzelne Sequenzen davon sein. Stehen mehrere Prozessoren zur Verfügung, so können mehrere Prozesse parallel ausgeführt werden. Bei nur einem Prozessor können mehrere Prozesse nur 'quasi parallel' (im Multiplex-Betrieb) ausgeführt werden. Das heißt, dass ein Prozess an einer Stelle unterbrochen und ein anderer vom System wiederaufgenommen wird. Der unterbrochene Prozess wird später weiterbearbeitet. Ein Prozess kann nicht an einer beliebigen Stelle unterbrochen werden.

Vgl. Kap. Betriebssysteme



Aufgabe

Frage 8.3

Beschreiben Sie eine Client-Server-Rechnerarchitektur!

Antwort

In einer Client-Server-Architektur werden Kunden- und Anbieterprozesse (Clients und Server) um einen Betriebssystemkern (Kernel) angeordnet. Sie kommunizieren über Nachrichten miteinander. Der Betriebssystemkern verteilt diese Nachrichten und stellt die Schnittstelle zur Hardware dar.

Es wird zwischen 'Kernel-Modus' und 'User-Modus' unterschieden. Der Kernel-Modus ist der privilegierte Modus. Er darf alles, wird als fehlerfrei vorausgesetzt und beinhaltet den Betriebssystemkern. Der User-Modus ist der nichtprivilegierte Modus. Er hat nur eingeschränkte Befugnisse. Hier sind die Server und Clients angesiedelt. Auch Teile des Betriebssystems sind nur Server-Prozesse, haben also nur eingeschränkte Befugnisse. Das System ist dadurch sicherer und flexibler.

Die Kommunikation zwischen Client und Server beschränkt sich nicht auf einen Rechner. Client und Server können sich auf verschiedenen Rechnern befinden, welche durch ein Netzwerk verbunden sind (Verteilte Systeme). Dies funktioniert auch über verschiedene Hardware-Plattformen hinweg.

Vgl. Kap. Betriebssysteme

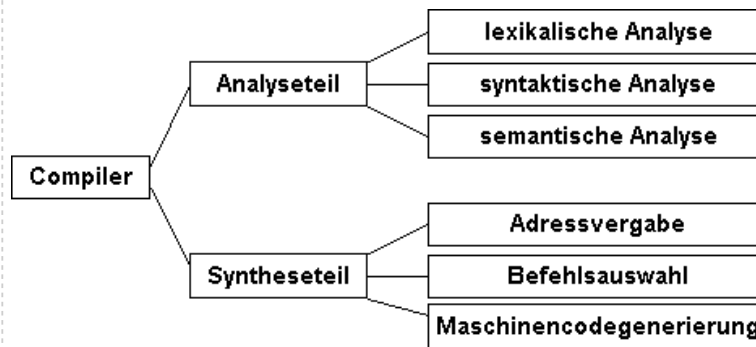


Aufgabe

Frage 8.4

Beschreiben Sie die Struktur eines Compilers!

Antwort



Vgl. Kap. Basissysteme



Aufgabe

Frage 8.5

Was ist ein Datenbanksystem?

Antwort

Ein Datenbanksystem (DBS) besteht aus:

- Einer (ggf. mehreren) Datenbank(en) und
- einem Datenbankverwaltungssystem (DBVS, auch Datenbankmanagementsystem DBMS).

Vgl. Kap. Basissysteme



Aufgabe

Frage 8.6

Beschreiben Sie das (Datenbank-) Relationenmodell.

Antwort

Relation: Zweidimensionale Tabelle, Identifikation durch einen Namen
Relationenschema: Beschreibung der Relationen durch ihre Attribute (Spalten der Tabelle); den Attributen sind Attributwerte zugeordnet
Tupel: Jede Eintragszeile

der Tabelle, bestehend aus Attributwerten Datenbank: Zusammenfassung mehrerer Relationen

Vgl. Kap. Basissysteme

9 Rechnernetze



Gliederung

9 Rechnernetze

9.1 Datenkommunikation

9.2 Aufgaben von Rechnernetzen

9.3 Ausdehnung von Rechnernetzen

9.4 Netzstrukturen und -architekturen

9.5 Internet und Dienste im Internet

9.6 Aufgaben zum Thema Rechnernetze

9.7 Fragen mit Musterlösungen: Rechnernetze

Die heutigen Arbeitsplatzrechner (Personal Computer und Workstation) stellen den Nutzern eine Vielzahl verschiedener Anwendungen zur Verfügung. Es gibt darüber hinaus aber viele Aufgaben, die erst durch die Vernetzung von Rechnern effizient bzw. überhaupt bearbeitet werden können.

9.1 Datenkommunikation

Informationen und Daten (Nachrichten) werden zwischen Menschen, Menschen und Rechnern und zwischen Rechnern ausgetauscht. Ein Kommunikationsnetz sorgt für die Verbindung zwischen den Endgeräten und der eigentlichen Datenübertragung. Die Sender werden allgemein auch Datenquellen und die Empfänger Datensinken genannt.

Mensch-Mensch-Kommunikation

Die Kommunikation zwischen Menschen, wie z. B. über Telefon, kann im direkten Dialog (bidirektional) erfolgen oder auch unidirektional, wie z. B. bei einem Telefax. Eine solche Kommunikation erfolgt über spezielle Endgeräte und spezielle Netze. Solche Kommunikationsarten gibt es auch über Rechner und Rechnernetze, z. B. Terminal-Terminal- Kommunikation und elektronische Post. Da hier jedoch beide Teilnehmer jeweils über einen Rechner mit bestimmten Anwendungsprogrammen kommunizieren, lässt sich technisch diese Kommunikation auf eine Rechner-Rechner-Kommunikation zurückführen (siehe unten).

Mensch-Rechner Kommunikation

Im Normalfall sind die Ein- und Ausgabegeräte (Tastatur, Maus und Bildschirm) direkt an einen Rechner angeschlossen. Darüber hinaus ist es auch möglich, sich in entfernte Rechner über ein Rechnernetz einzuwählen. Das Einwählen erfolgt über ein spezielles Anwendungsprogramm auf dem lokalen Rechner. Demnach liegt auch hier wieder eine Rückführung auf eine Rechner-Rechner-Kommunikation vor.

Rechner-Rechner-Kommunikation

Rechner kommunizieren über ein Rechnernetz miteinander. Spezielle Anwendungsprogramme stehen bei allen Kommunikationspartnern für die Verbindung und die eigentliche Datenübertragung zur Verfügung.

Die Datenkommunikation lässt sich weiterhin anhand der Art der zu transferierenden Daten einteilen, was unterschiedliche technische Realisierungen nach sich zieht.

Datenvolumen:

Anzahl der zu übertragenden Datenbytes

Kommunikationsrichtung:

Daten werden an einen / mehrere Empfänger versendet oder Daten werden zwischen mehreren Teilnehmern hin- und her gesendet (Dialog)

Zeitbedarf

Datenübertragung ist zeitkritisch / nicht zeitkritisch

Insbesondere die Kombination einiger obiger Kriterien erfordert speziell angepasste technische Realisierungen. Zur Beschreibung dieser Kriterien sind einige technische Begriffe zu nennen.

Bei einer Datenübertragung nur in einer Richtung wird von einem **Simplexverfahren** gesprochen. Können zwischen zwei Endgeräten in beiden Richtungen im Wechselbetrieb Daten übertragen werden, wird von einem **Halbduplexverfahren** gesprochen, und können die Endgeräte gleichzeitig senden und empfangen, von einem **Duplexverfahren**.

Des Weiteren wird zwischen einer **synchronen** und **asynchronen** Übertragung unterschieden. Bei einer synchronen Übertragung gibt es über eine zusätzliche Taktleitung eine Synchronisation zwischen Sender und Empfänger, bei einer asynchronen Übertragung jedoch nicht. Hier müssen Steuerinformation und Daten auf ein und derselben Leitung übertragen werden.

Ein Maß für das Datenvolumen, das je Zeiteinheit übertragen werden kann, ist die **Übertragungsrate**, gemessen in Bit pro Sekunde (bps). Sie hängt von dem Frequenzbereich, der so genannten **Bandbreite**, der Leitung ab. Kupferdrähte haben z.B. eine geringere Bandbreite als Glasfaserkabel. Die maximale Übertragungsgeschwindigkeit wird begrenzt durch die Anzahl der Pegelwechsel pro Zeiteinheit (gemessen in baud). Zur Codierung eines Bit sind mehr als ein

Pegelwechsel notwendig. Da aber im Durchschnitt bei guten Codierungen nur wenig mehr als ein Pegelwechsel pro Bit benötigt wird, kann 1 baud ungefähr mit 1 Bit/s gleichgesetzt werden.

9.2 Aufgaben von Rechnernetzen

Die Vernetzung von Rechenanlagen dient dazu, diverse Aufgaben besser zu erfüllen, andere Aufgaben können dadurch überhaupt erst bearbeitet werden. Die wichtigsten Aufgaben sind nachfolgend aufgeführt.

Kommunikations- und Informationsverbund:

Bereitstellung von Geräten und Verbindungen zur Kommunikation (z. B. Sprache, Text) und Verbreitung von Informationen (z. B. Text, Bild, Video).

Funktionsverbund:

Bereitstellung von Geräten und Programmen, die nur auf einigen Rechnern des Netzes vorhanden sind (z. B. Vektorrechner, leistungsfähiger Mehrzweckrechner, Datenbankrechner, Datei-Server, Web-Server, E-Mail-Server).

Datenverbund:

Logische Kopplung von räumlich getrennten Datenbeständen, unterstützt durch Systemkonstrukte zur Sicherung von Konsistenz und Aktualität.

Verfügbarkeitsverbund:

Bereitstellung einer Mindestleistung, vor allem bei Ausfall einzelner Komponenten, zur Schaffung fehlertoleranter Systeme (z. B. Realzeitsysteme in einer industriellen Fertigungsumgebung und der Prozesssteuerung).

Leistungsverbund:

Mehrere Rechner des Verbundes arbeiten am selben Problem. Die Leistungssteigerung durch Zerlegung des Problems auf einzelne Rechner ist begrenzt durch die dabei maximal erreichbare Parallelität.

Lastverbund:

Einsatz von Betriebsmitteln schwach belasteter Rechner zur Entlastung stärker belasteter Rechner.

9.3 Ausdehnung von Rechnernetzen

Es gibt verschiedene Arten der Klassifikation von Rechnernetzen. Eine davon bezieht sich auf die räumliche Ausdehnung. Man spricht von **lokalen** und **globalen** Netzen.

LAN (Local Area Network, lokales Netz): Es dient zur Verbindung von Arbeitsplatzrechnern und Servern in einem räumlich begrenzten Gebiet (bis zu 10 km). Die Leitungen sind nur für die Betreiber zugänglich. Die Übertragungsraten gehen bis zu 1 GBit/s.

WAN (Wide Area Network, Weitverkehrsnetz): Es dient zur Verbindung von Rechnern über große Strecken über öffentliche Datenübertragungseinrichtungen (bis zu 1000 km). Die Übertragungsraten gehen bis zu 155 MBit/s.

MAN (Metropolitan Area Network, Stadtnetz): Es handelt sich um ein spezielles Weitverkehrsnetz, das sich über Ballungsräume (Städte) erstreckt (bis zu 100 km). Gegenüber allgemeinen Weitverkehrsnetzen zeichnet es sich insbesondere durch höhere Übertragungsraten aus (100 MBit/s bis 1 GBit/s).

GAN (Global Area Network, globales Netz): Es dient zur Verbindung von Rechnern weltweit. In diesem Netz werden logisch verschiedene LANs, MANs und / oder WANs zusammengefasst. Die typische Übertragungsrate liegt z. Z. bei 2 MBit/s, sie wird begrenzt durch die schwächste Verbindung.

Die Ausdehnung der oben aufgeführten Netze nimmt in der Reihenfolge LAN, MAN, WAN, GAN zu. Aus erklärungs-technischen Gründen wurde oben von dieser Reihenfolge abgewichen.

9.4 Netzstrukturen und -architekturen

Ein Rechnernetz (Netz, Netzwerk) besteht im Wesentlichen aus Rechnern als Datenendgeräte (Data Terminal Equipment) und den Übertragungsleitungen (Transport /Transmission System). Zwischen die Übertragungsleitungen sind in der Regel Vermittlungselemente (Data Circuit-Terminating Equipment) geschaltet. Übertragen werden im Netz Nachrichten (Meldungen, Botschaften, Messages). Technisch wird auch von zu übertragenden Paketen (Packages) gesprochen.

Neben den physikalischen Verbindungen in einem Rechnernetz sind für die Kommunikation noch die Art und Abfolge der zu übertragenden Pakete zu definieren. Sender und Empfänger müssen sich verstehen. Man spricht von einem Kommunikationsprotokoll oder kurz von einem Protokoll.

Topologien

Die Verbindungsstruktur, über die die Rechner in einem Netz miteinander verbunden sind, wird Netztopologie genannt. Es gibt die folgende Unterscheidungsmöglichkeiten für Topologien:

- Sternnetze
- Busnetze
- Ringnetze
- Baumartige Netze
- Vermaschte Netze

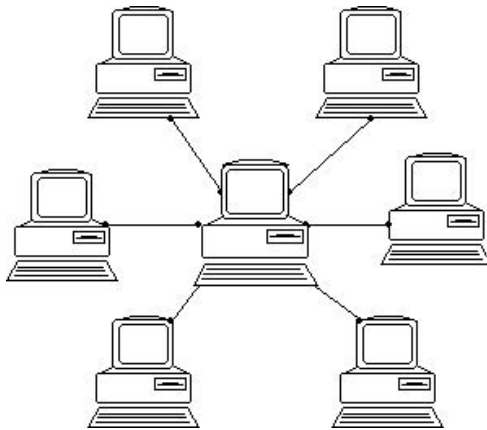
Keine Netztopologie stellt ein

- Netz von Netzen

dar. Die Vernetzung verschiedenartiger Netzwerke ist aber ein wichtiges Element, so dass es hier ebenfalls erläutert wird.

Sternnetze

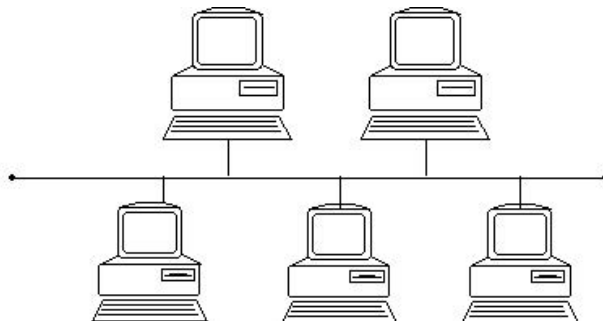
In einem Sternnetz sind alle Rechner sternförmig mit einem Zentralrechner verbunden (untenstehende Abbildung). Vom Zentralrechner werden z.B. reihum alle Rechner nach einem Sendewunsch abgefragt (Polling) und diesem Wunsch gegebenenfalls entsprochen.

**Beispiel eines Sternnetzes**

Busnetze

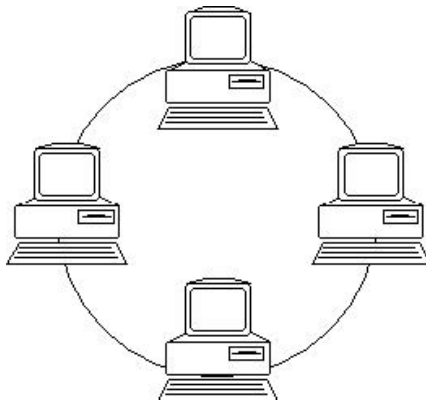
In einem Busnetz sind alle Rechner an eine Übertragungsleitung angeschlossen (untenstehende Abbildung). Es gibt keinen vermittelnden Rechner. Der Zugang muss von den gleichberechtigten Rechnern

untereinander geregelt werden, z. B. mittels eines Wettkampfverfahrens (CSMA/CD, Carrier Sense Multiple Access with Collision Detection).

**Beispiel eines Busnetzes**

Ringnetze

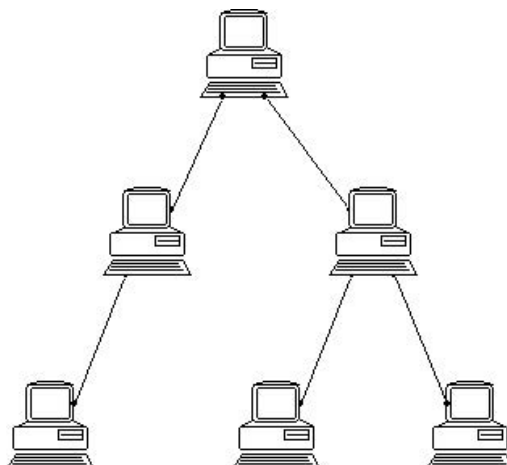
In einem Ringnetz sind alle Rechner über einen Leitungsring (geschlossene Übertragungsleitung) miteinander verbunden (untenstehende Abbildung). Es gibt auch hier keinen vermittelnden Rechner. Der Zugang zum Netz wird mittels einer von Rechner zu Rechner weiterzugebenden Berechtigungsmarke (Token) geregelt.



Beispiel eines Ringnetzes

Baumartige Netze

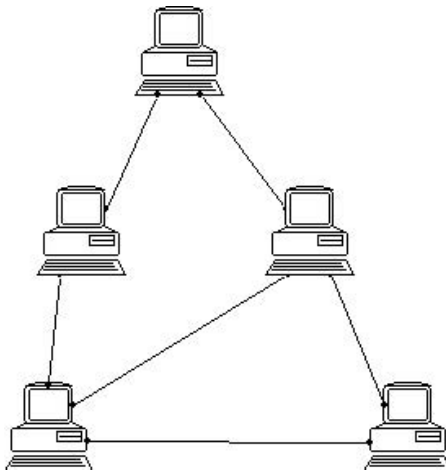
In einem baumartigen Netz sind die Rechner, ausgehend von einem Rechner als Wurzel, in einer baumartigen Struktur vernetzt (untenstehende Abbildung).



Beispiel eines baumartigen Netzes

Vermaschte Netze

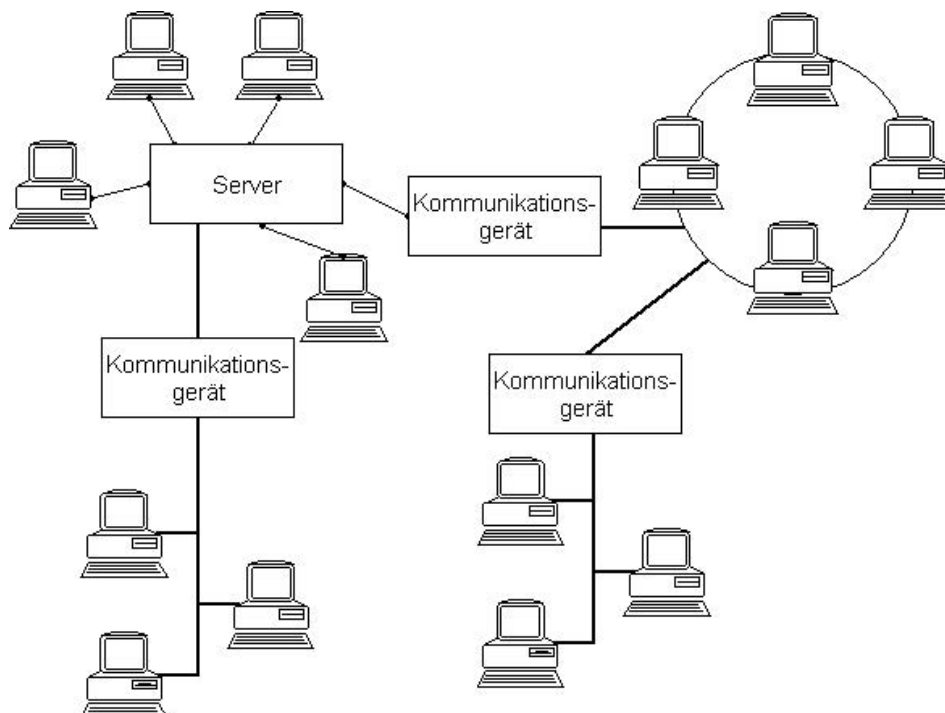
In einem vermaschten Netz sind die Rechner beliebig vernetzt (untenstehende Abbildung). Dies kann bis zu einer vollständigen Vermaschung gehen. Jeder Rechner ist dann mit jedem anderen Rechner über eine direkte Leitung verbunden.



Beispiel eines vermaschten Netzes

Netz von Netzen

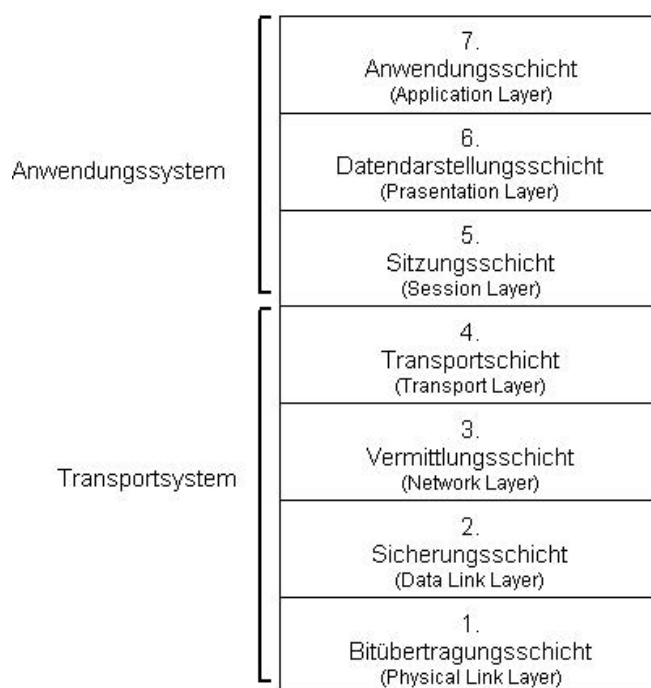
Gleichartige und unterschiedliche Netze können über sogenannte Kommunikationsgeräte, (Hubs, Transceiver, Router, Server etc.) miteinander gekoppelt werden (untenstehende Abbildung). Es sei hier nochmals darauf hingewiesen, dass solche Vernetzungen von Netzen selbst keine Topologien darstellen.



Beispiel von Verbünden verschiedener Netze

Das Schichtenmodell

Wie oben bereits erwähnt wurde, muss zur Kommunikation über ein Netzwerk neben den Geräten (Endgeräte und Leitungen) auch die Art und Abfolge der Übertragung definiert sein. Ein solches Kommunikationsprotokoll muss den gesamten Bereich von der physikalischen Signalübertragung bis zu den Anwendungssystemen, den Diensten (z. B. E-Mail), abdecken. Das OSI-Referenzmodell (Open System Interconnection) ist ein von der ISO (International Standard Organization) definiertes Protokoll für die Vernetzung offener Systeme. Das Protokoll ist in Schichten (Layer) aufgebaut, bei denen jede Schicht für die nächste, höhere eine gewisse Funktionalität bereitstellt und dabei die der nächst unteren Schicht nutzt (untenstehende Abbildung).



Das ISO - OSI - Schichtenmodell

Bitübertragungsschicht:

Es werden Bitfolgen übertragen.

Sicherungsschicht:

Die Bitfolgen werden als Datenpakete betrachtet, denen Adressen (MAC-Adresse, Media Access Control) und eine Bitfolge zur Fehlererkennung zugeordnet werden. Außerdem wird der geregelte Zugang zur Schicht 1 gewährleistet.

Vermittlungsschicht:

Der Übertragungsweg im Netz wird ermittelt. Den Datenpaketen werden die entsprechenden Adressen (IP-Adresse, Internet Protocol) zugeordnet, und sie werden übertragen. Dabei wird eine Fehlererkennung und -behandlung durchgeführt.

Transportschicht:

Es wird die Übertragungsqualität und -art zwischen Sender und Empfänger ausgehandelt und eine weitere Fehlerbehandlung ausgeführt. Große Nachrichten werden in Pakete segmentiert.

Sitzungsschicht:

Die Verbindung während einer Sitzung wird gewährleistet. Es werden die Anmeldung, Passwortabfrage und Dialogsteuerung sowie die Abmeldung durchgeführt.

Darstellungsschicht:

Es werden unterschiedliche Datencodierungen konvertiert. Die Daten werden komprimiert bzw. dekomprimiert, sowie verschlüsselt und entschlüsselt.

Anwendungsschicht:

Hier werden die anwendungsspezifischen Dienste, wie z.B. E-Mail, Dateitransfer und Nutzung des World Wide Web (WWW), realisiert.

Die Schichten 1 bis 4 werden dem Transportsystem zugeordnet, die Schichten 5 bis 7 dem

Anwendungssystem. Häufig ist die Schichtenaufteilung im Anwendungssystem nicht sichtbar, da die Funktionalitäten der Sitzungs- und Darstellungsschicht in die Anwendungsschicht integriert sind.

Eine Zuordnung von Diensten und Protokollen zeigt die Tabelle:

	OSI-Schicht	Dienst	Protokoll
7	Anwendungsschicht	E-Mail	SMTP (Simple Mail Transfer Protocol), POP (Post Office Protocol)
		Dateitransfer	FTP (File Transfer Protocol)

		WWW-Nutzung	HTTP (Hypertext Transfer Protocol)
		entfernte Bildschirmsitzung	Telnet (Telecommunication Network)
6	Darstellungsschicht		
5	Sitzungsschicht		
4	Transportschicht		TCP (Transmission Control Protocol)
3	Vermittlungsschicht		IP (Internet Protocol)
2	Sicherungsschicht		Ethernet, ATM (Asynchronous Transfer Mode)
1	Bitübertragungsschicht		

|



Zuordnung von Diensten und Protokollen zu OSI-Schichten

Auf das OSI-Referenzmodell wird auch aufgesetzt, um Übergänge zwischen Netzen logisch und technisch zu definieren:

Repeater werden verwendet, um Netze auf der Bitübertragungsschicht zu verbinden. Es kann damit also die Ausdehnung und Topologie von Netzen erweitert werden. Die Signale werden verstärkt und regeneriert.

Zur Verbindung von Netzen über die Sicherungsschicht werden **Bridges** eingesetzt. Es werden Pakete zwischengespeichert. Weitergeleitet werden die Pakete nur, wenn sie fehlerfrei sind und sie an einen Empfänger auf der anderen Netzseite adressiert sind. Es wird somit eine Lasttrennung der Netze erreicht.

Router verbinden Netze über die Vermittlungsschicht. Dort ist die gesamte Netztopologie bekannt und es können gezielt Übertragungswege ausgewählt werden. Dabei wird ein Router als Vermittler adressiert.

Über ein **Gateway** werden Netze ab Schicht 4 aufwärts verbunden. Z.B. ist die SNA (System Network Architecture) von IBM ein Netzwerk mit einem ganz anders strukturierten Aufbau. Es kann über ein SNA-Gateway angebunden werden.

9.5 Internet und Dienste im Internet

Das **Internet** ist ein weltweiter Verbund von Rechnernetzen. In ihm ist eine Familie von Protokollen zur Datenübertragung vereinbart, und es hat jeder Rechner eine eindeutige Adresse. Die Netze sind über Vermittlungsrechner (Router) miteinander verbunden. Ein Vorteil dieses Netzes ist seine dezentrale Struktur. Es gibt keine zentrale Steuerung. Bei Ausfall einzelner Komponenten bleiben alle anderen in Betrieb, und es können andere Übertragungswege gefunden werden.

TCP (Transmission Control Protocol) und IP (Internet Protocol) sind die wichtigsten Protokolle im Internet. Es gibt eine Vielzahl weiterer Protokolle. Die wesentliche Aufgabe des IP liegt darin, die Daten von einem Rechner zu einem anderen Rechner zu übertragen. Die Daten werden in Datenpakete verpackt, mit der Adresse des nächsten Vermittlungsrechners versehen und dorthin geschickt. Über viele Zwischenstationen gelangt das Datenpaket an den Vermittlungsrechner, in dessen Bereich (Domain) sich der Zielrechner befindet. Da die Pakete nicht unbedingt in der richtigen Reihenfolge ankommen, weil sie über unterschiedliche Wege auf den Zielrechner gelangt sein können, müssen diese dann erst wieder richtig zusammengesetzt werden. Die Segmentierung der Nachrichten in Datenpakete und ihr Zusammensetzen sind die wesentlichen Aufgaben von TCP.

Jeder Vermittlungsrechner hat ein lokales Benutzer-Namensverzeichnis und ist seinerseits in einem oder mehreren globalen Rechner-Namensverzeichnissen eingetragen. Die Rechner-Namen (IP- Adresse) bestehen aus vier jeweils durch einen Punkt getrennten Zahlen, wobei jede Zahl in einem Byte dargestellt wird, z.B. 141.41.32.110.

Dienste im Internet:

- E-Mail
- News
- Dateitransfer
- Entfernte Bildschirmsitzung
- World Wide Web (WWW)
- ...

E-Mail

Der E-Mail-Dienst realisiert eine elektronische Post. Über das Internet werden primär Text-Dateien versendet. An solche Dateien können aber auch Dateien anderer Typen (z. B. Bilder) angehängt werden. Alle Teilnehmer benötigen eine Adresse. Sie ist wie folgt aufgebaut:

Eine Domain ist ein geographischer oder organisatorischer Bereich, wie z. B. "de" für Deutschland. Eine Subdomain ist ein Unterbereich oder auch schon der Name eines Vermittlungsrechners. Der Name ist in der Regel der Login-Name des Benutzers.

Name@Subdomain.Domain

Kommt eine E-Mail bei dem Vermittlungsrechner an, so wird sie von dem dafür zuständigen Programm, dem Mail-Server, empfangen, gespeichert und der Benutzer informiert. Mittels eines Programms auf dem Arbeitsplatzrechner des Benutzers, dem Mail-Client, kann diese E-Mail gelesen und weiterbearbeitet oder auch neue E-Mails geschrieben und versendet werden.

News

Der News-Dienst realisiert eine Art Diskussionsforum. Alle Teilnehmer eines solchen Dienstes können Texte einreichen, die dann allen zum Lesen zur Verfügung gestellt werden, wie das Anbringen eines Zettels an ein Schwarzes Brett. Auf diese Weise kann über ein Thema offen diskutiert oder es können Fragen gestellt werden, in der Hoffnung, einer der Teilnehmer könne sie vielleicht beantworten.

Dateitransfer

Der Dateitransfer-Dienst dient zur Übertragung von Dateien. Ist man auf dem entsprechenden

Rechner regulärer Nutzer, so meldet man sich entsprechend an und hat damit die üblichen Lese- und Schreibrechte auf diesem Rechner. Es gibt aber auch öffentliche Datei-Archive, so genannte anonyme FTP-Server, auf die jeder zugreifen kann. Die Adressierung erfolgt über

Rechnername.Domain

Entfernte Bildschirmsitzung

Mittels einer entfernten Bildschirmsitzung ist es möglich, sich auf einem beliebigen Rechner im Internet anzumelden, vorausgesetzt man hat dort einen Benutzernamen (remote login). In der Regel erhält man aber nur einen Zugang über eine zeichenorientierte Bildschirmemulation, so dass zeilenorientierte Kommandos abgearbeitet werden können. Die Adressierung erfolgt wie beim Dateitransfer-Dienst über

Rechnername.Domain

World Wide Web (WWW)

Der WWW-Dienst realisiert ein verteiltes, hypermediales Informationssystem. Auf WWW-Servern im Internet werden in hierarchisch organisierten Dateisystemen Hypertext-Dokumente (Dateien im HTML-Format, Hypertext Markup Language) bereitgestellt. Über WWW-Clients, so genannte Browser, können diese Dateien abgerufen und lokal dargestellt werden. Der Zugriff erfolgt über das Protokoll HTTP (Hypertext Transfer Protocol). Eine Adresse hat häufig die folgende Form:

`http://www.Subdomain.Domain`

Ein solche Adresse wird als URL (Universal Resource Locator) bezeichnet. "www" ist meist der Rechnername des WWW-Servers, er kann aber auch einen anderen Namen haben. Über das Anhängen von /Pfad/Dateiname kann direkt auf bestimmte Dateien zugegriffen werden. Ist eine solche Datei nicht spezifiziert, wird die Datei index.html aufgerufen.

In einem Hypertext-Dokument wird zunächst einmal ein textueller Inhalt spezifiziert, dieser strukturiert und Hinweise zu seiner Darstellung (z. B. Font, Schriftgröße, Fettdruck) gegeben. Integriert werden können Bilder, Audio, Video und auch auf dem Client oder Server ablauffähige Programme, so genannte Applets bzw. Servlets. Es sollte also eher von Hypermedia-Dokumenten gesprochen werden.

Der Aspekt allerdings, der dieses über das Internet verteilte Informationssystem so leicht bedienbar macht, ist die Möglichkeit, in die Texte und Bilder der Dokumente so genannte Links (Verweise auf andere URLs) zu integrieren, so dass per Mausklick andere Dokumente von anderen Rechnern abgerufen werden können. Für einen Benutzer stellt sich also der Zugriff über das Rechnernetz völlig transparent dar.

9.6 Aufgaben zum Thema Rechnernetze

In diesem Unterkapitel sind Übungsaufgaben zu dem vorangehenden Lehrstoff zusammengestellt. Es empfiehlt sich, die Aufgaben selbstständig auf einem Blatt Papier zu lösen. Musterlösungen werden während des Semesters nach und nach freigeschaltet. Aus ähnlichen Aufgaben kann sich die Klausur zusammensetzen.



Aufgabe

Aufgabe 9.1

Das Ethernet ist die Netztechnologie, die als der am weitesten verbreitete LAN-Standard eingesetzt wird. Das dort bei Busnetzen eingesetzte Zugriffsverfahren, das den Zugang zum Netzwerk steuert, ist CSMA/CD. Erläutern Sie unter Zuhilfenahme geeigneter Literatur die Technologie des Ethernets und seines Zugriffsverfahrens.

9.7 Fragen mit Musterlösungen: Rechnernetze

In diesem Kapitel sind Verständnisfragen mit ihren Musterantworten zu dem vorangehenden Lehrstoff zusammengestellt. Solche Fragen werden auch in der Klausur vorkommen.

Im Folgenden sind die Verweise zu den Fragen aufgelistet. Mit einem Klick auf 'Antwort' am Ende der Fragenseite bekommt man die Antwort für das gestellte Problem angezeigt. Ein Klick auf "Info-Basis ..." führt zu dem Kapitel des für die Aufgabe relevanten Lehrstoffes.

Das Textfeld dient z.Z. lediglich dazu, eigene Antworten zu notieren, um sie besser mit der Musterantwort vergleichen zu können. Eine Speichermöglichkeit des Textes ist damit jedoch nicht gegeben.



Aufgabe

Frage 9.1

Erläutern Sie die Begriffe Datenvolumen, Übertragungsrate und Bandbreite!

Antwort

Datenvolumen: Anzahl der zu übertragenden Datenbytes Übertragungsrate: Anzahl Bits pro Sekunde (Maß für das je Zeiteinheit übertragbare Datenvolumen) Bandbreite: Frequenzbereich einer Leitung, in der Signale übertragen werden können; häufig gleichgesetzt mit maximaler Übertragungsrate einer Leitung.

Vgl. Kap. Datenkommunikation



Aufgabe

Frage 9.2

Nennen Sie Topologien von Netzwerken!

Antwort

- Sternnetz
- Busnetz
- Ringnetz
- baumartiges Netz
- vermaschtes Netz

Vgl. Kap. Netzstrukturen und -architekturen



Aufgabe

Frage 9.3

Erläutern Sie das OSI Referenzmodell!

Antwort

Das OSI Referenzmodell ist eine Protokolldefinition für die Vernetzung offener Systeme. Es besteht aus sieben Schichten (von oben nach unten):

- Anwendungsschicht
- Datendarstellungsschicht
- Sitzungsschicht
- Transportschicht
- Vermittlungsschicht
- Sicherungsschicht
- Bitübertragungsschicht

Vgl. Kap. Netzstrukturen und -architekturen



Aufgabe

Frage 9.4

Was ist das Internet? Nennen Sie einige Dienste im Internet!

Antwort

Das Internet ist ein weltweiter Verbund von Rechnernetzen. In ihm ist eine Familie von Protokollen zur Datenübertragung vereinbart und jeder Rechner hat eine eindeutige Adresse. Die Netze sind über Vermittlungsrechner (Router) miteinander verbunden. Ein Vorteil dieses Netzes ist seine dezentrale Struktur. Es gibt keine zentrale Steuerung. Bei Ausfall einzelner Komponenten bleiben alle anderen in Betrieb. Somit können andere Übertragungswege gefunden werden.

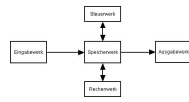
Dienste im Internet: E-Mail, News, Dateiübertragungsdienst, entferntes Anmelden, WorldWideWeb

Vgl. Kap. Internet und Dienste im Internet

I Abbildungsverzeichnis



Digitalisierung eines Bildes (analog - diskret - digital)..... 8



Blockbild eines von Neumann-Rechners..... 10



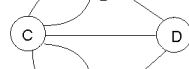
Darstellung der Modellbildung..... 14



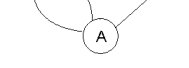
Das Königsberger Brückenproblem..... 15



Das Königsberger Brückenproblem als Graph..... 16



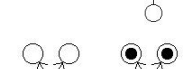
Stellen und Transitionen für Petri-Netze..... 19



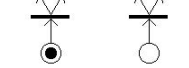
Beispiele für eine schaltende Transition mit drei Stellen..... 19



sequentieller Prozess..... 20



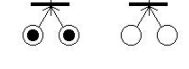
nebenläufiger Prozess..... 20



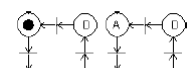
nebenläufiger Prozess mit wechselseitigem Ausschluß..... 21



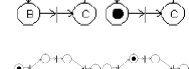
Erzeuger-Verbraucher-Prozess..... 21



Beispiel für ein ER-Schema..... 22



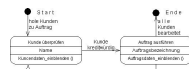
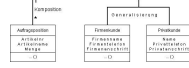
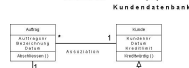
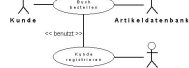
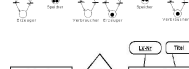
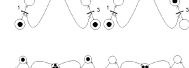
Beispiel eines Anwendungsfalldiagramms..... 24

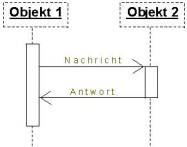
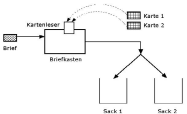


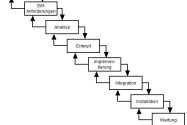
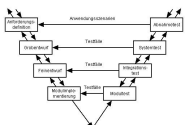
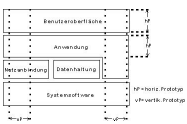
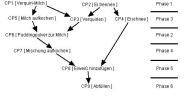
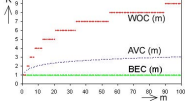
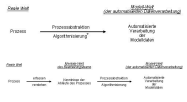
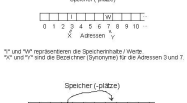


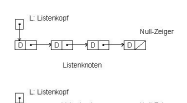







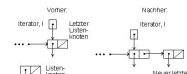
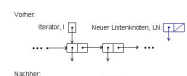
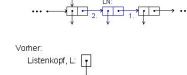
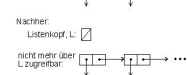


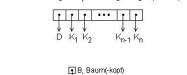
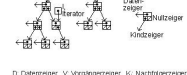
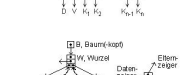
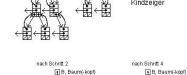

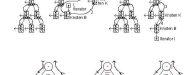


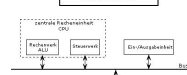
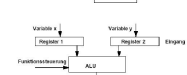

Klassen und ihre Beziehungen..... 26

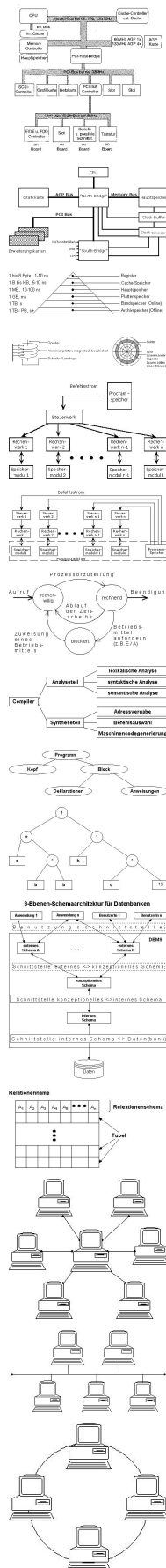


Beispiel eines Zustandsdiagramms..... 26



	Beispiel eines Sequenzdiagrammes.....	27
	Briefsortiermaschine.....	29
	Der Simulationsbegriff der Angewandten Informatik.....	30
	Der Simulationsbegriff der angewandten Informatik in der Praxis.....	33
	Das Wasserfallmodell.....	35
	Das V-Modell.....	36
	Das Prototypen-Modell.....	37
	PCP [CP in Phasen].....	80
	Rechenzeitbestimmung.....	88
	Algorithmisierung.....	89
	Algorithmisierungspraxis.....	89
	Speicherung eines Wertes und seine Adresse.....	98
	Speicherung eines Pointers auf einen Wert.....	99
	Array F aus Buchstaben.....	100
	Eine Liste.....	105
	Naive einfach verkettete Liste.....	105
	Einfach verkettete Liste.....	106
	Einfach verkettete Liste mit unterschiedlichen Datentypen.....	106

	Einfügen eines ersten Listenknotens.....	108
	Einfügen eines neuen letzten Listenknotens.....	108
	Einfügen zwischen zwei Listenknoten.....	109
	Löschen einer kompletten Liste.....	111
	Geografische Ortsangaben, die einander teilweise enthalten.....	112
	Ein Baum als Abstraktion einer Struktur.....	113
	Ein Baumknoten.....	113
	Ein einfacher Binärbaum.....	114
	Ein Baumknoten mit Vorgängerzeiger.....	115
	Ein Binärbaum mit Eltern-Zeigern.....	115
	Löschen eines inneren Baumknotens in einem einfachen Binärbaum.....	116
	Einfügen eines inneren Binärbaumknotens vor einem Blatt.....	117
	Traverse durch einen Baum mit Eltern-Zeigern.....	118
	Schichtenmodell.....	129
	von Neumann'sches Rechnermodell.....	131
	ALU mit Registern.....	132
	Struktur eines Computers mit sechs Ebenen.....	137
	138



Bussysteme im IBM-kompatiblen PC.....147

North- und South-Bridge..... 147

Speicherhierarchie..... 148

Plattenstapel, Struktur einer Festplatte..... 149

Informationsfluss im SIMD-Rechner.....157

Informationsfluss im MIMD-Rechner..... 158

Prozesszustände.....167

Struktur eines Compilers..... 172

Programmstruktur..... 173

Ausdrucksbaum173

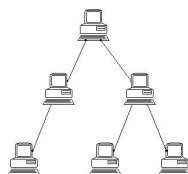
3-Ebenen-Schemaarchitektur für Datenbanken..... 175

Relation des Relationenmodells.....176

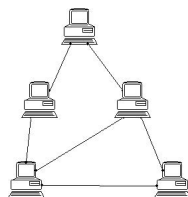
Beispiel eines Sternnetzes.....190

Beispiel eines Busnetzes..... 191

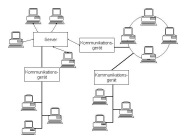
Beispiel eines Ringnetzes.....191



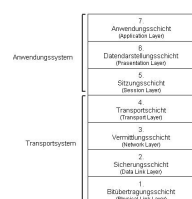
Beispiel eines baumartigen Netzwerkes.....192



Beispiel eines vermaschten Netzwerkes..... 192













Beispiel von Verbänden verschiedener Netze..... 193



Das ISO - OSI - Schichtenmodell..... 194

II Tabellenverzeichnis

	Beispiele zu Graphen.....	17
	Codes für die Ziffern.....	45
	Zeichenvorrat des ISO-7-Bit-Code.....	46
	Wahrheitstafel logisches und.....	96
	Wahrheitstafel logisches oder.....	97
	Dynamisches Verhalten des Algorithmus 'Sortieren durch Einfügen'	122
	Komponenten eines Ein- und Ausgabe-Systems.....	169
	Client-Server-Architektur.....	171
	Datenverarbeitung vs. Wissensverarbeitung.....	179
	Zuordnung von Diensten und Protokollen zu OSI-Schichten.....	195

III Medienverzeichnis



Beispiel Zahlenkonvertierung.....63


























Rechner zum Euklid-Algorithmus.....83


















Beispiel Sortieren durch Einfügen (420KB).....124

IV Aufgabenverzeichnis

	Aufgabe 3.1.....	27
	Aufgabe 3.2.....	28
	Aufgabe 3.3.....	28
	Aufgabe 3.4.....	28
	Aufgabe 3.5.....	29
	Frage 3.1.....	38
	Frage 3.2.....	38
	Frage 3.3.....	39
	Frage 3.4.....	39
	Frage 3.5.....	39
	Frage 3.6.....	40
	Frage 3.7.....	40
	Frage 3.8.....	41
	Frage 3.9.....	41
	Frage 3.10.....	42
	Aufgabe 4.1.....	51
	Aufgabe 4.2.....	51
	Aufgabe 4.3.....	51
	Frage 4.1.....	52
	Frage 4.2.....	53
	Frage 4.3.....	53
	Frage 4.4.....	53
	Aufgabe 5.1.....	72

	Aufgabe 5.2.....	72
	Aufgabe 5.3.....	72
	Frage 5.1.....	73
	Frage 5.2.....	74
	Frage 5.3.....	74
	Frage 5.4.....	74
	Frage 5.5.....	75
	Aufgabe 6.0.....	112
	Aufgabe 6.1	125
	Aufgabe 6.2	125
	Aufgabe 6.3.....	126
	Aufgabe 6.4	126
	Aufgabe 6.5	127
	Aufgabe 6.6	127
	Aufgabe 7.1.....	161
	Aufgabe 7.2.....	161
	Aufgabe 7.3.....	161
	Aufgabe 7.4.....	161
	Aufgabe 7.5.....	162
	Aufgabe 7.6.....	162
	Frage 7.1.....	163
	Frage 7.2.....	163
	Frage 7.3.....	163
	Frage 7.4.....	164
	Frage 7.5.....	164

	Frage 7.6.....	165
	Aufgabe 8.1.....	180
	Aufgabe 8.2.....	181
	Aufgabe 8.3.....	181
	Frage 8.1.....	182
	Frage 8.2.....	182
	Frage 8.3.....	183
	Frage 8.4.....	183
	Frage 8.5.....	184
	Frage 8.6.....	184
	Aufgabe 9.1.....	200
	Frage 9.1.....	200
	Frage 9.2.....	201
	Frage 9.3.....	201
	Frage 9.4.....	202

V Index

#

1-Komplement 64

2-Komplement 64

A

AGP 145

ALU 130

Abstraktionsgrad der Programmierung 8

Addition 67

Akkumulator 132

Akteur 24

Aktion 24

Algorithmus 76

Algorithmus des Euklid 76

Algorithmus intuitiv 76

Amdahls Gesetz 153

Analoge Darstellung 8

Analoge Nachricht 43

Analysemethoden 24

Angewandte Simulation 29

Anwendung 24, 179

Anwendungsfalldiagramm 24

Anwendungssystem 179

Applikation 179

Assembler 171

Assemblerprogramm 8

Assoziation 24

Asynchrone Übertragung 186

Asynchroner Bus 145

Attribut 13, 22, 24, 171

Auftragssteuersprache 166

Auftragsverwaltung 166

Axiomatisch 55

B

B-1-Komplement 64

B-2-Komplement 64

B-Komplement 64

Bandbreite 186

Basissystem 171

Baum 15

Baumartiges Netz 189

Befehlsregister 132

Befehlssteuerung 138

Befehlszyklus 132

Befehlszähler 132

Benchmark 150

Betriebsmittelverwaltung 138

Betriebssystem 136, 166

Beziehung 13, 24

Bildschirmsitzung 197

Binder 171

Binäres Zahlensystem 55

Blockierter Prozess 166

Bottom-up 34, 89

Bridge 145, 189

Bus 145

Busbreite 145

Busnetz 189

Busprotokoll 145

Bussysteme 145

Bäume 112

Büroanwendung 171

C

CCD 141

CD 148

CISC-Architektur 132

CMOS-Bildsensor 141

CPU 128

CP (Cremepudding) 76

Cantorsches Diagonalverfahren 55

Client-Server-Architektur 166

Code 45

Codierung 45

Coloured Petri-Nets 18, 18

Compiler 171

D

DBMS 171

DBS 171

DBVS 171

DDL 171

DML 166

DVD 148

Darstellung 8

Datei 166

Dateistruktur 166

Dateisystem 166

Dateitransfer 197

Dateiverwaltung 166

Datenbank 171

Datenbank-Konsistenz 171

Datenbank-Operation 171

Datenbank-Schema 171

Datenbankmanagementsystem 171

Datenbankmodell 171
Datenbanksystem 171
Datenbankverwaltungssystem 171
Datendefinitionssprache 171
Datenkommunikation 186
Datenmanipulationssprache 171
Datenpaket 186
Datenschutz 171
Datenverarbeitung 179
Datenverbund 171
DatenverbundDatenverbund 188
Dead-Lock 166
Decodierung 45
Device-Driver 166
Dhrystone-Benchmark 150
Differenztechnik 128
Digitale Nachricht 43
Directory 166
Diskrete Darstellung 8, 8
Domain 197
Duales Zahlensystem 55

E
E-Mail 197
ER-Modell 22
Echtes Komplement 64
Editor 171
Effektivität 84
Effektor 43
Effizienz 84
Ein-/Ausgabeorganisation 138
Einerkomplement 64
Empfänger 43
Entity 22
Entity-Relationship-Modell 22
Entität 22
Entwurf von Algorithmen 89
Entwurfsmethode 24
Ereignis 24
Erstes Cantorsches Diagonalverfahren 55
Expertensystemen 179
Externer Speicher 148
Externes Datenbank-Schema 171

F
FTP 197
Festplatte 148
File-System 166
Flynn 156

Front-Side-Bus 145
Funktionsverbund 188

G
GAN 189
Ganze Zahlen 55
Gateway 189
Gekoppelter Prozess 166
Generalisierung 24
Gerichteter Graph 15
Gerätetreiber 138, 166
Geräteverwaltung 166
Gleitpunktzahl 70
Global Area Network 189
Globales Netz 189
Graph 15

H
HTML 197
HTTP 197
Halbduplexverfahren 186
Halbleiterspeicher 148
Hardware-Schnittstelle 166
Harvard-Architektur 130
Headcrash 148
Hexadezimaales Zahlensystem 55

I
IDE-Bus 145
ISA-Bus 145
Informatik 7
Information 43
Informationsverarbeitung 48
Informationsverbund 188
Intelligente Datenverarbeitung 179
Internes Datenbank-Schema 171
Interpretationsvorschrift 43
Interrupt 132, 138
Interrupt-Service-Routine 138
Interrupt-Vektortabelle 138
Interruptcontroller 138
Interrupthandler 138
Intuitive Simulation 29

K
Kanal 43
Kante 15, 24
Kaufmännisch-administratives System 179
Klasse 24

Klassendiagramm 24
Knoten 15
Kommunikation 186
Kommunikationsverbund 188
Komplexe Zahlen 55
Komplexität 84
Konventionelle Datenverarbeitung 179
Konvertierung 59
Konzept 13
Kritischer Abschnitt 166

L

LAN 189
Lastverbund 188
Leistungsbewertung 150
Leistungsverbund 188
Lexikalische Analyse 171
Linker 171
Linpack-Benchmark 150
Load Balancing 158
Local Area Network 189
Logische Dateistruktur 166
Logisches Ein-/Ausgabesystem 138
Lokales Netz 189

M

MAN 189
MIMD-Architektur 156, 158
MIPS 150
MISD-Architektur 156
Maschinenprogramm 8
Maschinenprogramme 136
Maschinensprache 136
Master Bus 145
Mehrprozessorsysteme 153
Mehrrechnersysteme 153
Methode 24
Modell 13
Modellbildung 13
Multimedia 141
Multimedia-Peripherie 141
Multiplex-Betrieb 166
Mutual exclusion 166

N

Nachricht 43
Nachrichtengerät 43
Nachrichtenverarbeitung 48
Nachrichtenübertragung 43

Natürliche Zahlen 55
Nebenläufigkeit 18
Negative ganze Zahl 64
Netz von Netzen 189
Netzarchitektur 189
Netzstruktur 189
Netztopologie 189
News 197

O

Objektorientierte Entwurfsmethode 24
Oktales Zahlensystem 55

P

PCI-Bus 145
PCP (paralleler Cremepudding) 76
Paralleler Prozess 166
Parallelverarbeitung 153
Petri-Netz 18, 18
Petri-Netz-Nebenläufigkeit 18
Physikalisch Dateistruktur 166
Pipelining 132
Plug 138
Polling 138
Programm 8
Programmiersprache 8
Prototypen-Modell 34
Prozess 34, 166
Prozessor 132
Prozessorarchitektur 132
Prozessorverwaltung 166
Prozessverwaltung 166
Prozesszustand 166

Q

Quasi paralleler Prozess 166

R

Rapid prototyping 34
Rationale Zahlen 55
Reale Speicheradressierung 166
Rechenwerk 132, 132
Rechenwilliger Prozess 166
Rechnender Prozess 166
Rechnerarchitektur 128
Rechnernetz 189
Reele Zahlen 55
Refresh 148
Relais 43

Relation 13, 171
Relationale Datenbank 171
Relationenmodell 171
Relationship 22
Remote login 197
Repeater 189
Rezeptor 43
Ringnetz 189
Router 189

S

SIMD-Architektur 156
SISD-Architektur 156
SPEC-Benchmark 150
SPOOLER 138
Schichtenmodell ISO/OSI 189
Schichtenmodell Rechner 128
Schlüssel 171
Schlüsselattribut 22
Schnittstelle 166
Sedizmales Zahlensystem 55
Semantische Analyse 171
Semaphore 166
Sender 43
Sequentieller Prozess 166
Sequenzdiagramm 24
Simplexverfahren 186
Simulation 29
Slave Bus 145
Software-Entwicklungsprozess 34
Software-Schnittstelle 166
Speicheradresse 166
Speicheradressierung 166
Speicherart 130, 148
Speicherhierarchie 148
Speichertechnologien 148
Speicherverwaltung 166, 166
Stellenkomplement 64
Stellenwertcode 59
Sternnetz 189
Steuerwerk 132
Subtraktion 67
Synchrone Übertragung 186
Synchroner Bus 145
Synchronisation 171
Syntaxanalyse 171

T

TCP/IP 197

TPC-Benchmark 150
Technisch-wissenschaftliches System 179
Telnet 197
Theoretische Simulation 29
Thread 156
Timesharing-Systeme 153
Top-down 34, 89
Transaktion 171
Treiber 128
Tupel 171

U

UML 24
UML-Diagramm 24
USB 145
Ungerichteter Graph 15
Unified Modeling Language 24
Übertragungsgeschwindigkeit 186
Übertragungskanal 43
Übertragungsrate 186

V

V-Modell 34
Verfügbarkeitsverbund 188
Verklemmung 166
Vermaschtes Netz 189
Verstärker 43
Verteiltes System 189
Verzeichnis 166
Virtuell Speicheradressierung 166
Von Neumann-Rechner 130
Vorgehensmodell 34

W

WAN 189
WWW 197
Wandler 43
Wartender Prozess 166
Wasserfallmodell 34
Wechselseitiger Ausschluss 166
Wellenwiderstand 128
Whetstone-Benchmark 150
Wide Area Network 189
Wissensverarbeitung 179
World Wide Web 197

Z

Zahl 55
Zahlendarstellung 59, 59

Zahlengefühl 59
Zahlenmenge 55
Zahlensystem 55
Zustand 24
Zustandsdiagramm 24
Zweierkomplement 64
Zweites Cantorsches Diagonalverfahren 55