

- Daten = Teilmenge aller darstellbaren Informationen, Zeichenketten (kodierte Signalfolgen), die Informationen repräsentieren und zur Speicherung und Verarbeitung in Computern bestimmt sind.
→ bestehen aus beliebigen Zeichen des rechnerexternen Alphabets
→ Mit ihnen dürfen Transport- und Vergleichsoperationen ausgeführt werden.

Grundfunktionen der Datenverarbeitung :

- DV-Systeme (oft als Computer bezeichnet) sind Anlagen, die dazu dienen, Daten, die in das System eingegeben werden oder dort digital gespeichert sind,
 - _ zu verknüpfen,
 - _ zu verarbeiten, um die Ergebnisse wieder
 - _ abzuspeichern oder dem Benutzer in geeigneter Form
 - _ zur Verfügung zu stellen.

Vorgänge im Datenverarbeitungssystem (DVS → 5 Kategorien von Grundfunktionen

- Ein-/Ausgabe (input,output): Führt einem DVS alle Daten zu, die verarbeitet werden sollen und stellt die Ergebnisse einem Nutzer zur Verfügung.
- Transport (transfer): Ermöglicht das Zusammenspiel der einzelnen Funktionseinheiten.
- Speicherung (storage): Sichert die Verfügbarkeit der Daten zwischen den Ein- und Ausgabevorgängen.
- Verknüpfung (processing): Beinhaltet die eigentliche Verarbeitung von Daten.
- Steuerung (control): Bewirkt das folgerichtige Zusammenspiel aller Funktionen.

Programm:

= Aufeinanderfolgende Teilschritte, die verschiedene zusammenhängende Funktionsanweisungen in einer maschinell verarbeitbaren Sprache (Maschinensprache) darstellen

= sind formgebundene, stofflich konkretisierte Geistesschöpfungen.

- Programmieren = Algorithmen zu kodieren auf:
 - _ Maschinen-Niveau,
 - _ maschinenorientiertem Niveau,
 - _ problemorientiertem Niveau (=virtuelle Betrachtungsweise einer Maschine).
- Werden diese Grundfunktionen durch technische Einrichtungen bewirkt = Hardware

durch den Ablauf von informativischen Funktionsanweisungen = Software (= ist die Gesamtheit aller Verarbeitungsprogramme, d.h. aller festgelegten Funktionsabläufe der DV)

- alle Computeranwendungen = Datenverarbeitungen

Algorithmierung

- Ohne Algorithmus kein Programm → Ohne Programm keine Ausführung auf dem Rechner!

Algorithmus:

= Beschreibung, wie eine Aufgabe vom Computer auszuführen ist

= beschreibt die Methode, mit der eine Aufgabe gelöst wird. #

- besteht aus einer Folge von Schritten, deren korrekte Abarbeitung die gestellte Aufgabe löst. → Vorgang = Prozess
- dient stets zur Lösung einer Klasse von Aufgaben einheitlichen Typs.

Arten von Algorithmen:

1. Verbale Beschreibung
2. Computerprogramm

Computer

= spezieller Prozessor,

- 3 Hauptkomponenten (=Hardware):
 1. Zentraleinheit (führt Basisoperationen aus)
 2. Speicher enthält:
 - a) die auszuführenden Operationen als Algorithmus
 - b) die Daten, auf denen die Operationen wirken
 3. Ein- und Ausgabegeräte :über ihn werden Algorithmus und Daten in Hauptspeicher gebracht und Ergebnisse mitgeteilt.

Prozessor muß Algorithmus interpretieren können (versteht+ ausführen)

→ Algorithmus in Programmiersprache → A besteht aus Folge von Anweisungen, von denen jede Operationen angibt, die der Computer ausführen soll.

A = unabhängig von Programmiersprache und Computertyp.

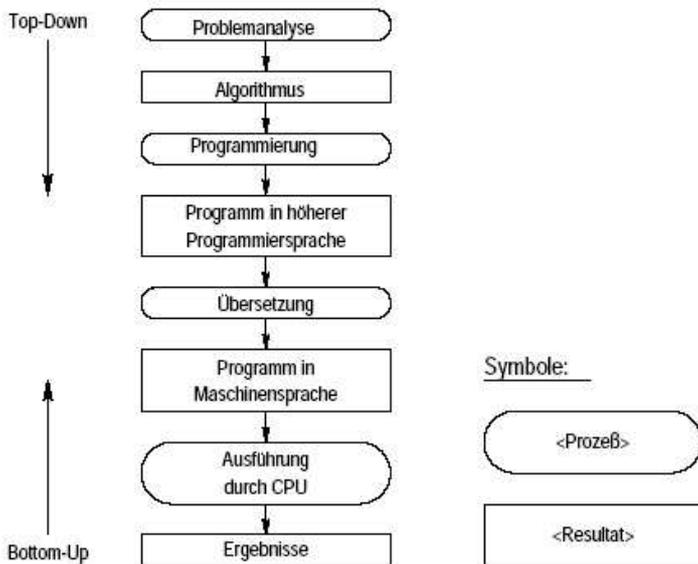


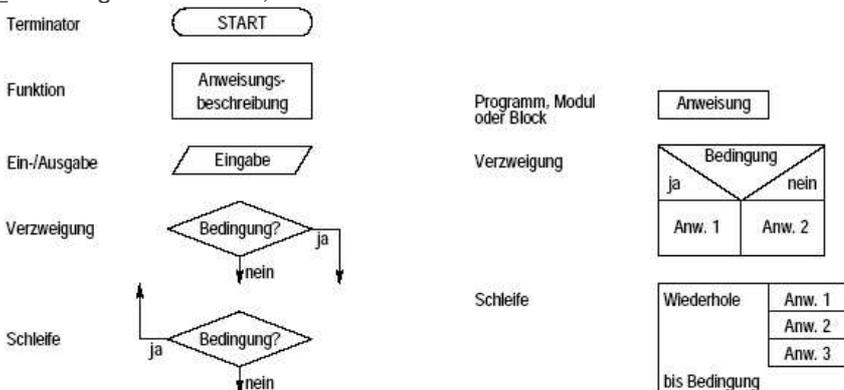
Abbildung 2.2: Stufen der Algorithmenausführung

allgemeine Merkmale von Algorithmen:

- maschinell durchführbar
- benötigte Information muß zu Beginn vorhanden sein)
- allgemeingültig
- keine Einschränkung der Größe der Datenmenge auf die A angewendet wird
- Jeder Schritt muß in seiner Wirkung genau definiert sein.
- muß nach endlichen Zeit (und nach einer endlichen Zahl von Schritten) enden → Abbruchbedingung muss formuliert sein.

Mittel zur Darstellung von Algorithmen:

- _ allgemeine verbale Beschreibung,
- _ Programmablaufplan,
- _ Programmlinienmethode,
- _ Struktogrammtechnik,



(a) Programmablaufplan

(b) Struktogramm

Prozesse:

- interagieren mit ihrer Umgebung → Eingaben entgegennehmen / Ausgaben erzeugen
- unendliche und endlose Prozesse mgl = Hauptmerkmal

warum keine normale Sprache?

- enormes Vokabular, komplizierte grammatikalische Regeln, Verständnis hängt nicht nur von Grammatik ab, auch von Kontext

→ Programmiersprachen

- einfache und knappe Darstellung des Algorithmus in bestimmtem Anwendungsbereich
- für Computer und Mensch leicht verständlich
- Fehlermöglichkeiten bei der Umsetzung des Algorithmus in Programm minimieren

Prozessor

- muss a verstehen und ausführen
 - Symbole erkennen und Bedeutung zuzuordnen (Vorr. Kenntnisse über Vokabular und Grammatik der Sprache)
 - jedem Schritt des Algorithmus eine Bedeutung in Form von Operationen zuzuordnen, die der Prozessor ausführen kann

Syntax:

= Menge der grammatikalischen Regeln, die bestimmen, wie die Symbole in der Sprache korrekt zu benutzen sind

- Programm, das Syntax der Sprache, in der es ausgeführt ist, befolgt= syntaktisch korrekt → Abweichung = Syntaxfehler

Semantik

= Bedeutung besonderer Ausdrucksformen einer Sprache

Prozessor muss in der Lage sein:

1. die Symbole, in denen der Algorithmusschritt ausgedrückt ist, zu verstehen,
 2. dem Algorithmusschritt in Form von Operationen eine Bedeutung zuzuordnen,
 3. die auftretenden Operationen auszuführen.
- Syntaxfehler werden in Stufe 1, bestimmte Semantikfehler in Stufe 2, andere Semantikfehler erst in Stufe 3 festgestellt.

ANSI-C

- Grundvokabular basiert auf mehreren Klassen von Grundsymbolen.(Buchstabe, Ziffer, Sonderzeichen)
- Operatoren
 1. primäre Operatoren → () Klammer, [] Feld, Struktur, Zeiger auf Strukturen
 2. Inkrement und Dekrement → Erhöhung um 1 (x++ oder ++x), Verringerung um 1 (x-- oder --x)
 3. Arithmetische Operatoren → + Addition, - Subtraktion, * Multiplikation, / Division, % Modulo (Divisionsrest)
 4. Zuweisungsoperatoren
 - = einfache Zuweisung x=y
 - += Zuweisung mit Addition x+=y
 - = Zuweisung mit Subtraktion x-=y
 - *= Zuweisung mit Multiplikation x*=y
 - /= Zuweisung mit Division x/=y
 - %= Zuweisung mit Modulo x%=y
 5. Vergleichsoperatoren
 - == Gleichheit
 - != Ungleichheit
 - <= kleiner gleich
 - >= größer gleich
 - < kleiner als
 - > größer als
 6. logische Operatoren
 - && UND-Verknüpfung
 - || ODER-Verknüpfung
 - ! logisches NICHT
 7. weitere Operatoren
 - (typ) Typumwandlungsoperator
 - & Adressoperator
 - sizeof Speicherbedarfsoperator

Bezeichner in C-Programmen

- besteht aus Folge von Buchstaben, Ziffern oder dem Zeichen (erste Zeichen darf keine Zi_er sein)
- beliebige Länge
- keine Verwendung von Schlüsselwörtern

C-Programme = formatfrei → müssen keine bestimmte Zeilenstruktur haben

Bibliotheksroutinen → enthalten Standardbezeichner

Kommentare → dienen besserer Lesbarkeit von Programmen

Syntaxregeln

1. Backus-Naur-Form

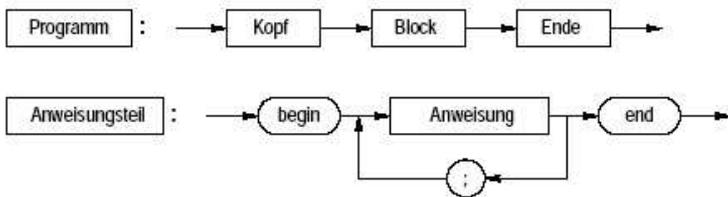
2. Syntaxdiagramme

- graphische Wiedergabe syntaktischen Regeln
- Abbildungsvorschriften :
 - S=knotenmarkierter, gerichteter Graph.

□□ jedes S hat Bezeichnung und zwei ausgezeichnete Knoten (Eingangs- und Ausgangsknoten)

→ Zwei Arten von Knoten:

- Rechtecke = markieren Nichtterminalsymbole.
- Ovale und Kreise = markieren Terminalsymbole.



2.3.3 Informationsdarstellung

- Signale an Rechner → kennzeichnen bestimmten Zustand (high oder low, 0 oder 1, vorhanden oder nichtvorhanden)
- Informationsgehalt eines 0-1 - Zustandes = Bit
- benutztes externe Alphabet durch Transformation in einen entsprechenden Code in ein rechnerinternes Alphabet umgewandelt.

3 Grundsätzliches zum Programmieren

- Aufgabe eines Programms = Manipulation von Daten
- Daten = Programmbereiche, in die durch Befehle des Anwenderprogramms Informationen ein- und ausgetragen werden.
- Datenobjekte = benennbare Speichereinheiten eines bestimmten Datentyps.

Grobstruktur

- _ Include-Dateien, die externe Quellcodedateien einbinden,
- _ Definitionsteil für Konstanten und Makros,
- _ Deklarationsteil für globale Variablen und Konstanten,
- _ Funktionen
- _ Hauptprogramm.

Variable

= Datenobjekt mit veränderlichem Wert

- besitzt Namen und Datentyp
- belegt ab einer bestimmten Adresse einen bestimmten Platz im Speicher
- müssen vor Verwendung deklariert werden.

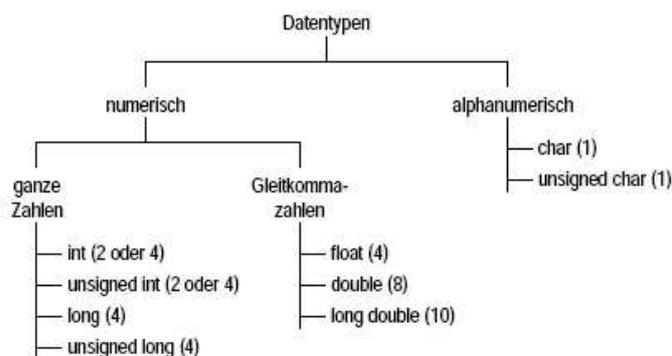
Konstante

= Datenobjekt mit unveränderlichem Wert

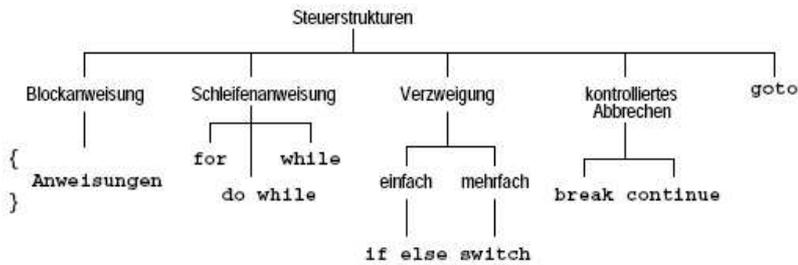
- kann eine Zahl, Zeichen oder Zeichenkette sein.
- Typen : -
 - Ganzzahlkonstanten (int, short, long),
 - Gleitkommakonstanten (float oder double),
 - Zeichenkonstanten (char),
 - Zeichenkettenkonstanten (Feld vom Typ char),
 - Aufzählungskonstanten (enumerators, intern int).

Datentypen

- vereinbaren Speicherplatz für Variablen



Steuerstrukturen



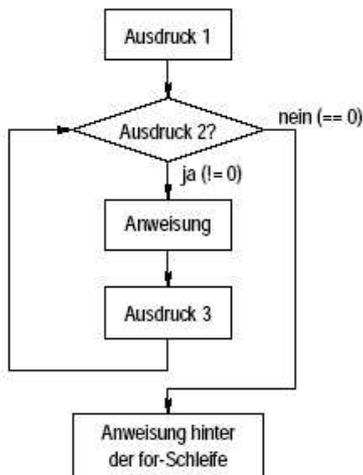
Schleifenanweisungen

= Steuerkonstrukte

- Wiederholung von ganze Teilen oder einzelne Anweisungen im Programm
- Merkmale = Anzahl der Durchläufe und Art der Abprüfung des Abbruch-kriteriums.

1. For-Schleife

- bekannte Dauer



Syntax:

```
for ( Ausdruck1 ; Ausdruck2 ; Ausdruck3 )
```

```
Anweisung ;
```

Ausdruck 1: Initialisierung der Schleifenvariablen

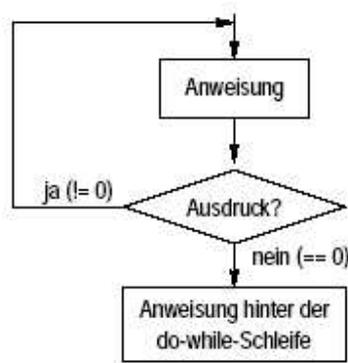
Ausdruck 2: Abbruchkriterium

Ausdruck 3: Reinitialisierung der Schleifenvariablen

2. do-while-Schleife

wird mindestens einmal durchlaufen

- o endgeprüft
- o unbekannte Dauer



Syntax

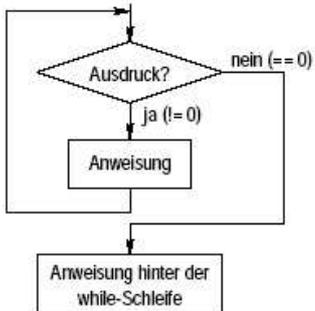
```
do
```

```
Anweisung
```

```
while ( Bedingung ) ;
```

while-Schleife

- anfangsgeprüft
- , unter Umständen abweisend
- unbestimmte Dauer.



Syntax (while):

```
while ( Ausdruck )
```

```
Anweisung ;
```

Verzweigung, Alternative:

- _ Die bedingte Anweisung (if)
- _ die bedingte Anweisung mit Alternative (if else),
- _ geschachtelte bedingte Anweisungen (if else if else),
- _ Mehrfachanweisungen (switch).

Bedingung?	
!=0 (wahr)	==0 (falsch)
Anweisung ausführen	%

if Anweisung

Bedingung?	
!=0 (wahr)	==0 (falsch)
Anweisung 1 ausführen	Anweisung 2 ausführen

if – else Anweisung

if Anweisung = Bedingte Anweisung

- führt Programmteile nur dann aus, wenn eine bestimmte Bedingung erfüllt ist.

Jeder Wert des Ausdrucks, der verschieden von 0 ist oder bei Vergleichsoperationen den Wert wahr liefert, wird auf !0 gesetzt, d.h., da_ die nachfolgende Anweisung ausgeführt wird. Dagegen wird jedes Ergebnis des Ausdruckes mit dem Wert 0 oder falsch auf 0 gesetzt und die nachfolgende Anweisung wird übersprungen.

If/else Anweisung = Bedingten Anweisung mit Alternative

- Anweisungen soll auch ausgeführt werden, falls die Bedingung nicht erfüllt ist

switch-Anweisung = Mehrfachentscheidungen

switch (Variable)

case Konstante 1 : Anweisung 1 ;

case Konstante 2 : Anweisung 2 ;

...

case Konstante n : Anweisung n ;

default : Anweisung x ;

Ausdruck			
Konst. 1	Konst. 2	Konst. n	default
Anweisung 1 ausführen	Anweisung 2 ausführen	Anweisung n ausführen	Anweisung x ausführen

break-Anweisung:

- bewirkt vorzeitige Beendigung einer Anweisungen
- in Schleifen und in switch-Anweisung

continue-Anweisung

- nur in Schleifen zu verwenden
- bewirkt den Abbruch des gerade durchlaufenen Schleifendurchgangs
- gibt nach Reinitialisierung der Schleifenvariablen die Steuerung an den Anfang des folgenden Schleifendurchgangs ab.
- Die Syntax lautet:

goto-Anweisung

= unumgänglichen (unbedingten) Sprung an die Stelle eines Programms, die durch die entsprechende Einsprungsmarke gekennzeichnet ist.

- Syntax: label : Anweisung ; goto label ;

5 Zusammengesetzte Datentypen

- drei Typen (Felder, Strukturen, Unions)

Felder

- fassen Datenelemente gleichen Typs zusammen
- speichern diese unmittelbar hintereinander.
- Typen der Elemente:
 - _ Elementare Datentypen (char, short, int, long, oat, double)
 - _ Felder,
 - _ Strukturen und Unions,
 - _ Zeiger (Speicherung von Adressen von Datenobjekten)
- eindimensional = Vektoren.
- zweidimensional = Tabelle

Eindimensionale Felder

- Im Speicher werden die Elemente eines Feldes hintereinander abgelegt

Typische Operationen: Indizierung, Zugriff und Initialisierung

- initialisierung mit 0: `float zahl[100] = 0.0;`
- elementweise Ausgabe und Eingabe → for-Schleife

Mehrdimensionale Felder

- im Speicher nach dem letzten Index geordnet → z.B. zeilenweise Abspeicherung
- Ein- und Ausgabeoperationen über geschachtelte Schleifen.

Zeichenketten (Strings)

= Zeichenfolgen, die aus Zeichen des darstellbaren Zeichensatzes bestehen.

- können als Variablen abgebildet werden → eindimensionales char-Felder.
- z.B. `char s[13] = "Variable";` die 12 Zeichen und eine Endemarkierung (0) aufnehmen kann.

Operationen auf Zeichenketten

- Initialisierung: `char s[] = "ABER";` oder `s2[4][6] = {"ALPHA", "BETA", "GAMMA", "DELTA"};`
- Ein und Ausgabe: `scanf("%s", name); printf("%s", name);`
- Verwaltung von zweidimensionalen Zeichenketten:

`char names[10][20];` (→ 10 Namen mit bis zu 19 Zeichen)

□ Ein und Ausgabe: `gets(names[0]); puts(names[0]);`
(→ Zeichenkette in das erste Element des Feldes eingelesen, bzw ausgegeben.)

- vordefinierte Funktionen:

→ □ Kopieren: `strcpy(s1, s2)` = String s2 wird in den String s1 kopiert).

→ □ Verkettung: `strcat(s1, s2)` = String s2 wird an s1 angehängt

→ □ . Vergleich `strcmp(s1, s2)` = Beiden Strings werden verglichen. → Ergebnis als int-Wert zurückgegeben. → kleiner 0, wenn $s1 < s2$, gleich 0, wenn $s1 = s2$ und größer 0, wenn $s1 > s2$.

→ □ Stringlänge `strlen(s)` = Liefert Länge der Zeichenkette in Byte, (Anzahl der Zeichen). → abschließendes 0-Byte wird nicht mitgezählt.

Strukturen

= Zusammenfassung verschiedener Einzeldaten (Komponenten) verschiedenen

Typs zu einer Gesamtstruktur

Syntax (struct):

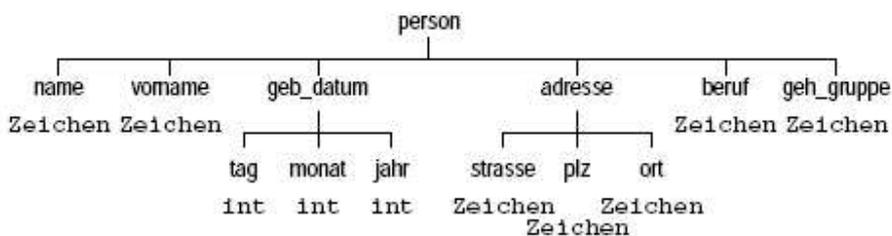
```
struct strukturname {
    typ1 komp_name_1;
    typ2 komp_name_2;
    :
    typn komp_name_n;
} [variablenname1, variablenname2, ...];
```

oder

Syntax (typedef):

```
typedef struct {
    typ1 komp_name_1;
    typ2 komp_name_2;
    :
    typn komp_name_n;
} strukturname;
```

- Definition = Anlegen einer Strukturbeschreibung
- Deklaration = Reservierung von Speicherplatz.



- struct kann wiederum struct enthalten → innere vorher definieren!

Operationen auf Strukturvariablen

- Zugriff ganzheitlich oder über Komponenten (mit Punktoperator) möglich
→ `studentin.name` (greift auf den Namen der Variable Studentin zu).
- Zuweisungen: `strcpy(angestellte.adresse.strasse, "Reuter-Platz");`
- Adress- und Größenoperationen
→ □ können nur auf Komponenten ausgeführt werden
- Einlesen: komponentenweise möglich od Initialisierung (siehe Felder)
- Definieren:

Typ union

- enthält ebenfalls Komponenten, → teilen sich Speicherplatz (Gesamtspeicherplatz entspricht dem der größten Komponente,

Selbstdefinierte Datentypen

- Schlüsselworte: typedef und enum

→ enum :ein echter neuer Datentyp kann kreiert werdñ

→ typedef : liefert neuen Namen für bereits festliegende Bezeichner der bekannten Datentypen

Syntax (typedef):

```
typedef datentyp ersatzname_1 [, ersatzname_2 ...];
```

Syntax (enum):

```
enum name_des_aufzählungstyps {
    name_1, name_2, ... , name_n
} aufzählungsvariable;
```

Aufzählungsvariable → wird wie eine int-Variable behandelt. Sie hat den gleichen Speicherbedarf und es können auch vergleichs- und arithmetische

5.4 Speicherklassen

- Einflußparameter = Definitionsort im Programm und zugeordnete Speicherklasse
- Lokale Variablen → werden in Funktionen definiert, sind auch nur dort gültig
- Globale Variablen → außerhalb von Funktionen (auch außerhalb main) definiert, besitzen ihre Gültigkeit im gesamten Programm ab dem Definitionsort
- Jede Variable besitzt genau eine der folgenden Speicherklassen:
 - _extern, static, auto, register.
- für lokale Variablen alle Speicherklassen möglich → für globale Variablen nur die ersten beiden
- auto → sichert die Speicherung der Variablen für den Zeitpunkt ab, wenn die betreffende Funktion gerade in Bearbeitung ist (Vor und nach Bearbeitung ist der Speicherplatz wieder frei)
- static → für lokale Variablen vorgesehen → Aufrechterhaltung der Belegung des Speichers über die Lebensdauer der lokalen Funktion
- register → Zugriffszeit auf eine Variable wird verkürzt (wird im Speicherbereich des Prozessors gehalten).
- extern → sichert den Zugriff auf globale Variablen aus allen Programmteilen, die nach der Deklaration stehen → Deklaration vor main-Funktion

	Speicherklasse	Gültigkeitsbereich	Lebensdauer
lokal	auto	Block	Laufzeit des Blocks
	register	Block	Laufzeit des Blocks
	static	Block	Laufzeit des Blocks
global	extern	Quelldatei mit Definition und je nach Deklaration alle Module des Programms	Laufzeit des Programms
	static	Quelldatei mit Definition	Laufzeit des Programms

6 Funktionen

6.1 Grundsätzliches

= Programmteil, der aus einer oder mehreren Anweisungen besteht

- können vordefiniert sein (in Bibliotheken
- können auch in n Quellprogramm definiert und deklariert werden
- Funktionsdefinition beinhaltet:
 - _ Speicherklasse der Funktion,
 - _ Ergebniswert der Funktion mit Angabe des Datentyps,
 - _ Name der Funktion,
 - _ Parameter, die an die Funktion übergeben werden mit Namen und Datentyp,
 - _ lokale und externe Variablen, die die Funktion nutzt,
 - _ andere Funktionen, die von der Funktion aufgerufen werden,
 - _ die Anweisungen, die die Funktion ausführen soll.

6.2 Definition und Deklaration

Syntax (Funktionsdefinition):

```

[speicherklasse] [typ] name([d1 n1, ..., dn nn]) { /* Funktionskopf */
  [Definition lokaler Variablen]                /* Funktionsrumpf */
  [Definition externer Variablen]
  [Deklaration weiterer Funktionen]
  [Anweisungen]
}

```

- Funktionskopf = Schnittstelle der Funktion.
- Speicherklasse = extern oder static
- Funktionen dürfen keine weiteren Funktionsdefinitionen enthalten
- Datentypen für Ergebniswerte = einfache Datentypen (char, short, int, long, float, double, long double), Strukturen, Unions, Zeiger
- Parameter → beschreiben Übergabe der Daten in die aufrufenden Funktion
 - Funktionsdefinition = formale Parameter
 - Beim Aufruf = aktuelle Parameter.
 - formale und aktuelle Parameter müssen in Anzahl, Typ und Reihenfolge übereinstimmen
 - keine Übergabe von Parametern → void
- return-Anweisung → beendet die Funktion und gibt die Steuerung an den aufrufenden Programmteil zurück
- Deklaration:
 - vor ersten Aufruf
 - können lokal, global, in verschiedenen Modulen und in include-Dateien deklariert werden
 - Syntax = Funktionskopf mit abschließendem Semikolon = Prototyp
 - Lokale Deklaration → in main deklariert → nur dort lokal verfügbar.
 - Globale Deklaration → Außerhalb des Funktionsblockers, ohne Speicherklassenangabe

Parameterübergabe durch call by value

- Übergabe von einfachen Variablen als Parameter
- es wird von den Parametern eine Kopie angelegt, auf die die Funktion dann lesend und schreibend zugreifen kann.
- Wert des Originals wird dabei nicht verändert
- Vorteil: Parameter nicht veränderbar
- Nachteil: Rückgabe des Ergebnisses nur über return-Anweisung möglich
- Es kann immer nur ein Wert zurückgegeben werden
 - Tauschfunktion über call by value mit lokalen Variablen nicht möglich (zwei rückgabewerte nötig) → nur durch globale Variablen, Verwendung von Strukturen oder Übergabe der Variablen nach dem Prinzip call by reference möglich

Parameterübergabe durch call by reference

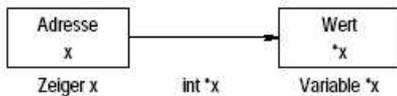
- keine Übermittlung von Werten, sondern Adressen der Datenobjekte, mit denen man direkt arbeiten will
- keine Kopie → Datenobjekte können schreibend verändert werden
- kann bei einfachen Datentypen, Feldern, Strukturen, Vereinigungen und Funktionen angewandt werden.
 - Einfache Variablen → Übermittlung mit Hilfe von Zeigern (deklaration: int * a → zeigt auf Adresse von a)
 - Eindimensionale Felder → generell als Zeiger vereinbart
 - Strukturen → können auch by value) übergeben werden → sehr zeitaufwendig und bedeutungslos
 - Funktionen
 - 3 verschiedene Arten: → durch Aufruf aus einem Hauptprogramm oder Unterprogramm heraus, oder durch Aufruf eines Unterprogrammes als Parameter eines anderen Unterprogrammes
 - 3. durch Definition von Funktionszeigern:
 - Syntax (Definition von Funktionszeigern):**
 - datentyp (*zeigername) ([d1 n1, d2 n2, ..., dn nn]);
 - datentyp bestimmt Rückgabewert der Funktion, die Klammerangaben die Parameterliste → Funktion kann auch mehrere Zahlen zurückgeben
- Selectionsort
 - = Sortieralgorithmus, vom kleinen zum großen
 - selektiert das kleinste Element und tauscht es an seinen Platz
 - nur durch cbr möglich
- Rekursiver Funktionsaufruf
 - = Selbstaufruf. Man spricht in diesem
 - muss eine Abbruchbedingung für die Rekursion besitzen.

7 Das Zeigerkonzept

= Zugriff auf den Speicherplatz einer Variablen über Adressen

- enthalten Anfangsadressen von Variablen, d.h.
- Verwendung:
 - _ direkt mit Adressen rechnen,
 - _ man kann Funktionen mit mehreren Ergebnisparametern organisieren

_ Felder = Zeiger auf das erste Feldelement.
 _ Aufbau dynamische Datenstrukturen



Zeigervariablen

- können auf Datenobjekte beliebigen Typs verweisen
`typ_des_datenobjektes * name_der_zeigervariablen ;`
- Syntax: `<--- Zeigertyp ----> | <--- Variablenname ---->`
- Bsp. `int *zi ;` (→ kann Adresse aufnehmen, hinter der sich ein integer-Datenobjekt *zi befindet)

Operationen auf Zeigern

- zwei Arten:
 1. Zugriff mittels Dereferenzierungsoperator und Zeiger auf Datenobjekte,
 2. Manipulation von Zeigervariablen selbst über die sogenannte Zeigerarithmetik.

Zeigerarithmetik

- Addition einer Konstante n: (Bewirkt Erhöhung der Adresse um n mal der Bytezahl des Typs des Datenobjektes (zs = zs+5;).)
- Inkrementierung: Zeigt die gleiche Wirkung wie Addition von 1 (zs++);
- Subtraktion einer Konstanten n: Verschiebt den Zeiger um das n-fache der dem Typ entsprechende Anzahl von Speichereinheiten zurück (zs = zs-2;).
- Dekrementierung: Bewirkt das Gegenteil der Inkrementierung (zs--);
- Vergleich: Zeiger können auf Gleichheit (==) und Ungleichheit (<, >) verglichen werden.
- Wertzuweisung der Konstante NULL: Bewirkt ein Zeigen ins Leere (zs = NULL;).

Anwendung von Zeigern

Felder und Zeiger

- Zeiger können auf den Feldanfang oder auf Feldelemente gesetzt werden
- Zeiger können zu Zeigerfeldern zusammengefasst werden
- Durchmustern von Vektoren mittels Zeigern:

```

zeiger=vektor ;
for (n=0; n<100; n++)
vektor [n] = n;
    oder
for (n=0; n<100; n++)
*zeiger++ = n;
  
```

Zeichenketten und Zeiger

- Zeichenketten = Vektoren, die Zeichen als Elemente enthalten
- durch Zuweisung von Zeigern kann man Datenobjekt `zk []` einen neuen Inhalt geben
 → ohne die Funktion `strcpy` für das Kopieren zu verwenden
 → durch Umlenken der Zeiger

Dynamische Felder und Zeiger

- Speicherplatz nicht vorher festgelegt, ergibt sich aus Algorithmus
- Vordef. Funktion `malloc` → kann Speicher auf dem Heap (Halde) reserviert
 Syntax: `zeiger = malloc(größe in bytes)`
 (→ `zeiger` = ist eine Zeigervariable, die die Anfangsadresse des Blockes angibt, der durch `malloc` reserviert wird)
- aber : Speicherplatzbeanspruchung compilerabhängig → besser: `dataArray = malloc (100_ sizeof (double)) ;`
- → bei bekannter Elementanzahl und Objektgröße → Benutzung der vordef. `calloc`-Funktion
 → Syntax: `calloc (anzahl, gröÙe)`
- Veränderung der Größe des Speicherblocks mit `realloc` → Syntax: `realloc(zeiger, gröÙe)`
- Freigabe des belegten Speicherplatzes mit `free`. → syntax: `free(Zeiger)`

Zeiger und Strukturen

- Zugriff auf Komponenten der Struktur:
 1. `(*zeiger).strukturkomponente`
 2. `zeiger->strukturkomponente`
- auch dynamische Strukturfelder möglich

Einfach verkettete Listen

- zur Optimierung der Speicherplatznutzung
- speicherplatz wird zusammengesucht → Elemente können verstreut liegen → trotzdem geht Zusammengehörigkeit der Struktur nicht verloren
- Aufbau von Listen
 → Kopf, der nur die Adresse auf das erste Element enthält
 → Endmarkierung, indem der Zeiger des letzten Elementes auf NULL zeigt

```
typedef struct article2 {
    char name[21];
    long num;
    /* Zeiger auf das nächste Listenelement, wobei der Typ der
     * gleiche ist, in welchem der Zeiger next definiert wurde.
     */
    struct article2 *next;
} listenelement;
```

Doppelt verkettete Liste

- bei Bearbeiten von Listen → oftmals angebracht, dass jedes Datenobjekt seinen Nachfolger\ und auch Vorgänger\ kennt = doppelt verkettete Listen
- dafür muss Typendefinition um ein Listenelement erweitert werden

```
typedef struct article2 {
    char name[21];
    long num;
    struct article2 *pre; /* Zeiger sorgt für Verweis auf Vorgänger */
    struct article2 *next; /* Zeiger verweist auf Nachfolger */
} listenelement;
```

- Aufbau: → Anfangszeiger basis, Zeiger für ein neu einzufügendes Listenelement new, einen Zeiger auf das zuletzt einsortierte Listenelement lastin und ein Listenende end
- Für Anlegen und Bearbeiten von doppelt verketteten Listen müssen spezielle Algorithmen genutzt werden

Weitere Anwendungsfälle für Zeiger

- Zeigerfeld: Speichert eine festgelegte Anzahl von Adressen auf Datenobjekte eines Typs
 - Zeiger auf Zeiger: Verweis auf einen Speicherplatz, der eine Adresse auf ein Datenobjekt verwaltet
- Aufbau dynamischer Zeigerfelder möglich

8 Dateiverwaltung

= Zusammenspiel zwischen Speichermedium und Anwendung

- Datei = kontinuierliche Folge von Zeichen oder Bytes → jedes Zeichen bekommt mit 0 beginnend und n-1 endend eine Positionsnummer = Datenstrom
- Anwendungsprogrammierer ist Dateistruktur und Dateiformat des Datenstroms überlassen
- zwei Ebenen der Operation mit Dateien
 1. unteren (low level) Ebene: → elementaren Dateizugriffe. Unmittelbarer Zugriff auf Routinen für Dateioperationen des Betriebssystems → nicht Bestandteil des ANSI-Standards.
 2. oberen (high level) Ebene: → arbeit mit komplexeren Funktionen → in der Standard-Bibliothek verwaltet = nichtelementar Dateioperationen

Textdateien

- Abruf oder Hineinschreiben von Daten auf einem permanenten Speicher aus einem Programm heraus immer über Pufferbereich im Arbeitsspeicher
- Puffer nimmt eine größere Menge von Daten auf, damit nicht jedes Datenobjekt separat zwischengespeichert werden muss.

→ Für dieses Wechselspiel ist strukturierter Typ FILE festgelegt:

```
typedef struct {
    char *buffer; /* Zeiger für die Adresse des Dateipuffers */
    char *ptr; /* Zeiger auf das nächste Zeichen im Puffer */
    int cnt; /* Anzahl der Zeichen im Puffer */
    int flags; /* Bits mit Angaben zum Dateistatus */
    int fd; /* Deskriptor (Kennzahl der Datei) */
} FILE;
```

- Datei stdio.h ist ein Feld aus solchen FILE-Strukturen deklariert. → Anwendungsprogramm: Aufnahme einer logischen Verbindung zur Bibliothek über Zeiger auf eine Strukturvariable vom Typ FILE : file*fz
- beim öffnen einer Datei zur Bearbeitung → Funktion sucht einen Platz im oben erwähnten Feld → Anfangsadresse wird dem Zeiger fz mitgeteilt → Alle Zugriffe erfolgen nun über diesen Zeiger

Öffnen von Dateien

- Funktion fopen → verschafft mit Hilfe des Betriebssystems Zugang zur gewünschten Datei
- Syntax (fopen): FILE * fopen (char * dateiname , char * zugriffsmodus);
- (→ dateiname = Zeiger auf eine Zeichenkette mit Namen der betreffenden Datei)

→ zugriffsmodus = Zeiger auf eine Zeichenkette, die angibt, welche Operationen nach der Öffnung mit der Datei ausgeführt werden können)

→ Ergebnis = Zeiger auf eine Variable des Datentyps FILE

Zugriffsmodus	Ziel der Operation
„r“	Öffnen zum Lesen, fopen==NULL wenn Datei nicht vorhanden
„w“	Öffnen zum Schreiben, Datei wird erzeugt, bereits existierende Datei wird überschrieben und geht damit verloren
„a“	Anfügen am Dateiende, wenn Datei nicht vorhanden, wird sie erzeugt
„r+“	Öffnen zum Lesen und Schreiben, fopen==NULL, wenn Datei nicht vorhanden
„w+“	Öffnen zum Schreiben und Lesen, Datei wird erzeugt, bereits existierende Datei wird überschrieben
„a+“	Öffnen zum Lesen und Anfügen, noch nicht existierende Datei wird erzeugt

Schließen von Dateien:

- Notwendigkeit: 1. Es wird die Verbindung zur Datei gelöst, → FILE-Zeiger wird freigegeben
2. Die Daten des Dateipuffers werden restlos in die betreffende Datei übertragen.
- Bibliotheksfunktion zum Schließen = fclose Syntax: int fclose (FILE * dateizeiger);
→ Parameter = Zeiger auf den Datentyp FILE

8.1.3 Lese- und Schreiboperationen mit Dateien

- durch Funktionen unterstützt
- spezieller Positionszeiger (seek pointer) gibt Stelle der Datei an, wo sie gelesen oder geschrieben werden soll
→ seine Position ändert sich bei jeder Operation
- Funktionen fputc und fgetc → einzelne Zeichen in eine Datei schreiben bzw. lesen:
- Lesen aus einer Datei mit Funktion fgetc: int fgetc (FILE * dateizeiger);
- fgets → Zeichenkette aus einer Datei lesen. → char * fgets (char * pufferzeiger, int anzahl, FILE * dateizeiger) → Als Parameter erwartet die Funktion einen Zeiger auf die Puffervariable, die Anzahl der zu lesenden Zeichen und den Dateizeiger
- fputs → Zeichenkette in eine Datei geschrieben → int fputs (char * pufferzeiger, FILE * dateizeiger);

Direktzugriff

=auf Datenobjekte an einer bestimmten Stelle der Datei zuzugreifen

- vordefinierte FILE-Zeiger
- → am Anfang jedes Programms werden bis zu 5 Gerätedateien geöffnet → sind in stdio.h mit FILEZeiger verbunden sind
 - stdin Standardeingabe meist Tastatur
 - stdout Standardausgabe meist Bildschirm
 - stderr Standardfehlerausgabe meist auch über Bildschirm
 - stdaux Standardzusatz Zusatzgerätee, wie Plotter, Tablett
 - stdprn Standarddruck Drucker

8.3 Binärdateien

- direkte Speicherung von Variablen als Folge der Bits, die den entsprechenden Datentyp abbilden.
- Zugriffsmodi um b ergänzt
- fwrite schreibt in Binärdateien int fwrite (adresse, groesse, anzahl, fhd);
- fread liest Binärdateien. int fread (adresse, groesse, anzahl, fhd);
(fhd ist ein Zeiger auf die FILE-Struktur)
- sizeof → Bestimmung der Bytezahl

9 Objektorientierte Programmierung mit C++

- Ansatzes → Vereinigung von Daten und Algorithmen zur Bearbeitung der Daten → Entstehung abstrakte Datentypen, die zur Beschreibung der Objekte genutzt werden
- Verschiedene Objekte in Beziehungen zueinander zu bringen → Zusammenwirken über Steueralgorithmen
- Oftmals ist die Situation so, da sich das zu realisierende Modell ändert. Bedeutet das
- Vorteil → Reduziert Abstand zwischen Model und Prototyp
- Weiterentwicklungen in Bezug auf f:
 1. De_nition und Einsatz abstrakter Datentypen,
 2. Objektorientierter Entwurf und Programmierung,