

TU Berlin

WS 2003

Ein Fragekatalog

Rechneraufbau, UNIX und C

Babak+Sascha

WS 2003

Betreut durch PD Dr. Obermayer

Inhaltsverzeichnis

1	Darstellung von Informationen	5
1.1	Bits und Bytes	5
1.2	Darstellung von Zeichen und Zahlen im Rechner	5
1.3	Logische Ausdrücke	7
2	Logische Schaltungen	9
2.1	Grundbegriffe	9
2.2	Einfache Logikschaltungen	9
2.2.1	Flip Flop	9
2.2.2	Halbaddierer	10
2.2.3	Volladdierer	11
3	Rechneraufbau	12
3.1	Teile des Rechners	12
3.2	Die Prozessorkarte	13
4	Betriebssystem UNIX	17
4.1	Allgemeines	17
4.2	Dateisystem	18
4.3	Prozeßsteuerung	21
4.4	UNIX-Shells	25
4.4.1	Shellprogrammierung	28
4.5	wichtige Unixbefehle	31
5	Die Programmiersprache C	32
5.1	Allgemeines	32
5.2	Übersicht über die Programmiersprache	33
5.3	Skalare Datentypen	35
5.4	Operatoren und Ausdrücke	39
5.4.1	Aus und Eingabefunktionen	41
5.5	Kontrollfluß	44
5.6	Arrays und Pointer	52
5.7	Speicherklassen	57
5.8	Strukturen und Unions	58
5.9	Verkettete Datenstrukturen	60

Inhaltsverzeichnis

5.10 Funktionen	61
5.11 Präprozessor	61
5.12 Ein- und Ausgabe	62

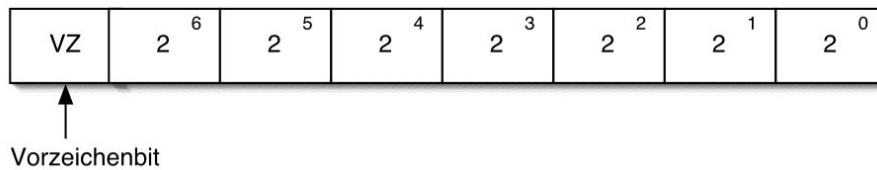
1 Darstellung von Informationen

1.1 Bits und Bytes

1. Was ist Codierung?
Übersetzung von Text in binäre Zeichenkette
2. Woraus besteht das binäre Alphabet?
1,0 oder hi ,low oder ein ,aus Wir verwenden 1 und 0
3. Was ist ein bit und ein Byte?
1 Element entspricht 1 bit 8 Elemente entsprechen 1 byte
4. Wieviel sind ein 1 MB , 1 GB, 1TB?
1 kb=1024 bytes =81292 bit
2 hoch20=1024 hoch2=1 MB
2hoch30=1024 hoch 3 =1 GB
2 hoch 40 =1024 hoch = 1TB

1.2 Darstellung von Zeichen und Zahlen im Rechner

1. Wie werden Zeichen und Zahlen rechnerintern dargestellt?
Rechnerintern wird die ASCII Codierung benutzt. Ursprünglich bestand sie aus 7 bits für ein Zeichen. Später wurde dies durch die ISO Norm 10646 ISO Norm 10646erweitert. Jetzt hat man 2 Byte pro Zeichen.=65536
ASCII ist ein Block und die Hexadezimale ein anderer Block (Potenzen von 16).
Ganze Zahlen
Die zwei Komplement Darstellung ist besser als die ASCII
2. Wie werden ganze Zahlen dargestellt?
Die ASCII Darstellung ist
a)aufwendig viele Bits
b) ungeschickt für Rechenoperationen (float, double).
Deshalb Zweikomplementdarstellung. Dabei nist eine feste Anzahl vo Bytes pro zahl festgelegt(1B, 2B, 4B, 8B). Die folgende Abbildung zeigt für 1 Byte die Darstellung. Es gibt ein Vorzeichenbit. Dabei entspricht 0=positiver Zahl und 1 =negativer Zahl.



Negative Zahlen werden invertiert und dann mit +1 addiert. Deswegen wahrscheinlich Zweierkomplementdarstellung. Der Begriff Zweierkomplementdarstellung ist bezogen auf die Darstellungsmöglichkeit der negativen Zahlen.

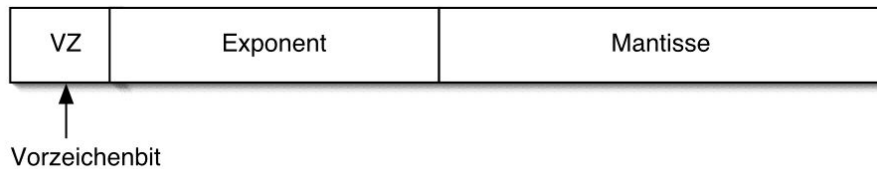
Man braucht zur Darstellung von Zeichen ASCII, da das binär-System (8 Bit) ja nur Werte von 0 (0) bis 255 (11111111) darstellen kann. Um also Zahlen wie 7060241 oder Buchstaben und Sonderzeichen darstellen zu können, braucht man ein zusätzliches System → ASCII. Jedes ASCII-Zeichen wird aus einem 1 Byte (8 Bit) dargestellt, das heisst es sind wiederum maximal 256 Zeichen möglich. Die binäre Darstellung und das Zweierkomplement von Zahlen ist also die Basis des ASCII-Zeichensatzes.

3. Was sind die Vorteile der Zweierkomplementdarstellung?

- eindeutige Darstellung der Null
- Vorzeichen werden nicht extra behandelt.

4. Wie werden Gleitkommazahlen dargestellt?

Zahlen in Exponentialform (ein bisschen unklar) Hier fehlen noch die Gleitkommazahlen Exponent Mantisse kapieren...



float 1Bit 8Bit 23Bit
double 1Bit 11Bit 52Bit

5. Was ist die Exzessdarstellung?

Zu jeder Zahl wird eine durch die Anzahl der Bits festgelegter Exzess addiert, so dass das Ergebnis nicht negativ ist.

6. Was ist er Exponent?

$0,123 = 123 * 10^{-3} = 1230 * 10^{-4} = 12300 * 10^{-5}$ durch die verschiedene Darstellung ist es uneindeutig. Deshalb braucht man die Mantisse.

7. Was ist die Mantisse ?

Eine Zahl deren Vorkommateil 1 ist . Norm 1.xxxxxx
 Problem ist die Null. Die Zahl ergibt sich aus Mantisse * Basis ^{Exponent}
 Die Bits für die Mantisse geben die Genauigkeit an.

1.3 Logische Ausdrücke

1. Was sind Programme?

Programme sind Fkt. die binäre Informationen auf binäre Informationen abbilden.

2. Welche elementaren logischen Funktionen gibt es?

NICHT, UND, ODER NAND NOR

NICHT Symbol: \neg


einwertige Fkt. Invertierung von bits

\neg	
0	1
1	0

UND Symbol: \wedge

zwei Argumente

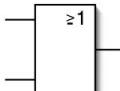
\wedge	0	1
0	0	0
1	0	0



OR Symbol: \vee

zwei Argumente

\vee	0	1
0	0	1
1	1	1



XOR Symbol: kein Symbol

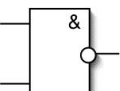
zwei Argumente

XOR	0	1
0	0	1
1	1	0

NAND Symbol: Invertierung einer AND Funktion

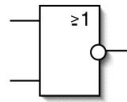
zwei Argumente

	0	1
0	1	1
1	1	0



NOR Symbol: zwei Argumente

	0	1
0	1	0
1	0	0



2 Logische Schaltungen

2.1 Grundbegriffe

Die Schaltungstechnische Realisierung von 0 und 1 durch zwei definierte Spannungspegel.
Elementare logische Funktionen siehe logische Ausdrücke.

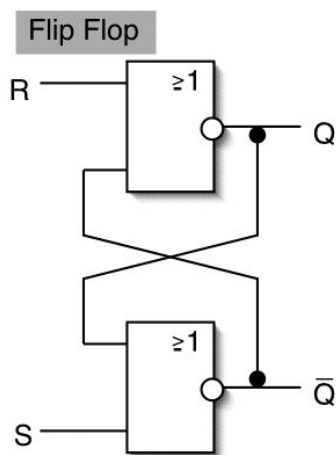
2.2 Einfache Logikschaltungen

2.2.1 Flip Flop

Rückgekoppelte Logikschaltung zur Speicherung eines Bits

Ziel: 1bit (0 oder 1)speichern

Realisierung mit zwei NOR Gattern



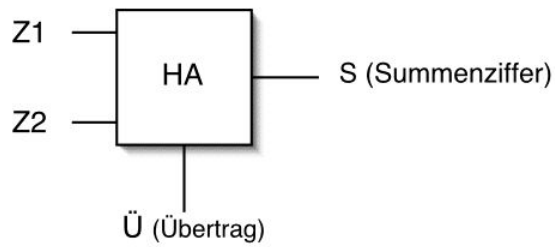
S set	R reset	Speicherinhalt Q	\bar{Q}	Zustand
1	0	1	0	setzen
0	0	1	0	speichern
0	1	0	1	rücksetzen
0	0	0	1	speichern
1	1	x	x	nicht definiert

Dieses Beispiel ist für einen Flip Flop aus NOR Verknüpfungen, man kann Flip Flops auch aus NAND Gattern bauen. Dort ist dann setzen und speichern umgekehrt. Mehrere Flip Flop parallel geschaltet ergeben eine Speicherzelle, aus diesen besteht das Register

2.2.2 Halbaddierer

HA ist Hardwareseitig

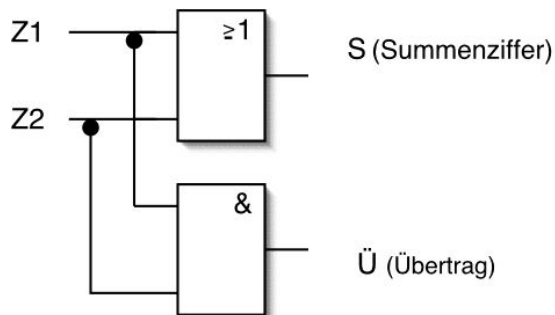
Halbaddierer



Summenziffer	0	1	Übertrag	0	1
0	0	1	0	0	0
1	1	0	1	0	1

daraus folgt folgende Schaltung im inneren des HA

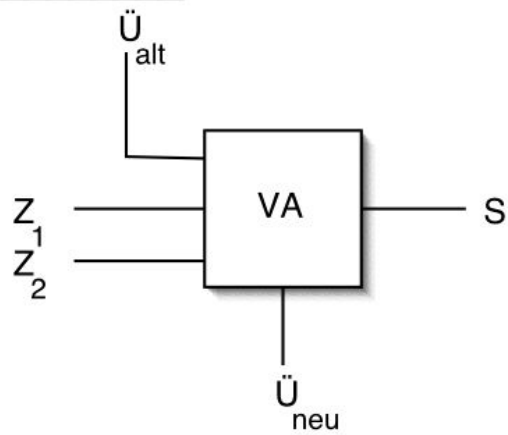
Halbaddierer



2.2.3 Volladdierer

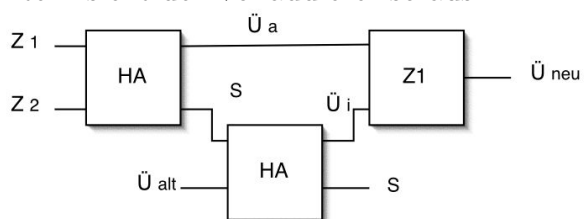
VA ist Hardwareseitig

Volladdierer



Z1	Z2	\ddot{U}_{alt}	S	\ddot{U}_{neu}
0	0	0	0	0
0	0	1	1	0
1	0	0	1	0
1	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	1	0	0	1
1	1	1	1	1

Intern sieht der Volladdierer so aus.



3 Rechneraufbau

3.1 Teile des Rechners

1. Welche Hardwareteile hat ein Rechner?

Rechner=Satz von Platinen mit elek. Bauelementen.

Hauptkarten

Prozessorkarte=Motherboard:Prozessor (CPU-central processing unit);Coprozessoren;
Hauptspeicher;Verbindungsleitungen (bus);Schnittstellenbausteine Grafikkarte: Pro-
zessor für Grafikoperationen; Speicher
Diskettenlaufwerk und oder Festplatte
Schnittstellenkarte (iec Bus)
Peripheriegeräte

2. Welche Softwareteile gibt es?

Betriebssystem:Programm für die Kommunikation des Users mit der Hardware des Rechners.

- a)Prozessverwaltung: laden und ausführen von Anwendungen
- b)Dateiverwaltung: gespeicherte Daten
- c)Strg der Peripherie: Maus, Screen...
- d)Bereitstellen von Tools
- e)Initialisierung und Test des Rechners
- f)Bediensystem (Schnittstelle zum User)

3. Wie kann man Programmiersprachen einteilen?

—>Microcode (Gatter 0/1): Maschinenbereich Befehle zur Strg. logischer Gatter und Schaltwerke innerhalb des Prozessors

—>Maschinensprache: Maschinenbereich nach außen sichtbarer Befehlssatz des Prozessors, Bit kombinationen die Sequenz von Microprogrammbefehlen entsprechen. werden hardwaremäßig innerhalb des Prozessors übersetzt

—>Assembler: Maschinenbereich maschinennahe Sprache, binäre Zeichenketten der Maschinensprache Übersetzung in kurze Namen; 1:1 Abbildung zwischen Maschinencode Assemblerkommando; übersetzungsprogramm Assembler; heutzutage nur noch für kurze, zeitkritische Programmstücke

—>Hochsprache: kompakte Befehle die Zusammenfassung über Maschinenbefehle

sein können, keine 1:1 Abbildung zwischen Maschinencode und Assemblerkommando; 4 Konzepte (Klassen) von Hochsprachen

- imperative (befehlorientierte) Sprachen Fortran, C
- objektorientierte Sprachen C++, Java, Smalltalk wechselwirkende Objekte
- funktionale Sprache Opal mathematisch orientierte Sprachen
- logische Sprache List, Prolog mathematisch orientierte Sprachen

→ natürliche Sprachen: Schriftsprache

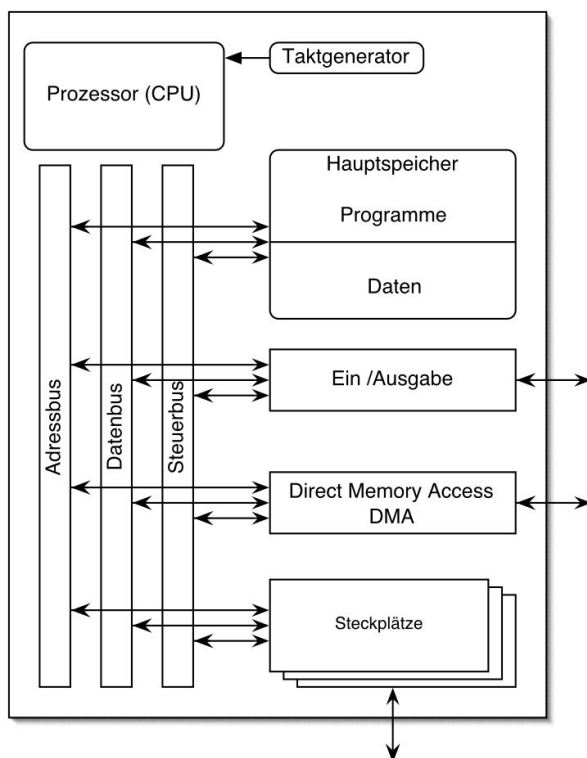
4. Was gibt es für Übersetzungsprogramme? Und wie arbeiten Sie?

→ Interpreter: sequenzielle Abarbeitung übersetzung und Ausführung (Zeile für Zeile)

→ Compiler: erst übersetzung in Maschinensprache im separaten Schritt, danach Ausführung Vorteil: hohe Lesbarkeit, strukturierte Programme, Wartungsfreundlich, portabel Nachteil: geringe Effizienz des Maschinencodes

3.2 Die Prozessorkarte

1. Wie ist der schematische Aufbau der Prozessorkarte?



CPU: dient der Datenverarbeitung

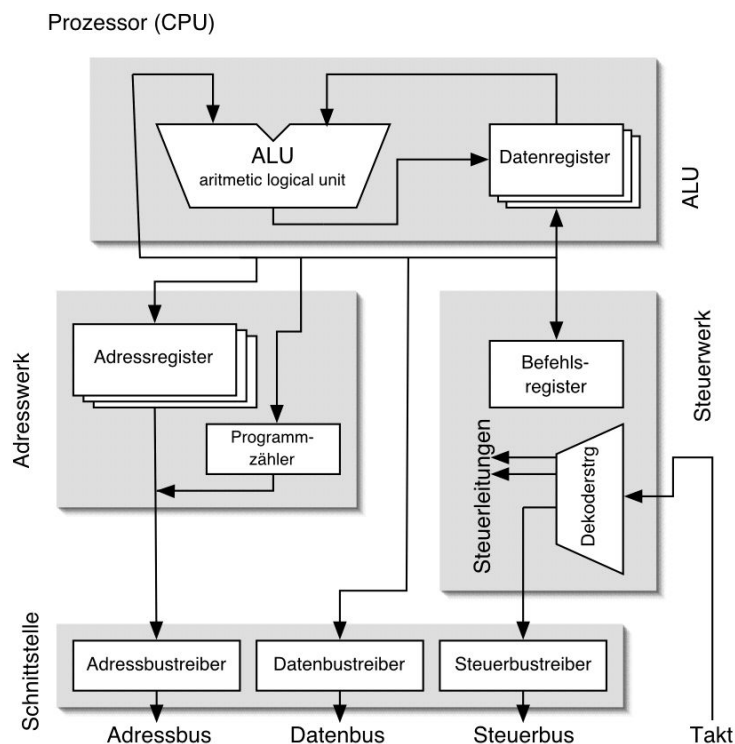
Speicher: dient der Speicherung von Daten und Programmen

Taktgenerator: synchrone Abarbeitung von Befehlen, typische Taktfrequenzen sind 70-200Mhz

I/O Einheit: Tastatur, Monitor sonstige Karten

DMA: I/O Einheit für Festplatte, Diskette

2. Welche Funktionseinheiten hat die CPU (Prozessor)?

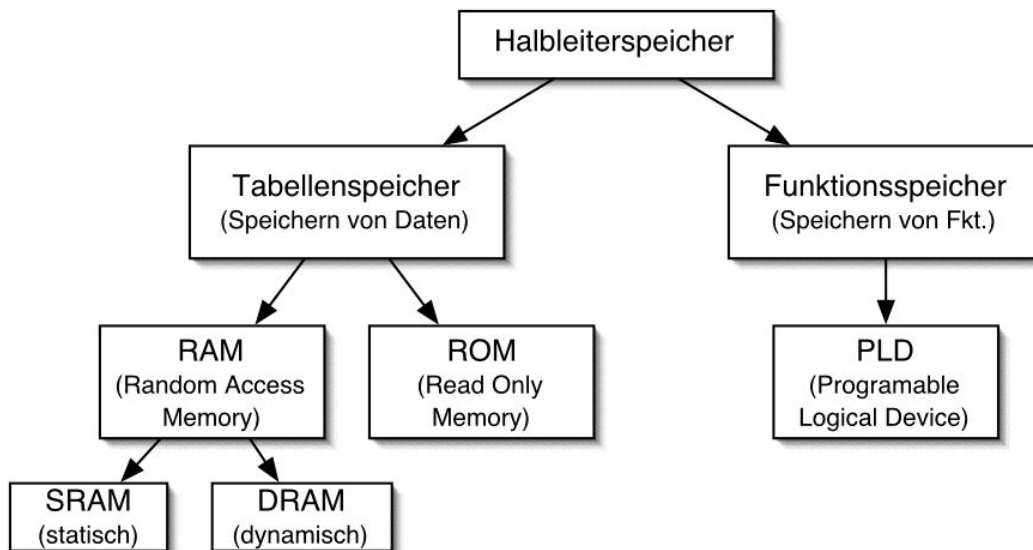


- ALU (arithmetic logical unit)+ lokaler Speicher: dient der Verarbeitung von Daten, Ausführung aller Rechenoperationen (log. Fkt., Vergleiche, arithmet. Fkt.)
- Steuerwerk: Decodierung und Umsetzung von Befehlen, Maschinenbefehl in Microbefehl übersetzt
- Adresswerk. Adressberechnung für Befehle und Daten im Hauptspeicher
- Schnittstelle: Kommunikation mit anderen Baugruppen auf dem Motherboard, Kontrolle der Busse

3. Was ist ein Register?

Speicher für einen Wert einer bestimmten Länge, meistens: 1, 8, 16, 32, 64 bits. Aufgebaut aus Flip Flops

4. Was gibt es für Speicherarten?



5. Erläutere Aufbau und Funktion von SDRAM, DRAM, ROM, EPROM ?

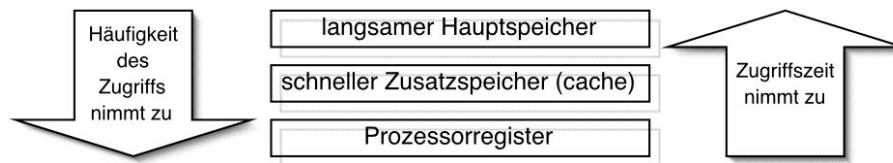
- statische RAMs (sRAM)
 - Aufbau aus Flip Flops
 - Information bleibt erhalten solange Versorgungsspannung aktiv ist.
 - keine Zugriffszeit
 - aber teuer, nicht so hoch integrierbar

- dynamische RAMs (DRAM)
 - Kondensatoren statt Flip Flop
 - Refresh Zyklus notwendig, da Ladungsverlust (alle 8ms)
 - Bitmuster geht beim Lesen verloren, lesen entspricht Lesen+ Schreiben
 - im Mittel längere Zugriffszeiten
 - aber. hohe Integrierbarkeit

- ROM
 - Festwertspeicher, Information nicht veränderbar
 - PROM, EPROM, Information kann vom User mit Schaltung eingebrannt und eventuell wieder gelöscht werden.

6. Was ist der Cache?

Der Cache ist ein schneller Zusatzspeicher aus SRAM, der den Hauptspeicher DRAM ergänzt.



7. Was gibt es für Strategien für den Cache?

alles was aus dem Hauptspeicher gelesen wird kommt in den Cache wenn cache voll ist muss man eliminieren → 2 Arten a) last recently used (löschen was lange nicht benutzt zeit) b) least frequently used (Mittelwert rest unleserlich)

8. Was ist der Prozessorbus?

ein Satz paralleler Datenleitungen (hier zwischen CPU und Hauptspeicher)

9. Was hat er für Kenngrößen?

→ Breite des Busses: Anzahl der Ltg. parallele Anzahl der gleichzeitig übertragenen Bits

→ Taktfrequenz: Anzahl der möglichen Datenübertragungen pro Sekunde

10. Was gibt es für Bussysteme?

→ Datenbus: Transport von Speicherinhalten vom/zum Prozessor (typische Breite 8,16,32 bit)

→ Adressbus: Transport von Adressen vom / zum Arbeitsspeicher, Festplatte, Ein-Ausgabe (typische Breite 16,24,32 bit)

→ Steuerbus: Verwaltungsleitungen

11. Was ist die DMA?

-Direct Memory Access

-Kommunikator mit der Peripherie

-Baustein der den Transfer von Daten ohne die CPU durchführt

-Der Transfer erfolgt dann, wenn der Prozessor den Bus nicht benutzt.

12. Welche Vor- und welche Nachteile haben Massenspeicher?

NACHTEILE

→ Sie sind langsamer als Hauptspeichersteine

→ es dauert lange beim Datentransfer

→ Sie benötigen DMA für den Transport

VORTEILE:

→ Die CPU wird nicht belastet

→ Der Transfer wird von DMA durchgeführt wenn die CPU den Bus nicht benötigt.

4 Betriebssystem UNIX

4.1 Allgemeines

1. Was ist UNIX ?

Ein C-Programm aus einer Vielzahl von Funktionen, die sich selber aufrufen.

2. Beschreibe das Schalenmodell!

Es besteht aus dem Kern, der inneren Schale und der äußeren Schale. Der Kern beinhaltet das Interface mit der Hardware, Funktionen im Kern heißen Systemroutinen. Die innere Schale enthält verschiedene Programme wie die Shell, Pascal, emacs und weitere Systemprogramme. Die äußere Schale enthält den Compiler, Matlab und eigene Anwendungsprogramme.

3. Was ist der Grund für das Schalenmodell?

Die Rechte von Usern lassen sich einfach setzen und die Software lässt sich wiederverwenden.

UNIX ist ein Mehrbenutzersystem. Das Schalenmodell ermöglicht die einmalige Bereitstellung von Programmen für alle Nutzer. Zudem können zentral die Zugriffsrechte von einzelnen Usern gesetzt werden und die Software lässt sich wiederverwenden.

4. Welche Aufgabe hat der Kern?

Der Kern verwaltet das Dateisystem (Verwaltung des Dateiraums, Steuerung des Dateizugriffs und Bearbeitung der Dateien) und steuert die Prozesse (Verwaltung der Prozesse und des Arbeitsspeichers, Kommunikation zwischen den Prozessen und deren Synchronisation, Regelung des Timesharing/Sheduling). Der Kern ist Schnittstelle für Systemaufrufe und steuert die Hardware.

5. Was ist eine Datei?

Eine Datei, oder File, ist ein Behälter für beliebige Informationen wie Programme oder Daten.

6. Was ist ein Verzeichnis?

Eine Datei mit einer festgelegten inneren Struktur, die Informationen I Node-Nummer (2 Byte) und Dateiname (14 Byte) enthält.

7. Was ist ein Prozess?

Ein in der Ausführung befindliche Kopie eines Programms.

4.2 Dateisystem

1. Woraus besteht das Dateisystem?

Das Dateisystem ist eine Struktur auf der Festplatte, Diskette, etc. und enthält drei Elemente: Die Verzeichnisdatei enthält Informationen über die Struktur des Dateisystems, gewöhnliche Dateien enthalten Programme, Texte und Daten, Spezialdateien sind angeschlossene Peripheriegeräte, die unter UNIX logisch wie Dateien behandelt werden.

2. Was ist eine Baumstruktur ?

Stellt Struktur des Verzeichnis in Form von Wurzel und Verzweigungen dar. Die Blätter an den Enden der Verzweigungen sind die Dateien. Der Weg von der Wurzel bis zu einem Blatt ist der Pfad.

3. Was ist das Arbeitsverzeichnis?

Der aktuelle Standort innerhalb der Baumstruktur. Feststellbar mit dem Befehl `pwd`

4. Was für Informationen über Dateien gibt es?

Dateityp, Zugriffsrechte, Anzahl der Links, Besitzer, Größe, Erstellungsdatum und Änderungsdatum, Name, ...

5. Beschreibe den internen Aufbau des UNIX-Filesystems!

Der Datenträger ist in Physikalische Blöcke unterteilt (meistens 512 Byte groß); das Filesystem fasst diese zu logischen Blöcken zusammen (meistens 2, 8 oder 16 physikalische Blöcke). Die logischen Blöcke sind in Bootblock (Code zum laden des Betriebssystems), Superblock (Größe der Blöcke, Anzahl und Liste der freien Blöcke, Größe der Inode-Liste, Anzahl und Liste der leeren Inodes), Inode-Liste (Inode-Nummer und Informationen jeder Datei, ohne deren Name) und den Dateienblöcken (können über den gesamten Datenträger verteilt sein) aufgeteilt.

6. Welche Dateitypen gibt es ?

Unter Unix ist alles (außer die I nodes einheiten) eine Datei.

DATEITYPEN:

—>gewöhnliche Dateien (ordinary file) Nutzdaten(Texte, tabellen bilder, Datenbanken, töne, programme usw..)

—>Verzeichnis (directory) dient der Org. aller dateien (liste von dateinamen)

- >Geräte-datei (devices) regelt Zugriff auf Hardware
- >Link Verweis auf eine Datei
- >Named Pipe dient dem Datenaustausch zwischen Programmen (zwischen Prozessen) in einem UNIX System;heißt auch FIFO first in first out (arbeitsweise)
- >Socket erlaubt den Austausch von Daten zwischen Programmen unterschiedlicher Plattformen

Named Pipe und Socket sind eigentlich eher für das Programmieren interessant

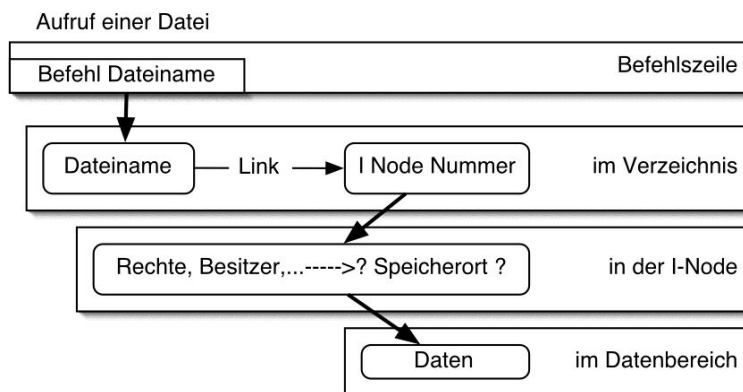
7. Wie funktioniert der Zugriff auf eine Datei?

Als Beispiel soll mit dem Befehl `cat .profile` (Befehlszeile) etwas gezeigt werden

- 1) Unix prüft ob die Datei existiert (Verzeichnis)
- 2) ist es der Fall wird der link dazu benutzt die i node nummer zu bestimmen (Verzeichnis)
- 3) über die nummer wird der Inhalt (Rechte Besitzer) der i node nummer ermittelt. (n der I node)
- 4) ist der zugriff erlaubt werden durch die ebenfalls in der I node vorhandenen Angaben zum physikalischen Speicherort, die Daten auf den Bildschirm ausgelesen (in der I node und dem Datenbereich)

Dateiverwaltung erfolgt in drei Schritten

- 1 in den Verzeichnissen Übersetzung des Dateiname in I node nummer
- 2 in der I node Prüfen der Zugangsberechtigung und Ermittlung des Speicherortes
- 3 Inhalt der Datei



8. Welche Informationen sind in der I-node?

- Dateityp
- Rechtetabelle
- Anzahl der links
- Dateibesitzer
- Besitzergruppe

Größe in bytes
Datum letzter Änderung
Datum letzter Benutzung
Datum letzter Änderung der I Node
physikalischer Ort auf Datenträger an dem sich Daten befinden

9. Was ist ein Link?

Ein Link ist ein Eintrag in ein Verzeichnis er besteht aus dem Dateinamen und der I Node Nummer Man erzeugt einen Link mit dem Befehl `ln`.

Das Dateisystem erlaubt mehrere Dateinamen, da über Links eine eindeutige Zuordnung erfolgt. Die eindeutige Zuordnung besteht zwischen I Node nummer und Datei. Durch den Link erfolgt eine Zuordnung name zu I Node nummer

Beispiel:

```
[localhost: /Desktop] louis% mkdir test (Verzeichniss erstellen)
[localhost: /Desktop] louis% ls -id test (I Node nummer anzeigen)
544627 test
[localhost: /Desktop] louis% ls -ia test (alle zeigen lassen)
544627 . (I Node nr des Arbeitsverzeichnisses )
127168 .. (I Node nr des Elternverzeichnisses)
```

wie man sieht haben test und . dieselbe I node da sie auf dieselben Daten zugreifen

noch ein beispiel

```
[localhost: /Desktop] louis% cd test (wechsel zu test)
[localhost: /Desktop/test] louis% mkdir unterverzeichnis(neues unterver-
zeichnis anlegen)
[localhost: /Desktop/test] louis% ls -id ( I Node von Test =aktuelles Ar-
beitverzeichnis)
544627 .
[localhost: /Desktop/test] louis% ls -ia unterverzeichnis/
544629 . (I Node von Unterverzeichnis und dessen
544627 .. Elternverzeichnisses)
```

10. Welchen Zweck haben INODES?

Informationen über eine Datei werden getrennt von dieser in der I Node abgelegt. Zu jeder Datei gehört genau eine I Node. I Nodes kann man mit Informationsknoten übersetzen; sie dienen der Dateiverwaltung. In der I Nodeliste steht die Startadresse und die Größe des Datenbereiches zu jeder Datei.

11. Welche Dateioperationen im Hinblick auf INODES gibt es?

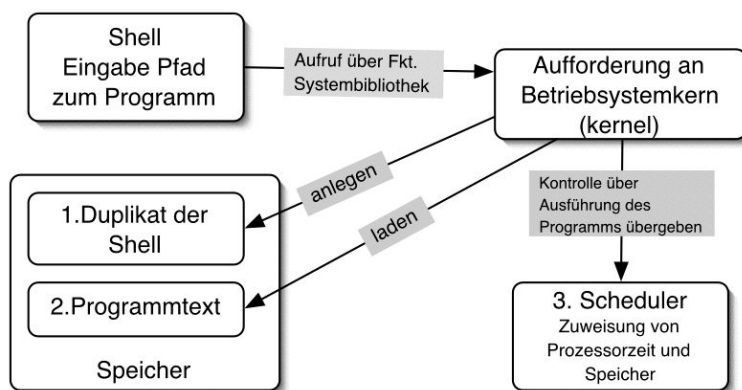
alle I Node Nummern des Wurzelverzeichnisses anzeigen `ls -ia /`

4.3 Prozeßsteuerung

1. Was ist ein Prozess und wie ist er aufgebaut?

Kopie eines Programmes bei der Ausführung in Form eines logischen Bereiches im Hauptspeicher. Dieser muss wie Dateien nicht notwendigerweise physikalisch zusammenhängen. Ein Prozess besteht aus Text (Maschinenbefehle) und Daten des übersetzten Programms und dem Stack. Der Stack wird von dem System verwaltet und enthält pipes, Signals, lokale Variablen und andere für Unterprogramm-Aufrufe benötigte Prozesse. Der Text und Datenbereich gibt und empfängt Systemaufrufe an / aus dem UNIX Kern. Zusätzlich wird ‚Buchhaltung‘ für den Kern geführt in Form von Einträgen in die Prozeßliste und den u-Datenbereich.

2. Wie ist das Ablaufschema der Initiierung eines Prozesses?



3. Was ist Multitasking?

Die gleichzeitige Bearbeitung von mehreren Prozessen, bei denen aber immer nur ein Prozess die CPU für sich hat (ausgenommen Parallelrechner), daher gibt es unterschiedliche Prozeßzustände.

4. Wofür braucht man Multitasking?

Die CPU ist schneller als der Benutzer und die Peripheriegeräte (1 Zugriff vs. 1000 Instruktionen), so können Wartezeiten genutzt werden. Zudem kann so der Zugang für viele Benutzer gleichzeitig geregelt werden.

5. Welche Prozeßzustände gibt es?

created

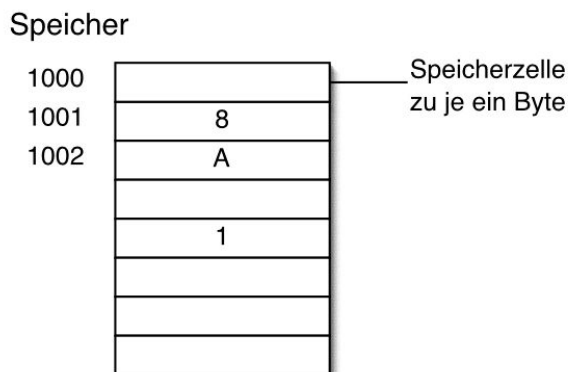
ready to run – swapped (ausgelagert, muss wieder in den Hauptspeicher geladen werden)

ready to run - in memory (kann durch den Scheduler der CPU zugeteilt werden)

user running (Zugriff nur auf den Speicherbereich des Prozesses)

- kernel running (Zugriff auf die Datenstrukturen → Systemroutinen)
- preempted (Zeitscheibe abgelaufen, verdrängt durch den Kern)
- asleep – swapped (Prozess wartet auf bestimmtes Ereignis z.B. I/O)
- asleep – in memory
- zombie (Beendet aber noch nicht dem Elterprozeß mitgeteilt)

6. Beschreibe die Speicherverwaltung!



Die Speicherverwaltung verwaltet den zur Verfügung stehenden Hauptspeicher und ordnet ihn den zu verarbeitenden Tasks zu.

Grundsätzlich sind 2 verschiedene Techniken zu unterscheiden. Bei der realen Speicherverwaltung wird nur der physikalisch vorhandene Hauptspeicher benutzt. Eventuell werden komplette Programme vom Hintergrundspeicher ein- bzw. auf diesen ausgelagert, um die Anzahl der bereiten Prozesse zu erhöhen. Dieser Vorgang wird als Swapping bezeichnet. Die Programme können jedoch nicht größer als der zur Verfügung stehende Hauptspeicher sein. Die virtuelle Speicherverwaltung dehnt den Adreßraum über den realen Speicher hinaus aus, indem zusätzlicher Speicherraum auf Hintergrundspeicher zur Verfügung gestellt wird.

7. Beschreibe das Konzept der virtuellen Adressen!

Die virtuelle Speicherverwaltung bietet eine bessere Ausnutzung von Hauptspeicher und Prozessor. Weiterhin können die Programme sehr viel größer sein als der physikalische Speicher. Ihre Größe ist nur durch den logischen Adreßraum begrenzt. In Kauf genommen werden diese Vorteile durch einen höheren Verwaltungsaufwand und eine etwas längere Verweilzeit der Programme. Es muß bedacht werden, daß der direkte Zugriff auf den Hauptspeicher etwa 1000 bis 10000 mal schneller erfolgt als der Zugriff auf den Hintergrundspeicher (z.B. Festplatte). Problematisch sind die Sprünge in den Programmen, denn nach dem Kompilieren ist noch unklar an welche Stelle im Arbeitsspeicher das Programm geladen werden wird. Deshalb werden virtuelle Adressen benötigt.

Jeder Prozeß hat seinen eigenen Adressraum. Da nicht klar ist welcher teil des

Programms wann benötigt wird, wird der Text- und Datenbereich des Prozesses in Seiten unterteilt. Die Seiten haben in der Regel einen Offset von 512 - 4000 Bytes. Eine Memory Management Unit sucht bei Bedarf die benötigte Seite im physikalischen Speicher anhand der Seitennummer und dem Seitenoffset (siehe Regionen).

8. Was ist ein OFFSET?

- Der Offset gibt an, welche Position der erste Block in dem gesamten File hat.
- Der Abstand in Bytes zwischen dem Beginn einer Datenstruktur und dem Beginn eines bestimmten Feldes in einer Datenstruktur. Die Anzahl der Bytes von einem Bezugspunkt (normalerweise dem Beginn einer Datei) bis zu einem bestimmten Byte. Kennzeichnet man eine Dateiposition mit einem Offset, beginnt die Zählung beim ersten Byte der Datei mit 0.

9. Was sind Regionen?

Regionen dienen der Zuordnung der virtuellen Adressen. Zu jedem Prozeß gibt es eine Prozeßregionentabelle, sie liegt, implementationsabhängig im User-Bereich oder in der Prozeßtabelle im Kernel. Sie enthält für jede Region einen Eintrag mit der Adresse auf der Regionentabelle. Die Regionentabelle gehört zum Kern und hat ebenfalls einen Eintrag für jede Region, der auf die logische ODER ??? physikalische Seiten-Nummer der Seitentabelle verweist. Die Seitentabellen werden in den Registern der CPU gespeichert, was einen extrem schnellen Zugriff ermöglicht.

10. Beschreibe die Prozeßtabelle!

Die Prozeßtabelle gehört zum Kernel und liegt im Hauptspeicher. Sie enthält:

- Sheduling Parameter (Priorität, CPU-Zeit, Blockzeit im letzten Quantum)
- Das Speicherabbild (Zeiger auf Daten, Stackregionen, Seitentabellen, externe Adressen)
- Signale (Welche Signale werden ignoriert, temporär ausgesperrt, von Verarbeitern behandelt oder gerade ausgesperrt)
- Statusanzeigen (PID, Elter-PID, UserID, GroupID, Zeit bis zum nächsten Alarm, Ereignis auf das gewartet wird)

11. Erläutere das Prozeß-Scheduling!

Sheduling sind auf Prioritäten basierende Regeln für die Zuweisung von CPU-Zeit an die einzelnen Prozesse. Jeder Prozeß erhält eine Zeitscheibe, nach Ablauf der Zeit wird der Prozeß verdrängt und erhält eine Priorität, die in der Prozeßtabelle eingetragen wird. Der Prozeß mit der höchsten Priorität wird ausgewählt; alle Prioritäten werden neu berechnet.

Die Aufgaben der Prozeßverwaltung sind:

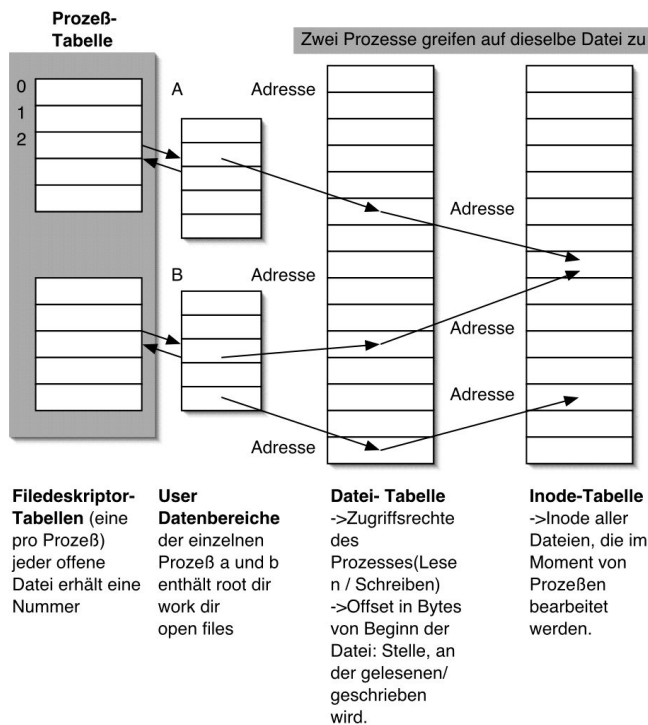
→ Aufbau und Verwaltung einer Warteschlange (Queue) der bereiten Prozesse

- Zuteilung der Betriebsmittel (wie z.B. Speicherplatz, Peripheriegeräte, Dienstprogramme) an den aktiven Prozeß
- Unterbrechungsbehandlung (interrupt handling).

Die Bestimmung der Prozeßreihenfolge durch die Einordnung in die Warteschlange der bereiten Prozesse und die Zuteilung und Freigabe der Betriebsmittel übernimmt der Scheduler. Anlässe zu einer Neubestimmung der Prozeßreihenfolge sind Unterbrechungen, die aus dem Prozeß selber resultieren können oder von außen verursacht werden, wie der Ablauf einer Zeitscheibe. Beim Zeitscheibenbetrieb kann ein Prozeß vom Zustand "aktiv" direkt in den Zustand "bereit" wechseln.

12. Beschreibe den Zugriff auf das Dateisystem!

Der Kernel unterhält drei Tabellen im Arbeitsspeicher



13. Wo liegt der u-DATENBEREICH ?

Der User-Datenbereich scheint ein Teil des Datenbereiches von Prozessen zu sein. Im u-Bereich stehen die offenen Dateien sowie das Root- und das Arbeitsverzeichnis. Er 'vermittelt' zwischen Prozesstabelle sowie der Datei- und Inodetabelle.

14. Wozu dienen die Befehle exec und fork?

Prozesse können selbst wieder neue Prozesse erzeugen. Dazu stehen zwei Möglichkeiten in Form von Systemaufrufen zur Verfügung: `exec` und `fork`.

Generiert ein Prozess mit `exec` einen neuen Prozess überschreibt er sich selber mit dem neuen Prozess und wird aus dem Speicher entfernt.

Wird hingegen `fork` benutzt, so erzeugt der aufrufende Prozess zunächst eine Kopie seiner Systemumgebung und lädt dann den neu zu erzeugenden Prozess nach.

15. Wozu dienen Prioritäten und welche gibt es?

Die Zuteilung von Prozessor und Speicherressourcen übernimmt der Scheduler. Dazu wird jedem Prozess eine Priorität zugeordnet. Es gibt zwei Ebenen des Scheduling; high-level (Systemprioritäten) und low-level (Benutzerprioritäten) die durch einen Schwellenwert von 60 getrennt sind. Die low level Prioritäten wird aus dem nice value berechnet. Die Priorität eines Prozesses lässt sich mit `nice` verändern. Siehe dazu 4.5 wichtige Unixbefehle.

4.4 UNIX-Shells

1. Was ist die Shell?

Die Shell ist ein Kommandointerpreter der die Kommunikation mit dem Kern realisiert. Sie wird beim Einloggen gestartet und meldet sich mit Prompt Aufforderung. Aufgaben der Shell:

- eine Programmiersprache in Form von Shellskripten (lauffähiges Programm aus UNIX Kommandos)

- einen Systemaufruf absetzen

2. Was ist ein Shell-Skript?

Ein lauffähiges Programm aus UNIX Kommandos. Die Shellskripte sind in der Interpretersprache der Shell geschrieben. Das heißt, sie werden nacheinander ausgeführt. Es gibt mehrere Dialekte je nach Shelltyp (`bash`, `cshell`, `Korn`) !Achtung : ein Shellskript muss immer ausführbar sein `chmod u+x;dateiname;`

Es gibt nur einen Datentyp!

3. Was für unterschiedliche Zeichen gibt es?

Die shell überprüft nicht die Argumente (Parameter), die einem Kommando übergeben werden. Dies ist Aufgabe des Kommandos selber. Es werden jedoch Metazeichen für Dateinamen 'expandiert'. Es gibt Metazeichen (haben eine Sonderbedeutung), Trennzeichen (Blank, Tab), alphanumerische Strings und Wörter (Folge von Zeichen, abgegrenzt durch ; & () — <>Blant Tab und CR).

4. Was für unterschiedliche Kommandos gibt es?

shell-Kommandos können in einer Datei stehen (shell-Skript). Sie werden von einer neuen shell durch das Kommando `sh` ausgeführt: `sh datei` oder nur `datei`. Dabei muß die Datei "datei" ausführbar sein (Zugriffsrecht `x`). Vielfach findet man zu Beginn eines shell-Skripts die Anweisung `#!/bin/sh`. Damit legt man das Programm (in unserem Fall den Kommandointerpreter) fest, das diese Datei abarbeiten soll. Man kann dann z.B. eine Bourne-shell-Prozedur von einem anderen Kommandointerpreter aus starten.

5. Was für unterschiedliche Kommandos zur Umlenkung gibt es?

`>` → Datenumleitung der Standardausgabe (Bildschirm) in bestimmte Datei. Hierbei wird der Inhalt überschrieben.

Bsp:

```
ls -l > umleitung
```

`>>` → Datenumleitung der Standardausgabe (Bildschirm) in bestimmte Datei. Hierbei wird der Inhalt angehängt.

`<` → Datenumleitung der Standardeingabe (Tastatur) aus bestimmter Datei. Hierbei wird der entsprechende Befehl aus der Datei geholt und nicht vom Prompt.

Bsp:

Wir haben die Dateien `kap1-3` und wollen Sie löschen. Löschbefehle (`y n` Antworten auf `rm`) in Datei `delete` anlegen

```
rm kap?<delete
```

6. Was ist eine Pipeline?

Eine Folge von Kommandos die durch das Pipesymbol `|` oder durch `|&` voneinander getrennt sind. Generell wird dabei die Ausgabe eines Programmes in die Eingabe eines anderen Programmes umgelenkt.

Bsp:

```
lprog|rprog
```

7. Was sind Listen?

Eine Folge von Kommandos oder Pipelines die getrennt sind durch `;` `&` `&&` `||`

`;` → hintereinander ausführen

`&` → hintereinander ausführen

`&&` → rechtes Kommando wird nur ausgeführt, wenn die Pipeline erfolgreich abgeschlossen ist (Rückgabewert=0)

`||` → dasselbe bloss nicht erfolgreich

8. Erkläre den Unterschied zwischen Variablen und Parametern!

Im UNIX-Kommandointerpreter gibt es nur Variable vom Typ Zeichenkette. Sie erfordern keine Deklaration und sind mit ihrem ersten Auftreten bekannt.

Variablenamen bestehen aus Buchstaben, Ziffern und dem Unterstreichungszeichen. Das 1. Zeichen muß ein Buchstabe sein. Die Zeichenketten, die als Werte einer Variablen zugewiesen werden, sollten in Anführungsstrichen stehen. In den Zeichenketten werden die Sonderzeichen \$, ' und umgedrehter Backslash von der shell speziell interpretiert. Es hat sich die Konvention eingebürgert, shell-Variablen in Großbuchstaben zu schreiben. Mit dem Gleichheitszeichen weist man einer Variablen einen Wert zu.

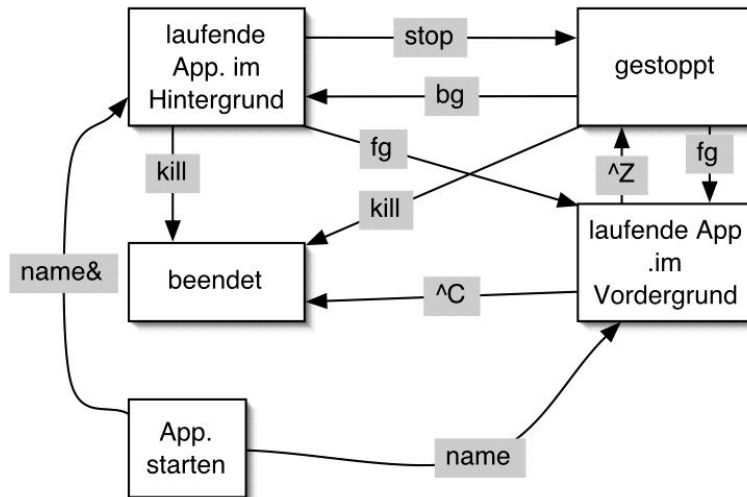
Parameter sind ein Spezialfall vordefinierter, automatischer Shellvariablen. Mit ihnen kann man Zeichenketten an ein Shellscript übergeben

Bsp.: `% mayscript sascha babak jani louis` wird im Skript an `$1 $2 $3` und `$4` übergeben.

9. Was für Operatoren gibt es?

Substitutionsoperatoren (...???), Stringoperatoren (beliebige Zeichenketten), Numerische Operatoren (nur numerische Zeichenketten), @-Kommando (Zuweisung des Ergebnisses an eine andere Variable).

10. Welche Kommandos zur Ablaufsteuerung gibt es?



top Anzeige aller laufenden Prozesse

ps Anzeige der Prozesse die von dem Shellfenster aus gestartet wurden

11. Was gibt es für Jobarten?

Vordergrund: Shell wartet bis der Prozess zu Ende ist

Hintergrund: Shell läuft parallel zum Prozess

Job gestoppt: Bearbeitung des Prozesses ist gestoppt

Job der über Shell gestartet wird hat

a) Prozess ID=Identifikationsnummer auf CPU;

b) Jobnummer=Prozess in dieser Shell

4.4.1 Shellprogrammierung

12. Welche Kommandos zur Ablaufsteuerung gibt es (in der csh)?

Es gibt die Kontrollflußanweisungen *if* und *switch* (bedingte Sprünge) sowie *while*, *foreach* und *repeat* (Schleifen) des Kommandointerpreters.

Kontrollflußanweisung if

```

if (Ausdruck) then
thenKOMMANDO
else
  
```

```
elseKOMMANDO
endif
```

Die Angaben müssen wie gezeigt jeweils in einer Zeile vorgenommen werden. Jedes Kommando liefert nach seiner Ausführung einen Wert zurück, den sogenannten exit-Status. Der Wert 0 ist dabei gleichbedeutend mit TRUE bzw. fehlerfrei. Werte ungleich 0 werden in der Regel benutzt, um den fehlerhaften Ablauf des Kommandos zu signalisieren und einen Hinweis auf die Fehlerursache zu geben.

Kontrollflußanweisung switch

```
switch (Wort)
case pattern1:
KOMMANDO1
breaksw
case pattern2:
KOMMANDO2
breaksw
default:
KOMMANDOn
endsw
```

Entspricht der Wert der shell-Variablen Wort der Zeichenkette pattern1 wird KOMMANDO1 ausgeführt. Entspricht sie pattern2, wird KOMMANDO2 ausgeführt. Entspricht sie keinem Muster, wird KOMMANDOn ausgeführt. Bei den Mustern können auch Metazeichen verwendet werden.

Kontrollflußanweisung for

```
foreach Variable (Wortliste)
KOMMANDO
end
```

Die Schleife wird durchlaufen, wobei die shell-Variable jeden Wert aus Wortliste annimmt. Ein Beispiel für Wortliste ist eine via Befehlssubstitution gewonnene Liste von Dateinamen ('ls'). Man kann auch explizit eine Liste angeben. Sie ist dann in runde Klammern einzuschließen: (datei1 datei2). Fehlt "Wortliste", so werden die Argumentwerte 1,2,... genommen. Die Wörter der Wortliste müssen durch 'blank' getrennt werden.

Kontrollflußanweisung while

```
while (Ausdruck)
KOMMANDO
end
```

Die Schleife wird durchlaufen bis KOMMANDO einen Wert ungleich 0 zurückliefert.

Kontrollflußanweisung repeat

repeat n KOMMANDO

Die Schleife wird durchlaufen bis KOMMANDO den Wert 0 zurückliefert. KOMMANDO darf nur einzellig sein, ohne Pipelines oder Listen.

4.5 wichtige Unixbefehle

Befehl	Option	Bedeutung
chmod		ändert Dateirechte
cp		Kopiert Datei
ls	-A -a -l -i -t -n	Befehl listet Inhalt von Verzeichnissen auf zeigt alle Dateien ausser die Unsichtbaren zeigt alle Dateien listet Dateiformationen ausführlich auf Anzeige der I-Node- Nummer der Datei Ausgabe sortierte Anzeige nach der letzten Benutzung Ausführliche Information . Statt Namen für die Besitzer
ln		Befehl erzeugt Link
nice		dient zur Angabe eines Wertes zwischen 1-19 um die Priorität eines Prozesses zu bestimmen. Standardwert ist 20. Je grösser der nice Wert desto niedriger ist die Priorität eines Prozesses.
pwd		print working directory, zeigt den aktuellen Pfad an
rm	files	löscht Files
rmdir	Pfadname	löscht leere Verzeichnisse vom Pfadnamen
cat	files	Zeigt Inhalt von Dateien an
exit		beendet diese Shell
mkdir	Pfadname	legt neues Verzeichnis an
mv	old new	bewegt oder nennt File um alt zu neu
man		zeigt die manual pages
gcc cc	-c -o datei	Compileraufruf ohne zu linken Leitet die Ausgabe in die angegebene Datei um. Diese Option macht aber nur Sinn wenn man gleichzeitig linkt oder nicht mehr als eine Datei übersetzen möchte. Ist diese Option nicht angegeben erzeugt gcc eine Datei mit dem Namen a.out.

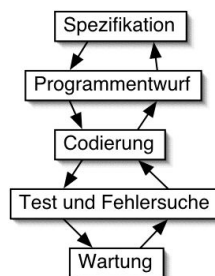
5 Die Programmiersprache C

5.1 Allgemeines

1. Was ist ein Programm?

vollständige Anweisung zum Lösen einer Aufgabe mit allen erforderlichen Vereinbarungen.

2. Wie stellt man Vollständigkeit und Effizienz sicher?



3. Was gehört zu einer Spezifikation?

- Was soll das Programm leisten? Benutzerabhängigkeit
- Welche Hard und Softwareumgebung wird gebraucht?
- Ist es kompatibel (Antwortzeiten, Verfügbarkeit, mögliche Erweiterbarkeit)?

4. Was ist Modularisierung und wozu dient Sie?

- Grund für Modularisierung ist eine erhöhte Lesbarkeit und Wartungsfreundlichkeit
- M.verringert Redundanz im Code
- Fkt. gruppieren
- Isolierung von Hard und Systemabhängigen Code
- Isolierung von Ein und Ausgabe

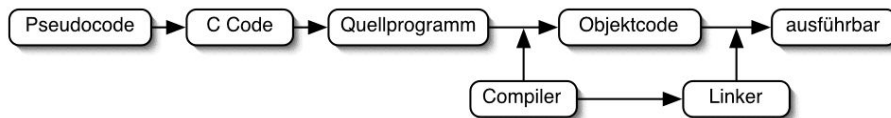
5. Was gehört zum Programmwurf?

- schrittweise Verkleinerung
- man beginnt mit der höchsten und abstraktesten Ebene
- vermeide sprachspezifische Details
- Füge Details so schnell wie möglich ein

6. Was ist der Pseudocode?

Das Resultat der schrittweisen Verkleinerung ist der PSEUDOCODE eine Zeile PSEUDOCODE entspricht wenigen Zeilen C

7. Wie sehen die Schritte der Kodierung aus?



8. Wie sieht der Test und die Fehlersuche beim Compiler und Linker aus?

—> Compiler: Fehlermeldung bezüglich der Syntax

—> Linker: Fehlermeldung falls referenzierte Objekte nicht gefunden werden

Bei der Ausführung gibt es zwei Fehler Core Dumps und falsches Ergebnis. Ein Hilfsprogramm für die Fehlersuche sind sogenannte DEBUGGER.

9. Was gehört zur Wartung?

Dokumentation

1) Beschreibt die Fkt. der wichtigsten Zeichen im Text (code)

2) Aufgabenstellung und Spezifikation, Vereinbarungen über besondere Ein- und Ausgabeformate

3) Testbericht: Beispiellauf

Informationen bundling —> Infos in ein Bündel / Modul zusammenfassen

5.2 Übersicht über die Programmiersprache

1. Was sind Funktionen?

Funktionen entsprechen einem Satz von C Ausdrücken in einer „Verpackung“. Fkt. können im Quelltext auf verschiedene Dateien verteilt sein. Sie werden im lauffähigen Programm hintereinander gebunden (LINKER)

2. Was ist ein C Programm?

Eine Menge von Fkt., die sich gegenseitig aufrufen.

3. Was ist die main Fkt?

Jedes lauffähige C Programm braucht eine main Fkt.. Diese wird beim Start des Programms vom Unix Kern aufgerufen! Nicht von anderen Fkt!

4. Wie ist eine C Funktion aufgebaut?

```
Funktionsstyp(Rückgabewert) name (Argumente mit Deklaration)
{
```

-Deklaration lokaler Variablen, diese sind dann nur innerhalb der Fkt
. bekannt
-Code- C Ausdrücke
}

Beispiel für eine Quadrierungsfunktion

```
int sqare (int num)
{
int answer;
answer=num*num;
return answer;
}
```

5. Was ist eine Konstante in C?

Ein Datenobjekt mit einem alphanumerischen Wert, das Objekt kann eine Zahl , eine Zeichenkette oder ein Zeichen sein. Es ist konstant im Gegensatz zu Variablen sie sind variabel. es wird auch kein Speicher reserviert sondern als Direktoperanten der Maschinenbefehle.

AUSNAHME STRINGKONSTANTEN diese werden vom Compiler in einem eigenen Speicherbereich (Konstantenbereich) gespeichert.

-Zeichenkonstanten werden in Hochkommata eingeschlossen Bsp.: '7' 'A' 'B'

-Zeichenkettenkonstanten werden in Anführungszeichen gesetzt Bsp. "Bitte Zahl eingeben"

-Zahlen bsp: 5, -4, 300, 150., 15.E1,

6. Was ist eine Variable?

Eine Variable bezeichnet einen Hauptspeicherbereich, in dem man zu unterschiedlichen Zeitpunkten beim Programmablauf unteschiedliche Werte speichern kann. eine Variable hat:

- 1)einen Namen über den sie im Programm angesprochen werden kann
- 2)eine bestimmte Speicherlänge
- 3) ein bestimmtes internes Format, in dem der Wert gespeichert wird

2+3 ergeben DATENTYP

Bsp.

`int s;` → Compiler reserviert Speicher aufgrund des Datentyps

`s=0;` → Zuweisung Wert 0 wird in der Variable gespeichert

7. Was ist der Unterschied zwischen globalen und lokalen Variablen?

globale Variablen werden außerhalb von Funktionen definiert und gelten im ganzen Programm lokale Variablen werden innerhalb einer Fkt. definiert, sie gelten nur innerhalb der Funktion

8. Was sind Ausdrücke?

Kombination von Konstanten Variablen und Operatoren

9. Was ist ein guter Programmstil?

-viele newlines und space -eine Zeile pro ausdrück -einrücken geschachtelte Texte
Kommentare setzen `\ **\` mehrzeilig `\\` einzeilig

10. Was ist der Präprozessor?

Der Präprozessor ist ein separates Programm das vom Compiler aufgerufen wird und Quellcode editiert bevor übersetzt wird

→ Präprozessordirektiven beginnen mit `#`

→ Präprozessordirektiven enden mit newline

2 Eigenschaften des PP werden häufig genutzt

1) `#include` → inhalt der Datei wird während der Übersetzung eingefügt zb headerdatei

2) `#define` → um einen Namen durch eine beliebige Folge von Symbolen zu ersetzen

11. Was sind Header Dateien?

Datei die definitionen und deklarationen, Fkt. zentralisiert enden auf `.h` wird in eckige Klammern gesetzt beispiel `#include <stdio.h>` Definition für eigenene Dateien `#include "myfile.h"`

12. Wie rufe ich den Compiler auf?

```
% cc -o quad main.c square.c
```

→ Übersetzen der Dateien

→ Linken der Dateien main.c und square.c

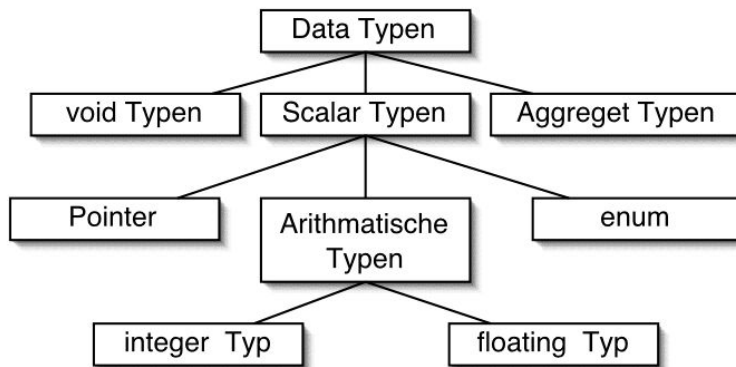
→ Speichern der Datei unter dem Namen quad

5.3 Skalare Datentypen

1. Was ist ein Datentyp?

Interpretationsvorschrift für binäre Zeichenketten (in Speicherzellen)

2. Was gibt es für Datentypen?



elementare arithmetische Datentypen:

char ein Byte, kann ein zeichen aus dem zeichensatz der Maschine aufnehmen
 int ein ganzzahliger wert, üblicherweise in der für die Maschine „natürliche“ Größe
 float ein einfacher genauer Gleitpunktwert
 double ein doppelt genauer Gleitpunktwert

Varianten in denen int auch weggelassen werden kann

short int sh;

long int counter;

signed und unsigned können auf char oder integer-Typen angewandt werden.

Typ	Grösse	Wertebereich
char	1	$-2^7 \text{ bis } 2^7 - 1$
short int	2	$-2^{15} \text{ bis } 2^{15} - 1$
int	4	$-2^{31} \text{ bis } 2^{31} - 1$
unsigned char	1	$0 \text{ bis } 2^8 - 1$
unsigned shortint	2	$0 \text{ bis } 2^{16} - 1$
unsigned int	4	$0 \text{ bis } 2^{22} - 1$

Typ	Grösse
float	4
double	8

3. Was bedeutet signed und unsigned?

Mit 'unsigned' weisen sie den Compiler an das er eine Datentyp Vorzeichenlos behandeln soll. Dadurch erhöht sich auch der Wertebereich der positiven Zahlen. Mit 'signed' machen sie genau das Gegenteil. Aber 'signed' ist bei allen Variablentypen voreingestellt.

Wenn sie nun das Schlüsselwort unsigned vor einen Datentyp stellen sind keine negativen Zahlen mehr möglich. Somit verdoppelt sich aber der positive Wert der Zahl. Sollten sie z.B. 2 Zahlen subtrahieren und es kommt ein negatives Ergebnis heraus und sie haben vor Ihrem Typen des Ergebnisses das Schlüsselwort unsigned

gesetzt, so wird das Minus der negativen Zahl ignoriert und es wird eine positive Zahl daraus.

4. Wozu dient der Datentyp enum?

5. Was sind aggregierte Datentypen?

Datentypen, die aus den Einzelfeldtypen aufgebaut werden. Bei den aggregierten Datentypen unterscheiden wir zwischen Strukturen (auch Feldleisten genannt) und internen Tabellen. Auf Datenobjekte aggregierter Datentypen kann man insgesamt oder auf einzelne Felder zugreifen.

6. Wozu dient der Datentyp void?

-Diese Zeichenkette wird nicht benötigt deshalb wird sie nicht interpretiert

-für Funktionen benötigt die nichts zurückliefern

-Void grÖÙe 0 Byte

Beispiele:

```
void func(int a, int b){...}
```

```
int func (void){..}
```

-void wird auch für die Typisierung von Pointern verwendet

-void darf nicht in einen anderen Typ umgewandelt werden

-aber mit einem Umwandlungsoperator kann ein Ausdruck in den Typ void umgewandelt werden

7. Was ist der Unterschied zwischen Deklaration und Initialisierung von Variablen?

DEKLARATION Variablen müssen deklariert werden bevor man sie verwenden kann, das heißt, Speicherzellen müssen reserviert werden, Der Name und die virtuelle Adresse müssen miteinander verknüpft werden, der Compiler muss wissen wie die binäre Zeichenkette interpretiert werden soll

—>Syntax Deklaration `unsigned int j, num, xc;`

Datentyp Variablenname durch Komma getrennt mit Semikolon abschluss

Dekalrationen zu Beginn des Blockes wo Sie verwendet werden

Deklarierte Variablen enthalten erst dann Werte wenn Sie initialisiert werden

INITIALISIERUNG ist die erstmalige Belegung einer Variablen, Speicherzelle

Es gibt zwei Arten der Initialisierung: a)bei der Deklaration Bsp: `int a=2;` b)initialisierung durch Zuweisung `double a; a=27,3;`

ACHTUNG: Spezialfall Charakter `char` Bsp: `char c; c='A';` ist gleichbedeutend zu `c =65;`

8. Was geschieht wenn man zwei unterschiedliche Datentypen miteinander verknüpft?
Was sind Konversionen?

Konversionen sind Datentypumwandlungen C erlaubt dies es gibt keine Fehlermeldung

bsp: `int j; j=2.6;` → Nachkommazahl abgeschnitten der ist 2

Stil: Vermeide unterschiedliche Datentypen links und rechts von einer Zuweisung.
Falls unterschiedliche Datentypen explizite Konvertierung aufgrund der Lesbarkeit des Codes. `int i; float a; a=(float)i;`

→ Konversionen werden bezüglich einer Hierarchie von Datentypen ausgewertet.

→ niedriger Datentyp wird zu höherem Datentyp umgewandelt

Hierarchie

∧ double

∧ float

∧ unsigned int

∧ int

∧ unsigned short int

∧ short int

∧ unsigned char

∧ char

Beispiel: `int i; float a;`

`a+i;`

→ Inhalt von i wird von float konvertiert

→ Inhalt von i und a werden addiert

→ Ergebnis float

9. Was ist ein Pointer?

Ein Pointer ist eine Variable, die eine Speicheradresse speichert.

10. Wie deklarieren ich eine Pointer?

```
int*ptr;
```

→ `int*` ist der Typ, `ptr` der Name der Variable.

Variable die eine Adresse enthält, die auf eine oder mehrere Speicherzellen zeigt in denen eine ganze Zahl stehen soll.

11. Wie erhalte ich die Adresse einer Variablen?

```
int* ptr;
```

```
int num;
```

```
ptr=&num;
```

Das & Symbol bedeutet Adresse von. Die Variable ptr enthält jetzt die Adresse der Speicherzelle mit dem Namen num.

5.4 Operatoren und Ausdrücke

1. Was sind Ausdrücke?

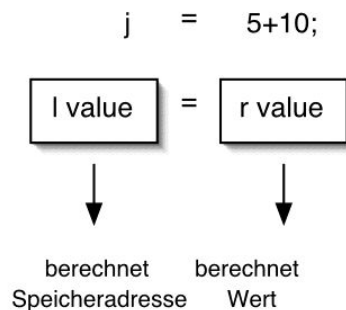
Ausdrücke sind Kombinationen von Variablen und Operatoren

2. Was sind Operatoren?

Operatoren sind die „Verben“ der Programmiersprache ermöglichen berechnen von Werten

3. Was ist der r value und der l value bei einem Ausdruck?

Jeder C Ausdruck evaluiert zu einem r value(Wert) oder einem l value(Adresse). Diese Werte stehen nach der Auswertung des Ausdrucks im Prozessorregister (Register der CPU). Sie können mithilfe eines Zuweisungsoperators in eine Speicherzelle des Hauptspeichers geladen werden.



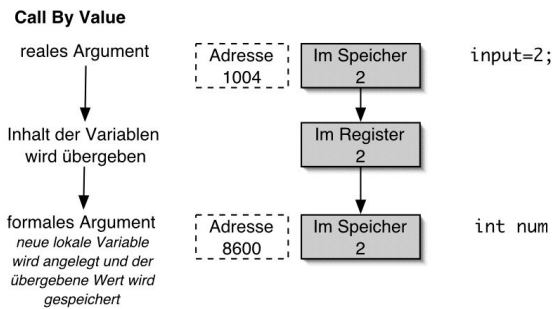
4. Was ist call by value und Call by reference?

Call by Value: Kopie des Inhaltes einer Variable wird übergeben an die Fkt.

Bsp:

```
int main(){
int answer, input=2;
answer=square(input)
}

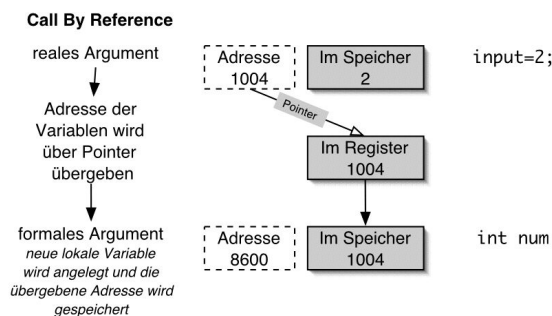
int square (int num){. . . }
```



Call by Reference: Kopie der Adresse einer Variable wird übergeben an die Fkt.
Bsp:

```
int main(){
int answer, input=2;
answer=square(&input)
}

int square (int * num){. . . }
```



5. Was sind Seiteneffekte?

Funktionsaufrufe, verschachtelte Zuweisungen und Inkrement und Dekrement Operatoren verursachen Nebenwirkungen, d.h., eine Variable wird geändert. Ein Seiteneffekt kann die reihenfolge der beteiligten Variablen ändern Bsp: a[i]=i++ ob der alte oder neue Wert verwendet wird ist Übersetzerabhängig.

6. Was gibt es für Operatoren?

- Arithmetische Operatoren: +,-,*,/,%

Achtung beim / Operator 5/3=1 → Abschneiden der Dezimalstelle -5/3=-1 oder -2 je nach Compiler

- Arithmetische Zuweisungsoperatoren = += -= *= /= %=

- Inkrement & Dekrement Operatoren ++ --

Bsp: int a=5;

`a++;` //Speicherzelle `a=6` Ausdruck evaluiert zu 5
`++a;` //Speicherzelle `a=6` Ausdruck evaluiert zu 6

- Vergleichsoperatoren `>= <= > < == !=`
 Verwenden Sie `==` nur fuer integrale Datentypen

- Logische Operatoren `&&`

! Achtung logische Operatoren werden von Links nach rechts evaluiert, Compiler evaluieren nur soviele wie nötig

- Bit Operatoren `>> << & — ^` , müssen ganzzahlige Datentypen sein

- Sonstige Datentypen `cast sizeof conditional`

- Speicher operatoren `adress of`

- Mathematische FUNKTIONEN Liegen in der Bibliothek `# include<math.h>`

Rang und Assozioativität Rang O. mit Vorrang werden zuerst evaluiert siehe Tabelle

5.4.1 Aus und Eingabefunktionen

1. Wozu dienen die Funktionen `printf()` und `scanf()`?

Beides sind Eingabe Ausgabe (I/O) Funktionen. Wobei `printf()` etwas auf den Bildschirm ausgibt, und `scanf()` etwas einliest.

Syntax `printf()`

```
int printf(char *format, arg1, arg2, ...)
```

Unter der Kontrolle der Zeichenkette *format* wandelt die Funktion *printf* ihre Argumente um. In der Formatzeichenkette gibt es gewöhnliche Zeichen die einfach ausgegeben werden und Umwandlungsangaben die mit % eingeleitet werden. Zwischen % und dem Umwandlungszeichen kann folgendes angegeben werden.

→ Minuszeichen- umgewandelte Argument wird nach links ausgerichtet.

→ eine Zahl die die minimale *Feldbreite* angibt

→ Ein Punkt der die Feldbreite von der Genauigkeit trennt

→ Eine Zahl, die *Genauigkeit* die die maximale Anzahl von Zeichen festlegt.

→ Buchstabe *h* für *short* und *l* für *long*

Als Feldbreite oder Genauigkeit kann jeweils * angegeben werden, dann wird der Wert durch Umwandlung des nächsten Argumentes festgelegt. *für die Formatanweisungen siehe nächste Frage.*

Syntax `scanf()`

```
int scanf(char *format, arg1, arg2, ...)
```

Praktisch die Umkehrung von `printf()`. `scanf()` liest Zeichen von der Standard Eingabe und interpretiert sie unter der Kontrolle von `format`.

2. Gib ein Beispiel!

```
/*Download:scanf.c*/
#include <stdio.h>

int main ()
{
int i;
printf("Bitte geben Sie eine Zahl ein : ");
scanf("%d",&i);
printf("Die Zahl die sie eingegeben haben war%d\n",i);
return 0; }
```

3. Was gibt es für Umwandlungen?

Zeichen	Argument, Ausgabe als
d,i	int, dezimale Zahl
o	int, oktale Zahl ohne VZ
x, X	int, hexadezimale Zahl ohne VZ
u	int, dezimale Zahl ohne VZ
c	int, einzelnes Zeichen
s	char*, aus der Zeichenkette werden zeichen ausgegeben bis Endzeichen \ 0
f	double, Nachkommastellen werden durch Genauigkeit festgelegt
e,E	double
g,G	double
p	void, als Zeiger hängt von der Implementierung ab
%	es wird kein Argument umgewandelt, ein % wird ausgegeben

4. Was gibt es für Escape Sequenzen?

Escapesequenzen	
\ a	BEL(bell) akustisches Warnsignal
\ b	BS(backspace) setzt Cursor um eine Position nach links
\ f	FF(formfeed) ein Seitenvorschub wird ausgelöst. Wird hauptsächlich bei Programmen verwendet mit denen man etwas ausdrucken kann.
\ n	NL(newline) Cursor geht zum Anfang der nächsten Zeile
\ r	CR(carriage return) Cursor springt zum Anfang der aktuellen Zeile
\ t	HT(horizontal tab) Zeilenvorschub zur nächsten horizontalen Tabulatorposition
\ v	VT(vertical tab) Cursor springt zur nächsten vertikalen Tabulatorposition
\ "	" wird ausgegeben
\ '	' wird ausgegeben
\ ?	? wird ausgegeben
\ \	\ wird ausgegeben
\ 0	NULL (ist die Endmarkierung eines Strings)
\ nnn	Ausgabe eines Oktalwertes.(z.B. \ 033 = ESCAPE-Zeichen)
\ xhh	Ausgabe eines Hexdezimalwertes. (z.B. \ xff = 255)

5.5 Kontrollfluß

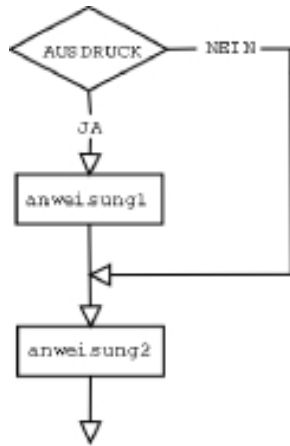
1. Wie kann man Kontrollflußmöglichkeiten in C grob unterteilen ?

Es gibt bedingte Sprünge (if, switch) und Schleifen (for, while, do-while).

- IF

In ihrer grundlegenden Form wertet die if-Anweisung einen Ausdruck aus und legt in Abhängigkeit vom Ergebnis dieser Auswertung fest, wo die Programmausführung fortzusetzen ist. Eine if-Anweisung hat folgende Form:

```
if (Ausdruck)
Anweisung;
```



Wenn Ausdruck wahr ist, wird Anweisung ausgeführt. Ist Ausdruck hingegen unwahr, wird Anweisung nicht ausgeführt. In beiden Fällen verzweigt die Ausführung in den Code, der auf die if-Anweisung folgt. Das lässt die Schlussfolgerung zu, dass die Ausführung von Anweisung von dem Ergebnis von Ausdruck abhängt. Die if-Anweisung besteht sowohl aus der Zeile if (Ausdruck) als auch aus der Zeile Anweisung;. Diese zwei Zeilen werden nicht als getrennte Anweisungen betrachtet.

Eine if-Anweisung kann die Ausführung mehrerer Anweisungen steuern, indem man einfach eine Verbundanweisung (einen Block) verwendet. Wie ich bereits zu Beginn dieses Kapitels definiert habe, versteht man unter einem Block eine Gruppe von zwei oder mehr Anweisungen innerhalb von geschweiften Klammern. Ein Block kann überall dort eingesetzt werden, wo auch eine einfache Anweisung verwendet werden kann.

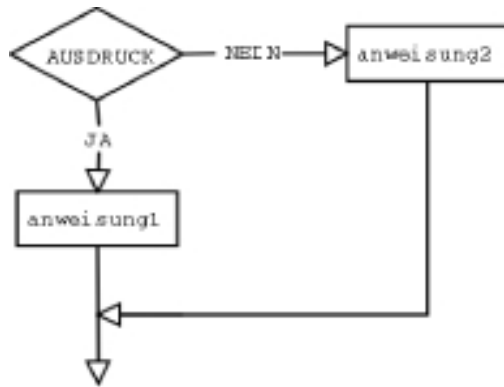
- IF-ELSE

Eine if-Anweisung kann optional eine else-Bedingung umfassen. Die else-Bedingung wird folgendermaßen mit aufgenommen:

```

if (Ausdruck)
Anweisung1;
else
Anweisung2;

```



Wenn Ausdruck zu wahr ausgewertet wird, wird Anweisung1 ausgeführt. Wenn Ausdruck zu unwahr ausgewertet wird, fährt das Programm mit der else-Anweisung, das heißt Anweisung2, fort. Beide Anweisungen, Anweisung1 und Anweisung2, können Verbundanweisungen oder Blöcke sein.

Beispiel

```

if (n>0)
if (a>b)
z=a;
else
z=b;

```

—>Das else gilt immer für das letzte if

Bezug auf ein früheres if nur mit Klammern!

```

if (n>0){
if (a>b)
z=a;
}
else
z=b;

```

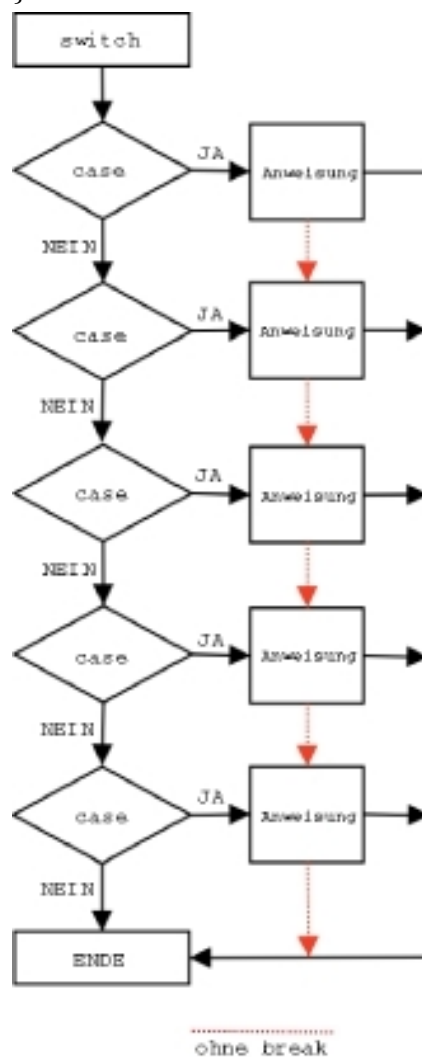
- SWITCH

Die wohl flexibelste Anweisung zur Steuerung des Programmflusses ist die switch- Anweisung. Mit ihr kann man weitere Programmausführung von Ausdrücken abhängig machen, die mehr als zwei Werte annehmen können. Programmsteueranweisungen, wie zum Beispiel if, waren auf boolesche Ausdrücke beschränkt, die nur einen von zwei Werten annehmen konnten: wahr oder falsch. Um den Programmfluss auf der Basis von mehr als zwei Werten zu

steuern, mussten mehrere verschachtelte if-Anweisungen verwendet werden. Mit der switch-Anweisung werden solche Verschachtelungen überflüssig.

Die allgemeine Form der switch-Anweisung lautet:

```
switch (Ausdruck)
{
case Konstante1: Anweisung(en);
case Konstante2: Anweisung(en);
...
case Konstante n: Anweisung(en);
default: Anweisung(en);
}
```



In dieser Anweisung ist Ausdruck ein beliebiger Ausdruck, der zu einem Integer-Wert des Typs long, int oder char ausgewertet wird. Die switch-

Anweisung wertet Ausdruck aus und vergleicht den Wert mit den Konstanten, die auf die case-Marken folgen. Diese drei Möglichkeiten ergeben sich:

→ Wird eine Übereinstimmung zwischen Ausdruck und einer der Konstanten gefunden, springt die Programmausführung zu der Anweisung, die auf die case- Marke folgt.

→ Wird keine Übereinstimmung gefunden, springt die Programmausführung zu der Anweisung, die auf die optionale default-Marke folgt.

→ Wird keine Übereinstimmung gefunden und gibt es auch keine default-Marke, wird das Programm mit der ersten Anweisung nach der schließenden Klammer der switch-Anweisung fortgesetzt.

Mit der switch-Anweisung können in Abhängigkeit vom Wert eines Ausdrucks in mehrere Anweisungsblöcke verzweigt werden. Diese Vorgehensweise ist effizienter und leichter nachzuvollziehen als eine tiefverschachtelte if-Anweisung. Eine switch- Anweisung wertet einen Ausdruck aus und verzweigt dann zu der case-Anweisung, deren Konstante mit dem Ergebnis des Ausdrucks übereinstimmt. Gibt es keine übereinstimmende case-Marke, springt die Programmsteuerung zu der Auffanganweisung default. Wenn keine default-Anweisung vorgesehen ist, springt die Programmsteuerung an das Ende der switch-Anweisung.

Der Programmfluss bewegt sich von der case-Anweisung kontinuierlich nach unten, es sei denn, eine break-Anweisung zwingt das Programm, an das Ende der switch- Anweisung zu springen.

Beispiel:

```
#include<stdio.h>
#include<conio.h>
main(){
char zeichen;
printf('\n Bitte ein zeichen eingeben:');
zeichen=getche();
switch(zeichen){
case'0':printf('\n''Null\n'');break;
case'1':printf('\n''Eins\n'');break;
default:printf('\n''Nichts\n''); break; }
}
```

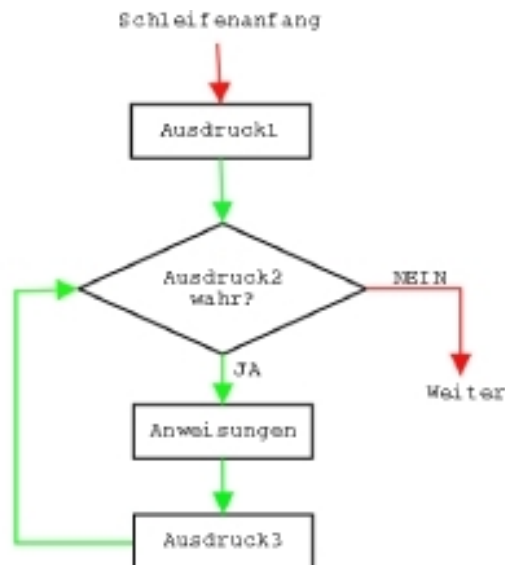
- FOR

Die for-Anweisung führt einen Block von einer oder mehreren Anweisungen mehrmals hintereinander aus. Manchmal wird sie auch for-Schleife genannt, da die Programmausführung diese Anweisung normalerweise mehr als einmal durchläuft.

Ein for-Anweisung hat die folgende Struktur:

```
for ( Initial; Bedingung; Inkrement )
```

Anweisung;



Initial, Bedingung und Inkrement sind allesamt C-Ausdrücke. Anweisung ist eine einfache oder komplexe C-Anweisung. Wenn die Programmausführung auf eine for- Anweisung stößt, passiert Folgendes:

1. Der Ausdruck Initial wird ausgewertet. Initial ist in der Regel eine Zuweisung, die eine Variable auf einen bestimmten Wert setzt.
2. Der Ausdruck Bedingung wird ausgewertet. Bedingung ist normalerweise ein relationaler Ausdruck (Vergleich).
3. Wenn Bedingung zu unwahr (das heißt Null) ausgewertet wird, endet die for- Anweisung und die Ausführung fährt mit der ersten Anweisung nach Anweisung fort.
4. Wenn Bedingung zu wahr (das heißt ungleich Null) ausgewertet wird, wird (werden) die C-Anweisung(en) in Anweisung ausgeführt.
5. Der Ausdruck Inkrement wird ausgewertet und die Ausführung kehrt zurück zu Schritt 2.

Eine for-Anweisung kann auch innerhalb einer anderen for-Anweisung ausgeführt werden. Dies nennt man Verschachteln. Eine Anweisung wird niemals ausgeführt, wenn Bedingung bereits bei der ersten Auswertung unwahr ergibt.

Beispiel für geschachtelte Schleife

```

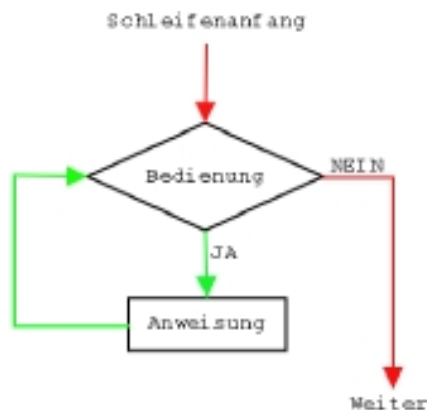
main(){
int x,y;
for(x=1; x<=10; x=x+1)
{printf(''\n'');
for(y=1; y<=10; y=y+1){
  
```

```
printf(''%4i'', x*y);
}}
```

- WHILE

Die while-Anweisung, auch while-Schleife genannt, führt einen Anweisungsblock so lange aus, wie eine spezifizierte Bedingung wahr ist. Die while-Anweisung hat folgende Form:

```
while (Bedingung)
Anweisung;
```



Bedingung ist ein beliebiger C-Ausdruck und Anweisung ist eine einfache oder komplexe C-Anweisung. Wenn die Programmausführung auf eine while-Anweisung stößt, passiert Folgendes:

1. Der Ausdruck Bedingung wird ausgewertet.
2. Wenn Bedingung als unwahr (das heißt als Null) ausgewertet wird, endet die while- Anweisung und die Ausführung fährt mit der ersten Anweisung nach Anweisung fort.
3. Wenn Bedingung als wahr (das heißt ungleich Null) ausgewertet wird, wird (werden) die C-Anweisung(en) in Anweisung ausgeführt
4. Die Ausführung kehrt zurück zu Schritt 1.

Bedingung ist ein beliebiger gültiger C-Ausdruck, in der Regel ein relationaler Ausdruck. Wenn Bedingung als unwahr (Null) ausgewertet wird, endet die while- Anweisung und die Ausführung fährt mit der ersten Anweisung nach Anweisung(en); fort. Im anderen Fall wird die erste Anweisung in Anweisung(en) ausgeführt.

Bei Anweisung(en) handelt es sich um die C-Anweisungen, die ausgeführt werden, solange die Bedingung wahr ist.

Die while-Anweisung gehört zu den Schleifen. Mit ihrer Hilfe kann man eine Anweisung oder einen Anweisungsblock so lange wiederholt ausführen lassen, wie eine gegebene Bedingung wahr ist (ungleich Null). Wenn die Bedingung

bereits bei der ersten Ausführung des while-Befehls nicht wahr ist, werden die Anweisung(en) nie ausgeführt. Eine while-Anweisung entspricht im Wesentlichen einer for-Anweisung, bei der die Initialisierung und Inkrementierung fehlen. Demzufolge ist

for (; Bedingung ;) äquivalent zu while (Bedingung)

Aufgrund dieser Ähnlichkeit kann alles, was mit einer for-Anweisung möglich ist, auch mit einer while-Anweisung bewältigt werden. Wenn jedoch while verwendet wird, müssen alle erforderlichen Initialisierungen zuvor in einer eigenen Anweisung vorgenommen werden und die Aktualisierung muss in Form einer Anweisung in der while-Schleife selbst erfolgen.

In allen Fällen, in denen Initialisierung und Aktualisierung unverzichtbar sind, bevorzugen die meisten erfahrenen C-Programmierer die for-Anweisung - hauptsächlich wegen der besseren Lesbarkeit des Quelltextes. Bei einer for-Anweisung stehen die Ausdrücke für Initialisierung, Test und Inkrementierung alle zusammen und sind damit leicht zu finden und zu ändern. Bei einer while-Anweisung stehen Initialisierung und Aktualisierung an getrennten Stellen und sind mitunter nicht leicht auszumachen.

- DO-WHILE

Die do...while-Schleife, führt einen Anweisungsblock so lange aus, wie eine bestimmte Bedingung wahr ist. Eine do...while-Schleife testet die Bedingung allerdings erst am Ende der Schleife und nicht am Anfang, wie das bei den for- und der while-Schleifen der Fall ist.

Der Aufbau einer do...while-Schleife sieht folgendermaßen aus:

```
do  
Anweisung;  
while (Bedingung);
```



Bedingung ist ein beliebiger C-Ausdruck und Anweisung ist eine einfache oder komplexe C-Anweisung. Wenn das Programm auf eine do...while-Anweisung trifft, passiert Folgendes:

1. Die Anweisungen in Anweisung werden ausgeführt.
2. Die Bedingung wird ausgewertet. Ist sie wahr, kehrt die Ausführung zurück zu Schritt 1. Ist sie falsch, wird die Schleife beendet.

Die Anweisungen einer do...while-Schleife werden immer mindestens einmal ausgeführt. Dies liegt daran, dass die Testbedingung am Ende der Schleife und nicht am Anfang ausgewertet wird. for- und while-Schleifen dagegen werten die Testbedingung zu Beginn der Schleife aus, so dass ihre Anweisungen nicht ausgeführt werden, wenn die Testbedingung bereits bei Eintritt in die Schleife unwahr ist.

Bedingung ist ein beliebiger gültiger C-Ausdruck, in der Regel ein relationaler Ausdruck. Wenn Bedingung als unwahr (Null) ausgewertet wird, endet die while- Anweisung und die Ausführung fährt mit der ersten Anweisung nach der while- Anweisung fort. Im anderen Fall springt das Programm zurück zu dem do-Teil und die C-Anweisung(en) in Anweisung(en) wird (werden) ausgeführt

Bei Anweisung(en) handelt es sich um eine einfache C-Anweisung oder einen Block von Anweisungen. Diese Anweisungen werden beim ersten Durchlauf der Schleife und danach so lange, wie Bedingung wahr ist, ausgeführt.

do...while-Schleifen sind vor allem dort angebracht, wo die Anweisung(en) der Schleife zumindest einmal ausgeführt werden muss (müssen). Das Problem kann auch ohne weiteres mit einer while-Schleife gelöst werden, indem man sicher stellt, dass die Testbedingung wahr ist, wenn die Programmausführung die Schleife das erste Mal erreicht. Mit ihrer Hilfe kann man eine Anweisung oder einen Anweisungsblock so lange wiederholt ausführen lassen, wie eine gegebene Bedingung wahr ist (ungleich Null). Im Gegensatz zu einem while-Befehl werden die Anweisungen einer do...while-Schleife mindestens einmal ausgeführt.

2. Was ist besser, eine switch-Anweisung oder eine verschachtelte if-Anweisung?

Wenn man eine Variable prüft, die mehr als zwei Werte annehmen kann, ist die switch-Anweisung fast immer die bessere Alternative. Der resultierende Code ist einfacher zu lesen. Wird eine wahr/falsch-Bedingung getestet, ist die if-Anweisung vorzuziehen.

3. Warum sollte man goto-Anweisungen vermeiden?

Auf den ersten Blick scheint es, als ob die goto-Anweisung eine recht nützliche Anweisung sei. goto kann jedoch mehr Probleme verursachen, als es behebt. Eine goto-Anweisung ist ein unstrukturierter Befehl, der Sie zu einem anderen Punkt in einem Programm springen lässt. Viele Debugger können goto-Anweisungen nicht ordentlich zurückverfolgen. Außerdem führen goto-Anweisungen zu Spaghetti-Code - Code ohne strukturieren Programmfluss.

4. Woher weiß man, welche Anweisung zur Programmsteuerung verwendet muss - for, while oder do...while?

Mit allen dreien kann man ein Schleifenproblem lösen. Jede Anweisung hat jedoch so ihre Eigenheiten. Die for-Anweisung eignet sich am besten, wenn man in der Schleife initialisieren und inkrementieren muß. Wenn man lediglich eine Bedingung abfragen und keine bestimmte Anzahl von Schleifeniterationen durchführen will, ist while zu empfehlen. Und wenn man davon ausgeht, dass der zugehörige Anweisungsblock mindestens einmal ausgeführt werden muss, ist do...while die beste Wahl.

5. Wie tief kann ich meine Schleifen verschachteln?

Für die Verschachtelungstiefe sind keine Grenzen gesetzt. Wenn das Programm mehr als zwei Verschachtelungsebenen benötigt, sollte man überlegen, ob man nicht besser eine Funktion verwendet (lesbarkeit).

6. Kann man unterschiedliche Schleifenbefehle verschachteln?

Ja, man kann if, for, while, do...while oder irgendwelche anderen Befehle verschachteln.

5.6 Arrays und Pointer

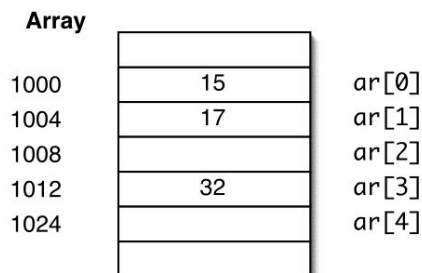
1. Was ist ein Array?

Ein Array ist die Zusammenfassung von vielen Variablen des gleichen Namens und des gleichen Datentyps wie beispielsweise eine Zeitreihe mit 1000 Werten für eine Aktie.

2. Wie initialisiere ich ein Array?

```
int ar[5]={7,-3,9,8,-5};  
int a[5];
```

3. Wie sieht ein Array im Speicher aus?



Bei einem Array werden die Speicherzellen hintereinander belegt.

Bsp:

`p=ar;` → Die Speicherzelle `p` enthält die Adresse 1000.

`ar[3];` ist gleich `p[3];` Zugriff auf Element 3 über Array-Operatoren

`*(ar+3);` ist gleich `*(p+3);` Zugriff auf Element 3 über Pointerarithmetik

→ daraus folgt: volle Äquivalenz zwischen arrays und Pointer `ar=p`

4. Wie greift man auf ein Arrayelement zu?

`aktie[5];` evaluiert den Wert des 5.Elementes

`aktie[0];` evaluiert den Wert des 0.Elementes (!)

5. Welche Elemente sind in einem Array zugelassen?

Alle Ausdrücke und Variablen die die zu nicht-negativen, ganzen Zahlen evaluieren.

6. Was passiert, wenn ich für ein Array einen Index verwende, der größer ist als die Zahl der Elemente im Array?

Wenn man einen Index verwendet, der mit der Array-Deklaration nicht übereinstimmt, wird sich das Programm wahrscheinlich trotzdem kompilieren und ausführen lassen. Ein solcher Fehler kann jedoch zu unvorhersehbaren Ergebnissen führen. Zudem sind solche Fehler meist nur sehr schwer zu finden. Deshalb bei der Initialisierung und dem Zugriff auf die Array-Elemente größte Sorgfalt walten lassen.

7. Was passiert, wenn ich ein Array verwende, ohne es zu initialisieren?

Dieser Fehler löst keinen Compilerfehler aus. Wenn man ein Array nicht initialisiert, können die Array-Elemente einen beliebigen Wert annehmen. Das Ergebnis kann unvorhersehbar sein. Man sollte Variablen und Arrays immer initialisieren, so dass man genau weis, was darin enthalten ist.

8. Wie viele Dimensionen kann ein Array haben?

Man kann so viele Dimensionen angeben, wie man will. Doch je mehr Dimensionen man hinzufügt, umso mehr Speicherplatz wird benötigt. Man sollten ein Array nur so groß wie nötig deklarieren, um keinen Speicherplatz zu vergeuden.

9. Gibt es eine einfache Möglichkeit, ein ganzes Array auf einmal zu initialisieren?

Jedes Element eines Arrays muss initialisiert werden. Der sicherste Weg besteht darin, ein Array entweder im Zuge der Deklaration oder mit Hilfe einer `for`-Anweisung zu initialisieren.

10. Kann ich zwei Arrays addieren (oder multiplizieren, dividieren oder subtrahieren)?

Arrays können nicht addiert werden. Alle Elemente müssen einzeln addiert werden.

11. Warum ist es besser, ein Array statt einzelner Variablen zu verwenden?

Mit Arrays kann man gleiche Werte unter einem einzigen Namen zusammenfassen. Das Erzeugen von beispielsweise 1000 Variablennamen und der Initialisierung jeder einzelnen Variablen mit einer Zufallszahl würde einen gewaltigen Tippaufwand erfordern. Durch die Verwendung eines Arrays wird die Aufgabe einfacher.

12. Was macht man, wenn man nicht weiß, wie groß das Array werden muss, beim schreiben des Programms?

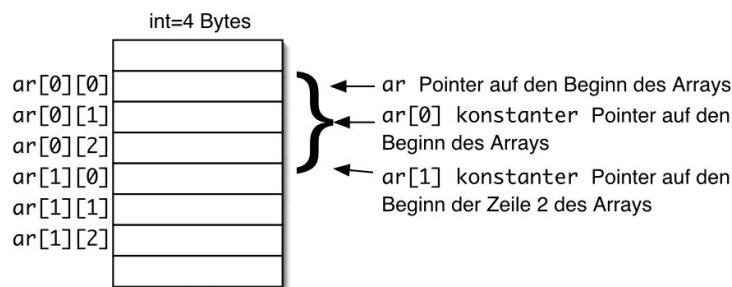
Es gibt Funktionen in C, mit denen man ein passant Speicher für Variablen und Arrays allokiert kann.

13. Wie deklariert man ein mehrdimensionales Array?

`int ar[Anzahl der Zeilen][Anzahl der Spalten];` (zweidimensionales Array)

14. Beschreibe den Zugriff und die Speicherbelegung auf ein zweidimensionales Array!

mehrdimensionales Array: `int ar [2][3];`



äquivalente Zugriffsmöglichkeiten

Bsp. Element 1,1

`ar[1][1];`

`(ar[1]+1);` Einstieg auf Zeile 2

`*(ar[0]+4);` Einstieg auf Zeile 1+4 Einheiten

`*(*(ar+1)+1);` Erhöhung um drei "Briefkästen"

15. Wie funktioniert die Übergabe von Arrays an eine Funktion?

Sie funktioniert über "call by reference", dh, Übergabe des Pointers auf den Beginn des Array.

Deklaration der Funktion

Größe des Arrays muss mitübergeben werden.

```
void func(float *ar, int d) ENTSPRICHT void func(float ar[], int d)
{
  ar[5];
  (ar+5);
} Aufruf und Übergabe
```

```
float fa[500];  
func(fa,500); oder alternativ func(&fa[0],500);
```

16. Pointer auf Pointer ????

Bsp.:

```
int *v,*q, **p;
```

17. Arrays von Pointern ????

!!!! FEHLT NOCH !!!!! FEHLT NOCH !!!!!

18. Was ist ein Pointer?

Der Arbeitsspeicher besteht aus vielen Millionen aufeinander folgender Speicherstellen, die jede durch eine eindeutige Adresse identifiziert werden. Die Speicheradressen reichen von 0 bis zu einem Maximalwert, der davon abhängt, wie viel Speicher installiert ist. Wenn man eine Variable deklariert, reserviert der Compiler eine Speicherstelle mit einer eindeutigen Adresse zur Speicherung dieser Variablen. Der Compiler verbindet diese Adresse mit dem Variablennamen. Wenn das Programm diesen Variablennamen verwendet, greift es automatisch auf die korrekte Stelle im Speicher zu. Die Adresse einer Variablen ist also die Adresse des ersten (niedrigsten) Byte, das von der Variablen belegt wird. Jeder Zeiger entspricht der Adresse des ersten Byte der Variablen, auf die gezeigt wird. Bei jedem Zeiger wird bei der Deklaration der Typ der Variablen, auf die er zeigt, mitgegeben. Der Compiler weiß also, dass ein Zeiger auf den Typ int auf das erste von vier Byte zeigt, ein Zeiger auf den Typ double auf das erste von acht Byte und so weiter.

19. Wie werden Pointer deklariert?

Ein Pointer ist eine numerische Variable und muss, wie alle Variablen, deklariert werden, bevor sie verwendet werden kann. Die Namensgebung für Zeigervariablen folgt den gleichen Regeln wie für andere Variablen. Der Name muss eindeutig sein: `typename *zgrname;`

wobei `typename` ein beliebiger Variablentyp von C ist und angibt, von welchem Typ die Variable ist, auf die der Zeiger verweist. Der Stern (*) ist der Indirektionsoperator und macht deutlich, dass `zgrname` ein Zeiger auf den Typ `typename` ist und keine Variable vom Typ `typename`. Zeiger können zusammen mit normalen Variablen deklariert werden. Die Adresse einer Variablen wird mit & in den Speicher des Pointers gelegt. Wenn dieser Adressoperator vor den Namen einer Variablen gesetzt wird, liefert er die Adresse der Variablen zurück. Deshalb werden Zeiger mit Anweisungen der folgenden Form initialisiert:

```
zeiger = &variable;
```

Mit Hilfe des Indirektionsoperators '*' vor dem Namen eines Zeigers, bezieht man sich auf die Variable, auf die verwiesen wird.

20. Warum sind Zeiger so wichtig in C?

Zeiger geben Ihnen eine größere Kontrolle über den Computer und Ihre Daten. Als Parameter von Funktionen erlauben sie Ihnen, die Werte der übergebenen Variablen in der Funktion zu ändern. Am Tag 14 zeige ich Ihnen noch weitere Einsatzbereiche für Zeiger.

21. Wie erkennt der Compiler den Unterschied zwischen dem * für Multiplikation, für Dereferenzierung und für die Deklaration eines Zeigers?

Der Compiler interpretiert die verschiedenen Verwendungen des Sternchens anhand des Kontextes, in dem es verwendet wird. Wenn die ausgewertete Anweisung mit einem Variablentyp beginnt, kann davon ausgegangen werden, dass mit dem Sternchen ein Zeiger deklariert wird. Wenn das Sternchen in einer Anweisung zusammen mit einer Variablen verwendet wird, die als ein Zeiger deklariert wurde, wird mit dem Sternchen wahrscheinlich dereferenziert. Wird es hingegen in einem mathematischen Ausdruck ohne Zeigervariable verwendet, kann man davon ausgehen, dass das Sternchen ein Multiplikationsoperator ist.

22. Was passiert, wenn ich den Adressoperator auf einen Zeiger anwende?

Man erhält die Adresse der Zeigervariablen. Ein Zeiger ist nur eine weitere Variable, die die Adresse der Variablen enthält, auf die sie zeigt.

23. Werden Variablen immer an der gleichen Speicherstelle gespeichert?

Nein. Jedes Mal, wenn ein Programm ausgeführt wird, können die Variablen des Programms an anderen Adressen im Computer gespeichert werden. Man sollte deshalb niemals einem Zeiger eine konstante Adresse zuweisen.

24. Werden in der C-Programmierung Zeiger häufig als Funktionsargumente übergeben?

Absolut! In vielen Fällen benötigt man Funktionen, die Werte von etlichen Variablen ändern müssen, und es gibt zwei Wege, dies zu realisieren. Der eine besteht darin, globale Variablen zu deklarieren und zu verwenden. Der zweite Weg besteht in der Übergabe von Zeigern, so dass die Funktion die Daten direkt ändern kann. Die erste Option ist nur zu empfehlen, wenn fast jede Funktion die Variable verwendet. Im anderen Fall sollte man globale Variablen vermeiden.

25. Was ist ein Nullpointer?

Ein nullpointer zeigt garantiert auf keine gültige Adresse und evaluiert somit zu falsch. Eingesetzt wird er als Abbruchkriterium. `char *p; p=0;`

26. Was ist die Pointerarithmetik?

Sie ermöglicht den Zugriff auf Arrayelement über Pointer, da die relative Lage der Daten klar ist. So können mit Adressen Rechenoperationen ausgeführt werden.
 $p=p1+3$ $p1$ zeigt auf eine um drei Brifkästen nach oben verschobene Adresse.

5.7 Speicherklassen

1. Was ist die DAUER und der BEREICH im Zusammenhang mit Speicherklassen?

DAUER: Zeitspanne über die eine Variable gültig ist, während der Abarbeitung eines Programmes. Für die Speicheroptimierung heisst dies, bei einem Funktionsaufruf werden lokale Variablen nur während der Abarbeitung gebraucht.

BEREICH: Bereich des Quellcodes über den die Variable gültig ist (information hiding). Das ist nützlich bei der Entwicklung grosser Programme → Vermeidung von Namenskonflikten.

2. Erläutere den Unterschied zwischen automatisch und fest beim Begriff DAUER!

AUTOMATISCH: Der Speicher wird reserviert wenn der entsprechende Bereich abgearbeitet wird. Bei jedem Bereichseintritt wird neu initialisiert, keine Default Initialisierung.

Bsp: `auto int j;` auto kann wegelassen werden.

FEST: Der Speicher wird beim ersten Eintritt in den Bereich reserviert, für die gesamte Dauer des Programms. Beim ersten Eintritt in den Bereich zu initialisieren mit Default

bsp: `static int j;`

3. Was ist variable hiding?

Wenn lokale Variablen globale verdecken.

Bsp eines Namenskonfliktes:

```
int j;  
func(){int j;j}
```

4. Was bedeutet register und volatile in Bezug auf die Speicherverwaltung?

register: Daten werden nicht in den Hauptspeicher sondern im Prozessorregister gespeichert.

Vorteil: Schneller Zugriff

Nachteil: wenn die Daten im Register sind haben Sie keine Adresse, dadurch ist ein Zugriff mit Pointern unmöglich.

Bsp:

`register int leu=0;` register ist dazu das Schlüsselwort

`volatile`: Daten können modifiziert werden ohne das der Compiler davon weiss. Der grund hierfür ist das bestimmte Optimierungsstrategien des Compilers ausser Kraft gesetzt sind.

5. Welche Funktionen zur dynamischen Speicherverwaltung gibt es?

`malloc()`: reserviert Speicher einer vorgegebenen Grösse und gibt Pointer anschliessend einen Pointer des reservierten Bereiches zurück. Die Funktion gibt Pointer auf void zurück und wird automatisch zum richtigen Typ konvertiert. TROTZDEM! Benutze explizite Konvertierung wegen der besseren Lesbarkeit.

Bsp.:

```
int*list;  
list=(int*) malloc(40);
```

Die 40 bedeutet 40 bytes.

`calloc()`: wie `malloc()` nur Initialisierung des Bereiches auf Null.

`realloc()`: ändert die Grösse von Speicherbereichen.

`free()`: gibt entsprechenden Speicherbereich frei

5.8 Strukturen und Unions

1. Was sind structures ?

Sie sind ein benutzerdefinierter Datentyp. Dabei können Daten von einem beliebigen Typ gruppiert werden , das ist der Unterschied zu der Definition eines neuen Datentyps. Strukturen sind eine Ansammlung von einer oder mehreren Variablen, möglicherweise verschiedenen Typs die unter einem Namen zur bequemen handhabung zusammengefasst sind.

2. Wie sieht die Syntax einer Structur aus?

Variante laut Kernigan

```
struct point{int x; int y;};
```

Variante in Obermayervorlesung:

```
typedef struct{  
char polname[20], polpassiv[15];  
short int pdmonat, pdtag, pdjahre;  
}PERSDAT; → Persdat ist der Name des neuen Datentyps
```

`struct` definiert einen Datentyp nach der rechten geschweiften Klammer kann eien Liste von Variablen stehen genau wie bei jedem elemnetaren Datentyp

`struct { ... }x,y,,z;` ist syntaktisch gleich zu `int x,y,z;`

Folgt keine Variablenliste wird auch kein Speicherplatz reserviert es ist lediglich eine Beschreibung der Struktur. Wenn die Struktur aber ein Etikett (im ersten Bsp point) kann dies spatter zur Identifikation benutzt werden

Bsp: `struct point pt;` definiert eine Variable `pt`; die eine Struktur vom Typ `struct point` ist.

Der Verweis auf eine Komponente einer Struktur erfolgt über Punktyntax
Struktur-Variablenname.Komponente

Bsp:

```
printf("%d,%d",pt.x,pt.y)
```

Die Adresse einer Struktur kann man mit `&` bestimmen

`struct point *pp;` legt fest das ein Zeiger auf eine Struktur vom Typ `point`

wenn `*pp` die Struktur ist dann sind `(*pp).x` die Komponenten. Die Klammern sind notwendig um den Vorrang des Punktes deutlich zu machen. Andere Schreibweise `p->Komponente` einer Struktur

Bsp

```
struct point origin,*pp;
```

```
pp=&origin;
```

```
printf(''origin is (%d,%d)\ n'', (*pp).x, (*pp).y);
```

verkuerzte Schreibweise

```
printf(''origin is (%d,%d)\n'', pp->x, pp->y);
```

3. Was ist eine union?

Eine Union ist eine Variable die zu verschiedenen Zeitpunkten Objekte mit verschiedenen Datentypen und Größen enthält, wobei der Übersetzer die nötige Größe und Ausrichtung des Speicherbereiches überwacht. Laut Obermayer sind Unions ein Datentyp der gleichzeitig verschiedenen Interpretationen eines Speicherbereiches erlauben. Daraus folgt das die Elemente eines unions sich im Speicher überlappen.

Die einzelnen Komponenten einer union werden nicht hintereinander im Speicher abgelegt, sondern beginnend an derselben Speicherstelle , so dass sich die Speicherbereiche überschneiden und im allgemeinen nur eine Komponente einen sinnvollen Wert besitzt. Bsp.:

```
typedef union
```

```
{
```

```
struct{char c1,c2;}
```

```
int j;
```

```
float x;
```

```
}u;
```

4. Wie erfolgt der Zugriff auf die Elemente eines union?

Mit dem Punktoperator `unionname.element` oder über Pointer auf union (->).

5. Was ist typedef?

Neue Typnamen können in C mit Hilfe von typedef vereinbart werden.

Bsp:

```
typedef int Length;
```

Mit dieser Vereinbarung ist Length synonym für int.

5.9 Verkettete Datenstrukturen

1. Warum macht man verkettete Datenstrukturen?

Vorteile

2. Benenne Merkmale von Listen !

-bestehen aus vielen Elementen

-jedes Element einzeln in Speicher verstaut

-jedes Element enthält Informationen über seine Nachbarn. Es enthält Pointer auf seinen Nachbarn in der Datenstruktur.

3. Was sind Listen Alternative zum Array Einfach verkettete Liste (hinmalen9

4. Welche typische Funktionen gibt es bei verketteten Listen?

zwei Datentypen erzeugen ELEMENT + LISTE verwendet: typedef, struct

Erzeugen eines Listenelementes Pointer auf die Struktur ELEMENT, Speicher alloziiert mit malloc

Suche + Einfügen nach letztem Element (kann man auch trennen in 2 Fkt.) Zählt durch bis er nullpointer gefunden hat und hängt neues Element ran If else Schleife

Einfügen eines Elementes Über Pointer wird das neue Element anstelle des alten gesetzt POINTER nur vertauscht

Entfernen eines Elementes Über Pointer For, if else und free() um Speicher wieder freizugeben

-

Doppelt verkettete Liste

Vorteile :dynamische Länge Bestandteile einer Liste: Pointer auf den Beginn/Pointer auf das Ende Listenelemente bestehen aus Inhalt und Daten und Verweisen(Pointer auf Nachbarelemente) Listenabschluß durch Nullpointer

5.10 Funktionen

1. Aufbau einer Funktion

Rückgabebetyp Name(Deklaration, der, Argumente){C-Code}

2. Wie ruft man eine Funktion auf?

Name der Funktion(übergabe der Argumente)

Rückgabe von Werten Return Ausdruck ; zum Stil falls kein return void angeben

3. Wie kann ich mehrere Werte zurückgeben?

—> Pointer auf struktur oder ein Array

—> call by reference durchführen !VORSICHT SEITENEFFEKTE!

—> globale datenstrukturen

4. Was ist eine Allusion was eine Definition?

Es gibt zwei Arten von Deklaration für globale Variablen.

DEFINITION: Speicher wird reserviert, Verbindung Name—Adresse

Darf nur in einer Datei sein möglichst da, wo die main Funktion drin steht.

Bsp:

```
int j=0;
```

ALLUSION: Information an den Compiler über a) Existenz b)Eigenschaften

In jeder Datei, Stil in die header Dateien einbinden mit include

Bsp:

```
extern int j;
```

5. Was ist eine Rekursion?

Funktionen in C können sich selbst aufrufen

6. Einsatz in Sortieralgorithmen

Aufsteigende vs absteigende Rekursion

Vgl Array Funktion—> Pointer auf Funktionen (babak?)

5.11 Präprozessor

1. Wozu dient der Präprozessor?

Er ist ein separates Programm das den Quelltext vor der Compilierung editiert.

Er kann Makros beinhalten die Abkürzungen für Zeichenketten sind oder er fügt neue zusätzliche Quelldateien ein. Man spricht von bedingter Compilierung.

2. Vor und Nachteile von Makros.

Vorteil: Makros sind schneller, da keine Parameter übergabe erfolgt. Kein Type checking, das heisst Funktionen für beliebigen Datentyp.

Nachteil: werden bei jedem Aufruf übersetzt, kein Information- hiding, dadurch mögliche Seiteneffekte.

3. bedingte Übersetzung mit den Befehlen `if else elif endif`

4. Wie füge ich Quelldateien ein?

Mit dem Befehl `# include`.

5.12 Ein- und Ausgabe

1. Was ist ein ID-Array von Bytes?

Der Stream ist in eine Art 'ID-Array von Bytes' unterteilt, d.h. er besteht aus vielen Arrays oder Blöcken, die durch das Offset von Bytes gekennzeichnet sind.

2. Was ist ein Stream ?

Streams sind einfache Datenströme, womit Daten von der Quelle zum Ziel bewegt werden. Dabei wird kein Unterschied zwischen verschiedenen devices gemacht. Die Standard-Streams in C sind die Standardeingabe (`stdin`), die Standardausgabe (`stdout`) und die Standardfehlerausgabe (`stderr`). Es gibt zwei Ebenen oder Formate für Streams: Der Unterschied zwischen der höheren Ebene und der niedrigeren Ebene der Ein/-Ausgabe liegt in der Form, wie die Daten im Stream von der Quelle zum Ziel übertragen werden. Bei der höheren Ebene (`text`) besteht der Stream aus einer Folge von Zeichen, durch `\newline` terminiert. Einige Bytes werden als Steuerzeichen interpretiert. Bei der niedrigeren Ebene (`Binary`), handelt es sich um einen unformatierte Folge von Bytes, die nicht interpretiert werden.

3. Wie greife ich auf einen Stream zu?

Mit einem Pointer auf eine Struktur vom Datentyp `FILE`. Der Aufbau der Struktur ist implementationsabhängig und deklariert in dem `stdio.h`. `fseek()` Bewegt den Positions-Pointer an eine gewünschte Stelle im Stream: `int fseek(FILE *, long int, int);`

`FILE *` Pointer auf Stream `long int` Offset in Bytes `int SEEK-SET: Dateibeginn` oder `SEEK-CUR: derzeitiger Pointer` oder `SEEK-END: Dateiende`

Beim `Binary-Stream` kann das `Offset-Argument` positiv oder negativ sein aber der `Positions- Pointer` muß innerhalb des Streams bleiben. Beim `Text-Stream` muß das `Offset-Argument` Null oder ein Wert sein, der von `ftell()` zurückgegeben wird (`ftell()` gibt den aktuellen Wert des `Position-Pointers` zurück).

4. Wie funktioniert das Öffnen und schließen von Dateien?

fopen() – Funktion:

```
FILE *fp; if ((fp=fopen("Test", "r"))==NULL) fprintf(stderr, "Error opening\n");
```

Dabei ist "Test" der Filename und "r" der Modus. fopen() ist die Standardabfrage ob eine Datei geöffnet wurde. fopen() gibt einen Pointer vom Typ FILE * zurück.

fclose() – Funktion:

```
FILE *fp; fclose(fp);
```

fclose(fp) gibt den reservierten Speicher wieder frei.

5. Wie funktioniert das lesen und schreiben von Dateien?

C Funktionen (langsamer aber keine Seiteneffekte):

- Byte: fgetc(), fputc() Einheiten: - Zeile: fgets(), fputs() - Block: fread(), fwrite()

Makros sind schneller, haben Seiteneffekte:

Einheiten: - Byte: getc(), putc()

6. Was ist Bufferung?

Speichermedien sind langsam, deshalb wird zeilen- oder blockweise geschrieben und gelesen. Diese werden zeitweilig in „Buffers“ gespeichert. Gelesen werden immer ganze Zeilen/Blöcke. Zeilen-Buffers: Alle Bytes bis zum nächsten newline; Blöcke-Buffers: feste Zahl von Bytes. Ein Unbuffered I/O ist nur sinnvoll, wenn Bytes separat präsentiert werden sollen. Nachteil: Ist extrem langsam!

```
setbuf(stdin, NULL);
```

dabei ist stdin der Stream und NULL ein Pointer auf den Buffer. NULL bedeutet 'kein Buffering'.

7. Was für Fehlermeldungen gibt es?

Alle Leser- und Schreibroutinen geben Fehlercodes zurück. Die FILE- Struktur enthält Elemente, die das Ende einer Datei anzeigen.

feof() : überprüft das EOF (end of file) Flag ferror() : überprüft das Fehlerflag clearerr() : Setzt beide Flags zurück, da dies nicht automatisch geschieht

Index

- Allusion, 61
- Arbeitsverzeichnis, 18
- Array, 52
 - Dimension, 53
 - index, 53
 - Zugriff, 53
- ASCII, 5
- Assembler, 12
- Aufbau, 33
- Ausdrücke, 35
- Ausdruck
 - r und l value, 39
- Baumstruktur, 18
- Betriebssystem, 12
- binäres Alphabet, 5
- Bufferung, 63
- Cache, 15
- Compiler, 33
 - Aufruf, 35
- CPU, 12, 14, 16
- Datei, 17
 - Zugriff, 19
- Dateisystem, 18
- Datentyp, 35
- DMA, 16
- Escapesequenzen, 43
- Exponent, 6
- Flip Flop, 9
- Funktion, 33
 - main, 33
 - printf()/scanf(), 41
 - Speicherverwaltung, 58
- ganze Zahlen, 5
- Halbaddierer, 10
- Header, 35
- Hochsprache, 12
- I-Node, 19, 20
 - Daeioperation, 20
- Kern, 17
- Konstante, 34
- Kontrollfluss, 44
 - do while, 50
 - for, 47
 - goto, 51
 - if, 44
 - if else, 44
 - switch, 45, 51
 - verschachteln, 52
 - while, 49
- Konversion, 38
- Link, 18, 20
- Listen, 26, 60
- Mantisse, 6
- Maschinensprache, 12
- Microcode, 12
- Modularisierung, 32
- Multitasking, 21
- Nullpointer, 56
- Offset, 23
- Operator, 40

- Operatoren, 28, 39
- Pipe, 19, 26
- Pointer, 38, 55
 - Deklaration, 38, 55
- Pointerarithmetik, 57
- Präprozessor, 35, 61
- Prioritäten, 25
- Prozess, 18, 21
 - Initiierung, 21
 - Kommandos, 28
 - Prozesstabelle, 23
 - Scheduling, 23
 - Zustände, 21
- Prozessorbus, 16
- Pseudocode, 33

- Regionen, 23
- Register, 14
- Rekursion, 61

- Schalenmodell, 17
- Seiteneffekte, 40
- Shell, 25
 - for, 29
 - if, 28
 - repeat, 30
 - Skript, 25
 - switch, 29
 - while, 29
 - Zeichen, 25
- signed,unsigned, 36
- Speicher, 22, 57
 - Array, 52
- Speicherarten, 15
- Spezifikation, 32
- Struktur, 58

- Umwandlungen, 43
- Union, 59

- Variable, 34
 - Adresse, 38
 - Deklaration, Initialisierung, 37
 - global, lokal, 34
 - variable hiding, 57
- Verzeichnis, 17
- Volladdierer, 11

- Zweierkomplementdarstellung, 6