PROGRAMMIEREN IN C

Inhaltsverzeichnis

1. Einführung

1.1. Vorteile

- Schnell
- plattform-unabhängig
- flexibel (Speicherzugriff)
- geeignet für Geräte-Treiber, Betriebssysteme und Anwendungen
- weit verbreitet

1.2. Geschichtlicher Abriss

1972: Von Dennis Ritchie entwickelt, um das Betriebssystem UNIX zu programmieren (Plattformunabhängigkeit, weg von Assembler).

1979: Entwicklung von C++ durch Bjarne Stroustrup.

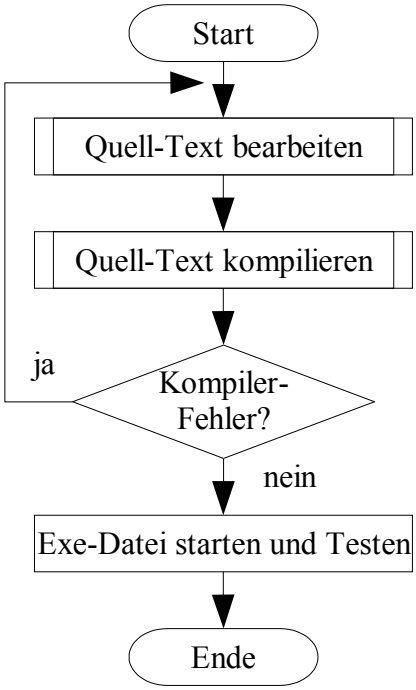
1989: Standardisierung durch ANSI:
ANSI-Standard bzw. C89-Standard
(ANSI = American National Standards Institute)

1998: ANSI/ISO-Standard für C++.
(ISO = International Standards Organisation)

1999: Neuer C-Standard:
C99-Standard

1.3. Arbeitsweise bei der Programmierung

a) Arbeitsschritte



Werkzeuge (=Tools)

- Text-Editor: Bearbeiten des Quell-Textes
  Wünschenswerte Fähigkeiten:
  - Anzeige von Zeilen- und Spalten-Nummer
  - Syntax-Highlighting
  - mehrere Dateien gleichzeitig
- C-Compiler: Quell-Text in Exe-Datei übersetzen
  - gcc für Linux und Windows
  - Visual C++ mit integrierter Entwicklungs-Umgebung; für Windows
  - pcc für DOS
- Debugger: Suchen logischer Programmfehler

1.4. Einfaches C-Programm

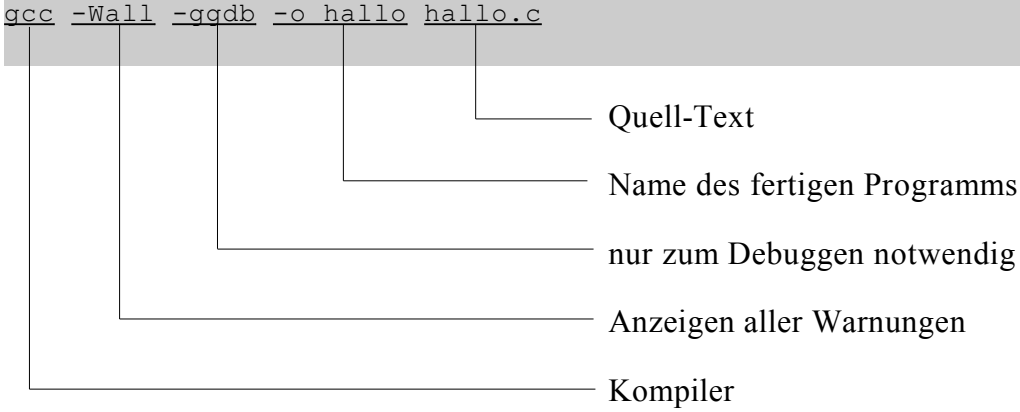
a) Listing (=Quelltext)

Datei: „hallo.c“:

```
#include <stdio.h>            //notwendig für Ein/Ausgabe

int main(void)                //Kopf des Hauptprogramms
{
  printf("Hallo, Welt!\n");   //Ausgabe auf Text-Bildschirm
  return 0;                   //Exit-Code: 0 = kein Fehler
}
```

b) Übersetzung des Programms

```
gcc -Wall -ggdb -o hallo hallo.c
```



c) Ermitteln der Kompiler-Version

```
gcc  -v
```

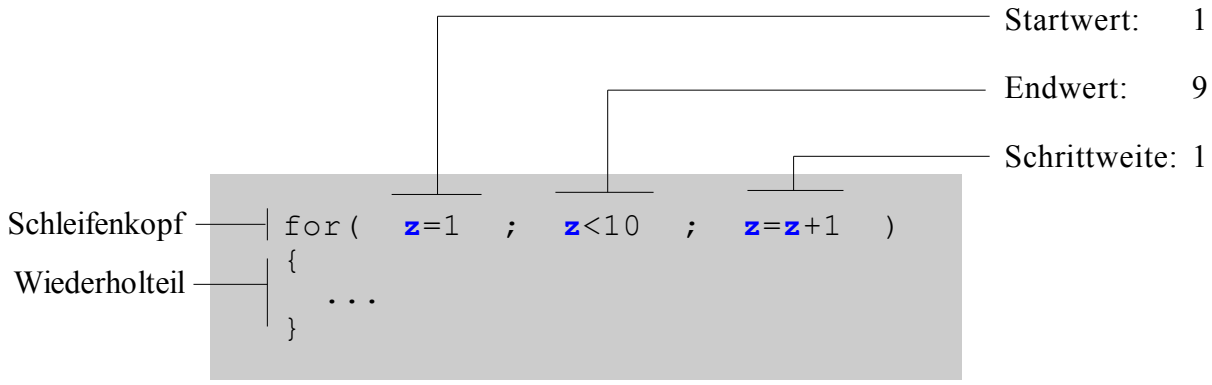
2. Zählschleifen mit for

a) Beispiel

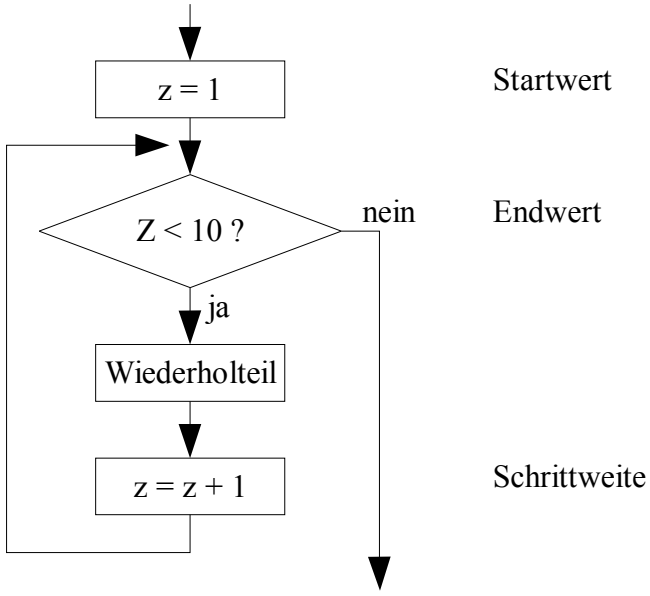
Datei: „zaehler.c“

```
#include <stdio.h>

int main(void)
{
  int z;

  for(z=1; z<10; z=z+1)
  {
    printf("z=%d\n", z);
  }
  return 0;
}
```

b) Die for-Anweisung



c) Ablaufplan



3. Ein-/Ausgabe-Funktionen

a) Beispiel

Datei: „ein_aus.c“

```
#include <stdio.h>

int main(void)
{
  int zahl;

  printf("Gib eine Zahl ein: ");
  scanf("%d", &zahl);
  printf("Deine Zahl war %d!\n", zahl);

  return 0;
}
```

b) printf-Funktion:

Sie dient zur Ausgabe auf den Bildschirm.

```
zahl = 12;
printf("Deine Zahl war %d!\n", zahl);
```

Ersetzen

ergibt:

```
Deine Zahl war 12!
```

c) scanf-Funktion:

Sie dient zum Einlesen von der Tastatur.

```
scanf("%d", &zahl);
```

Einlesen einer Integer-Zahl und speichern in Variable “zahl”.

Bei Eingabe von 17, wird Variable „zahl“ auf 17 gesetzt.

mehrere Zahlen gleichzeitig einlesen:

```
scanf("%d%d", &z1, &z2);
```

d) Format-Anweisungen

Format-Anweisung	*Beispiel*	*Bedeutung*
%d	`printf("%d", 10);`	Ausgabe einer Dezimalzahl
	`scanf("%d", &z);`	Eingabe einer Dezimalzahl
%x, %X	`printf("%x", 10);`	Ausgabe einer Hexadezimalzahl
	`scanf("%x", &z);`	Eingabe einer Hexadezimalzahl
%c	`printf("%c", 10);`	Ausgabe des entsprechenden Buchstabens
	`scanf("%c", &z);`	Eingabe eines Buchstabens
%s	`printf("%s", text);`	Ausgabe eines Strings
	`scanf("%s", text);`	Eingabe eines Strings

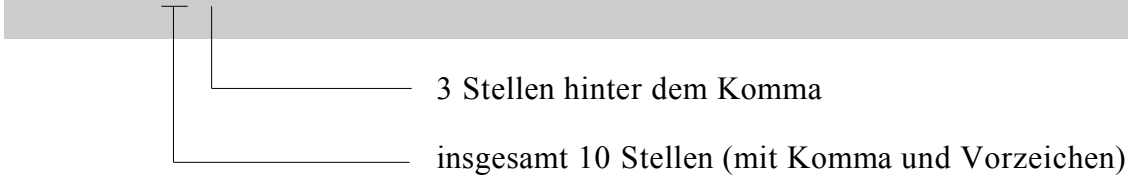
Format-Anweisungen bestimmen bei printf, wie eine Ausgabe aussieht. Bei scanf bestimmen sie, was eingegeben werden muss.

Es gibt noch weitere Format-Anweisungen.

e) Formatierte Ausgabe von Dezimal-Brüchen

Beispiel:

```
float x;
printf("%10.3f\n", x);
```

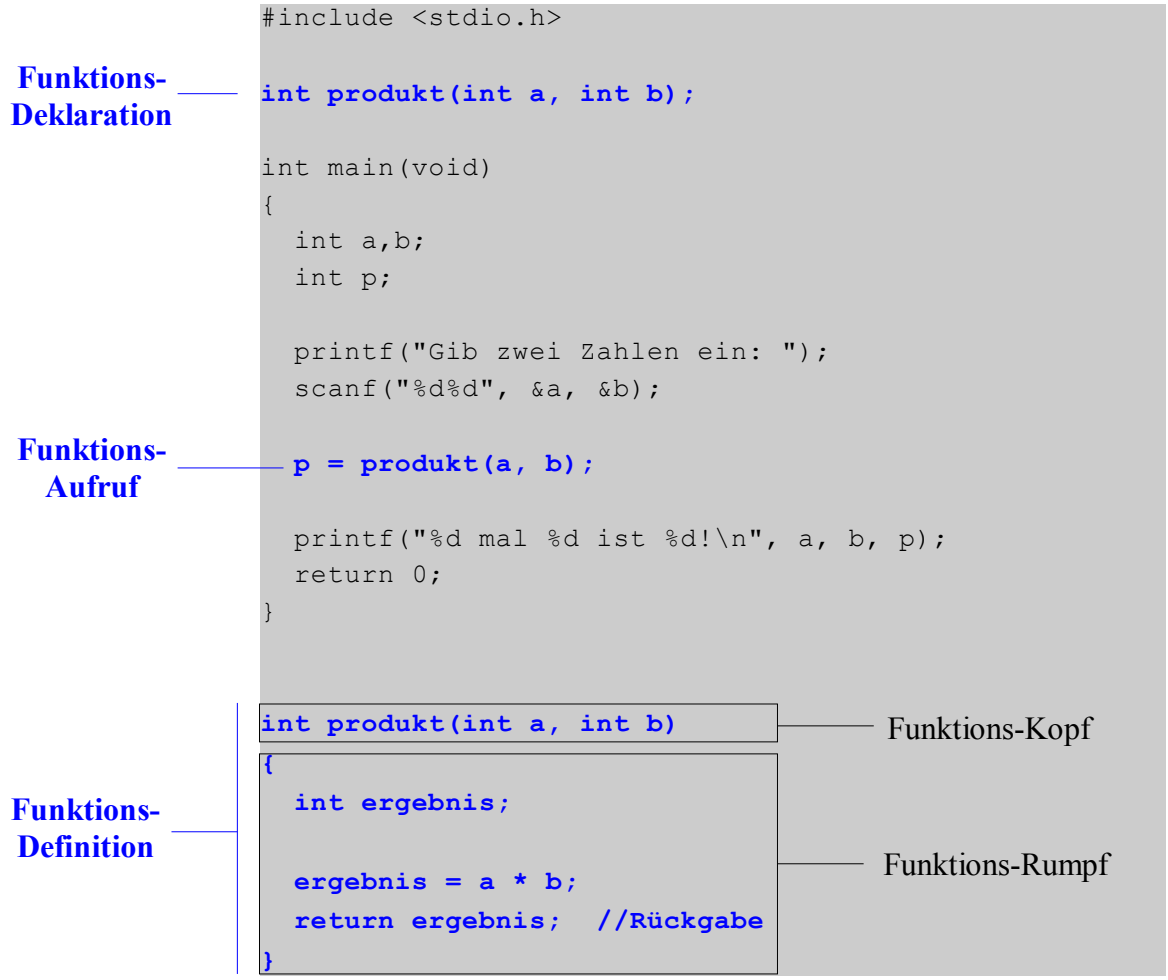


4. Funktionen (=Unterprogramme)

4.1. Syntax (=Schreibweise)

a) Beispiel:

Datei: „multiplizieren.c“

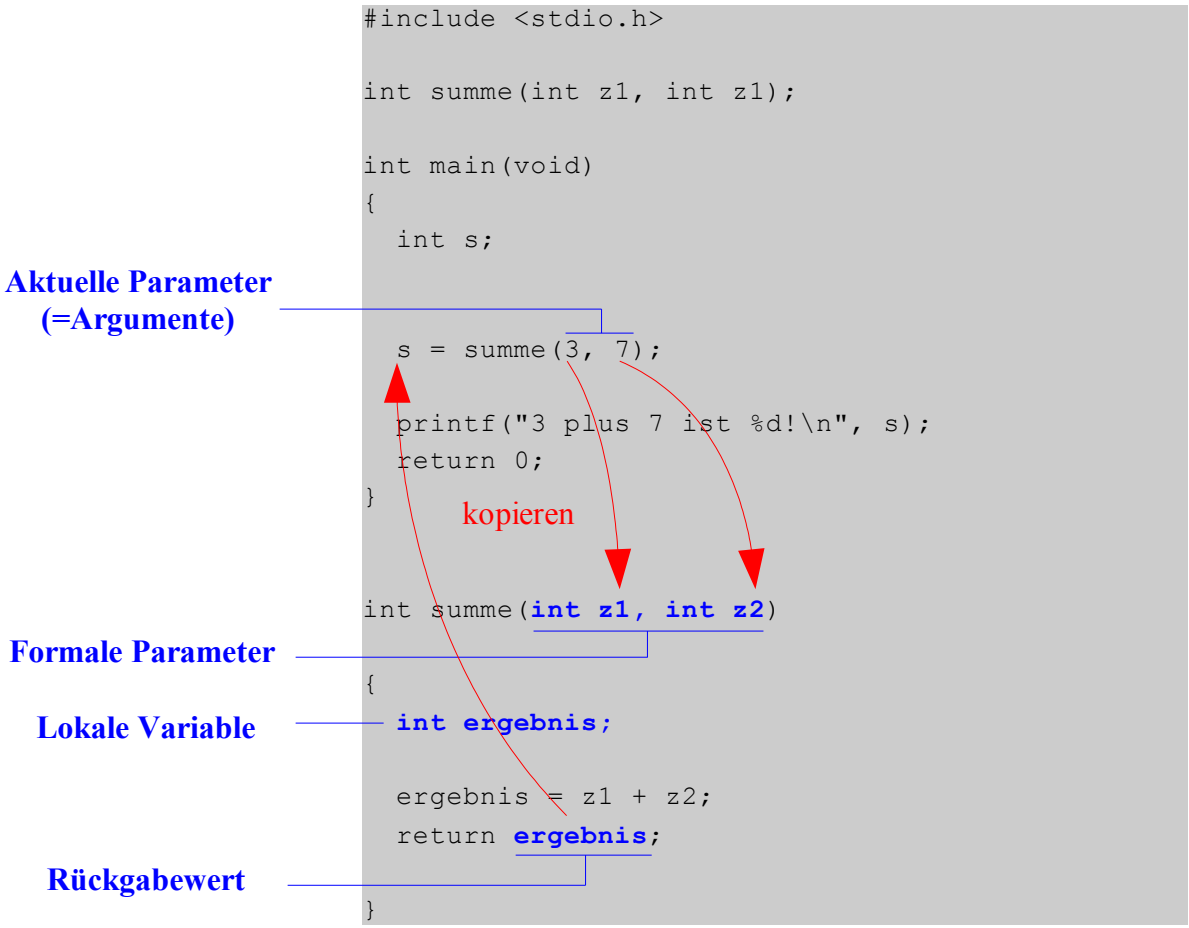


b) Regeln

- Jede Funnktion muss deklariert und definiert werden!
- Die Deklaration ist eine Kopie des Funktions-Kopfes mit abschließendem Semikolon. Sie muss im Programm oben stehen!
- Funktions-Aufrufe dürrfen beliebig oft vorkommen. Sie enthalten keine Datentypen wie int, float, char ...!

4.2. Parameter, lokale Variablen und Rückgabewerte

Datei: „addieren.c“




```c
#include <stdio.h>

int summe(int z1, int z1);

int main(void)
{
  int s;

  s = summe(3, 7);

  printf("3 plus 7 ist %d!\n", s);
  return 0;
}

int summe(int z1, int z2)
{
  int ergebnis;

  ergebnis = z1 + z2;
  return ergebnis;
}
```

- Aufruf der Funktion „summe“:
  - In das Unterprogramm „summe“ springen.
  - Variablen z1, z2 und ergebnis erzeugen.
  - Aktuelle Parameter (3 und 7) in die formalen Parameter (z1 und z2) kopieren.
- Beenden der Funktion „summe“:
  - Rückgabewert (= return value) zurückgeben.
  - Variablen z1, z2 und ergebnis vernichten.
  - Zum Hauptprogramm zurrückspringen.

5. Steuerbefehle

5.1. Einseitige und zweiseitige Verzweigung

a) Beispiel 1:

```
if(z == 3)
  printf("drei\n");
```

Falls z gleich 3, dann Ausgabe „drei“.

b) Beispiel 2:

```
if(x != 7)
{
  printf("nicht sieben: ");
  printf("Leider!\n");
}
else
{
  printf("sieben\n");
}
```

Falls x ungleich 7, dann Ausgabe „nicht sieben: Leider!“ sonst Ausgabe „sieben“.

c) Syntax:

```
if(Bedingung)
{
  Anweisung1;
}
else
{
  Anweisung2;
}
```

d) Mögliche Bedingungen:

Eine Bedingung ist ein Ausdruck, der „wahr“, oder „falsch“ ist. Für solche Ausdrücke stehen folgende Vergleichs-Operatoren zur Verfügung:

Vergleichs-Operator	***Bedeutung***	***Beispiel***
==	gleich	`if(a == b)`
!=	ungleich	`if(a != b)`
<	kleiner als	`if(a < b)`
>	größer als	`if(a > b)`
<=	kleiner oder gleich	`if(a <= b)`
>=	größer oder gleich	`if(a >= b)`

! Vorsicht ! Die Anweisung `if(x = 10)` setzt a auf den Wert 10!
Richtig ist: `if(x ==10)`

e) Logische Verknüpfung mehrer Bedingungen:

Um mehrere Bedingungen logisch zu verknüpfen benötigt man die logischen Operatoren.

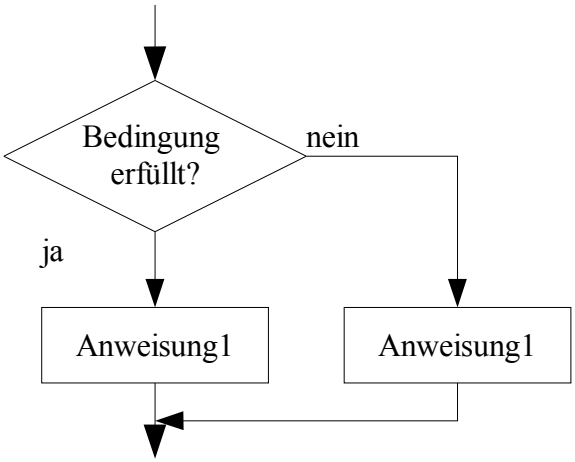
Logischer Operator	***Bedeutung***	***Beispiel***

&&	UND	`if((x>10) && (x<20))`
		`{`
		`  printf("x liegt zwischen 10 und 20!\n");`
		`}`

||	ODER	`if((x==10) || (x==20))`
		`{`
		`  printf("x ist 10 oder 20!\n");`
		`}`

!	NICHT	`if( ! (x == 10))`
		`{`
		`  printf("x ist nicht 10!\n");`
		`}`

f) Ablauf der if-Anweisung in C:



5.2. Mehrfach-Auswahl

a) Beispiel:

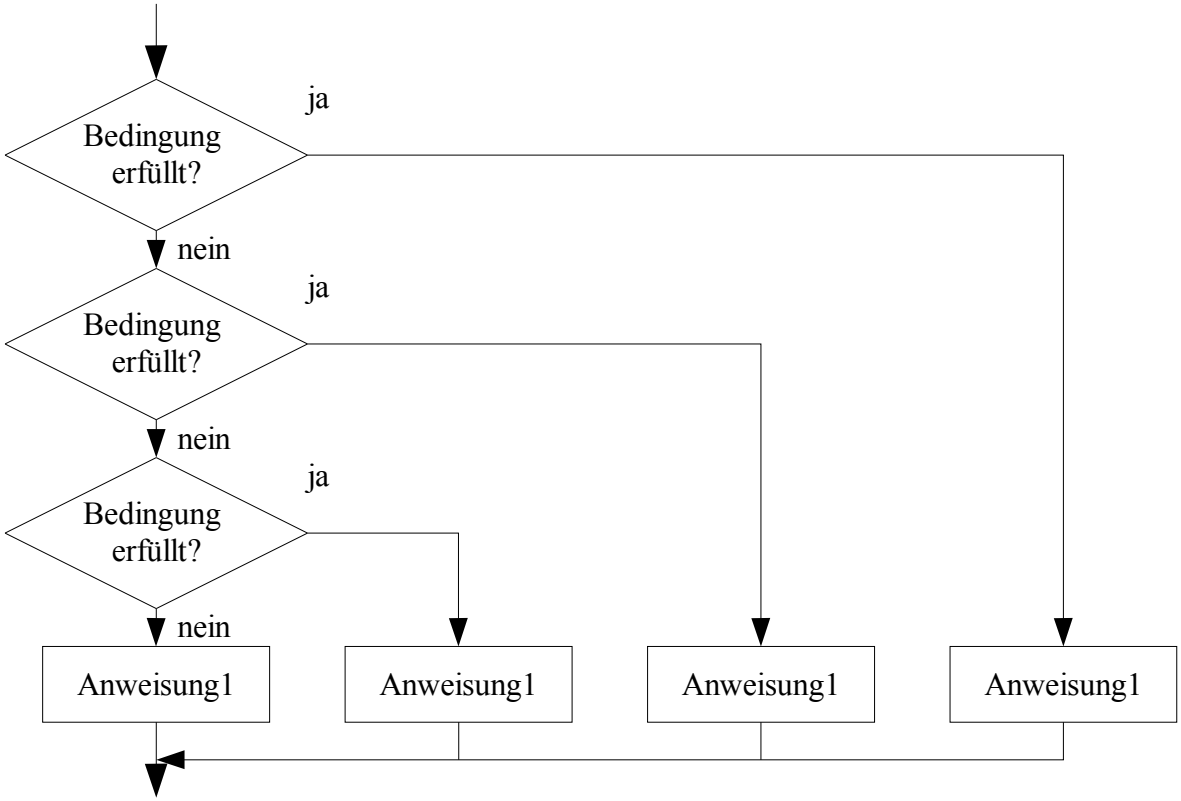
```
int x;
scanf("%d", &xd);
switch(x)
{
  case 3:
    printf(“drei”);
    break;

  case 4:
    printf(“vier”);
    break;

  case 5:
    printf(“fünf”);
    break;

  default:
    printf("etwas anderes");
    break;
}
```

- Schreibt die eingegebene Zahl aus.
- Wenn break fehlt, dann werden alle folgenden Anweisungen mit ausgeführt.

b) Syntax:

```
switch(Ausdruck)
{
  case Konstante1:
    Anweisung1;
    break;

  case Konstante2:
    Anweisung2;
    break;

  case Konstante3:
    Anweisung3;
    break;

  default:
    Anweisung4;
    break;
}
```

c) Ablauf der switch-Anweisung in C:



5.3. Kopfgesteuerte Wiederholung

a) Beispiel:

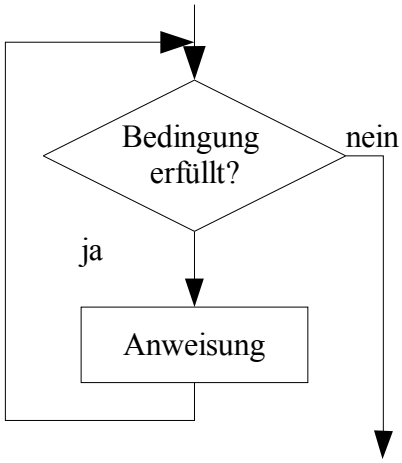
```
int zahl;
zahl = 1;

while(zahl != 0)
{
  scanf("%d", &zahl);
}
```

Die Schleife wird solange wiederholt, bis der Benutzer 0 eingibt.

b) Syntax:

```
while(Bedingung)
{
  Anweisung;
}
```

c) Ablauf der while-Anweisung in C:



5.4. Fußgesteuerte Wiederholung

a) Beispiel:

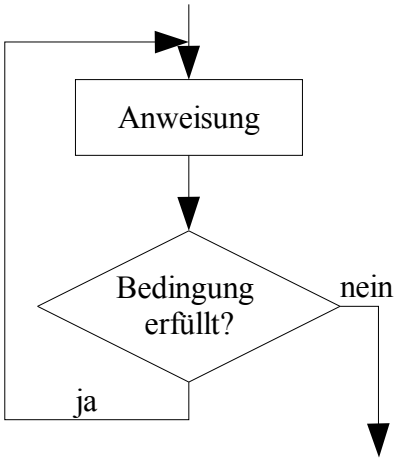
```
int zahl;

do
{
  scanf("%d", &zahl);
} while(zahl != 0);
```

Die Schleife wird solange wiederholt, bis der Benutzer 0 eingibt.

b) Syntax:

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

c) Ablauf der do-while-Anweisung in C:



5.5. Wiederholung mit Abbruchsbedingung

a) Beispiel:

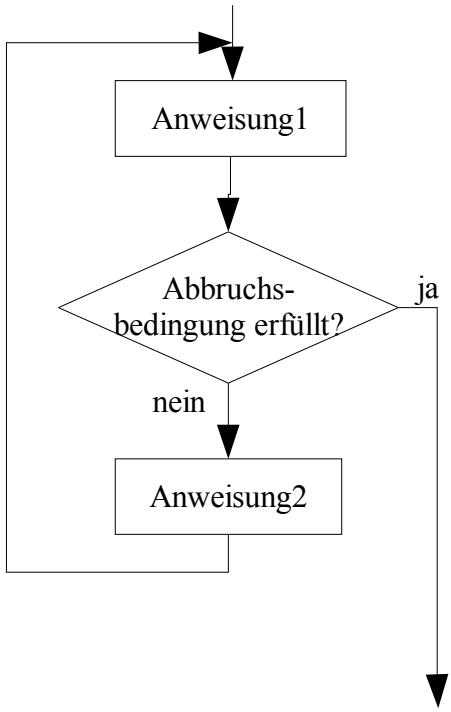
```
int zahl;

while(1 == 1)
{
  scanf("%d", &zahl);
  if(zahl == 0)
    break;
  printf("Die Zahl war nicht Null!\n");
}
```

Die Schleife wird solange wiederholt, bis der Benutzer 0 eingibt.

b) Syntax:

```
while(1 == 1)
{
  Anweisung1;
  if(Bedingung)
    break;
  Anweisung2;
}
```

c) Ablauf der Wiederholung mit Abbruchsbedingung in C:



6. Arrays

6.1. Eindimensionales Array

a) Beispiel

Datei: „zahlen.c“

```
#include <stdio.h>

int main(void)
{
  float  zahlen[4];
  int    i;

  //4 float-Werte einlesen
  for(i=0; i<=3; i++)
  {
    scanf("%f", &(zahlen[i]));
  }

  //eingelesene Werte rückwärts ausgeben
  for(i=3; i>=0; i--)
  {
    printf("%f\n", zahlen[i]);
  }

  return 0;
}
```

Array mit 4 float-Variablen

Zugriff auf Array-Element

b) Belegung des Arrays

Bei Eingabe von -1.3; 2.0; 7.12; 0.25 ergibt sich:

	[0]	[1]	[2]	[3]
zahlen	-1.3	2.0	7.12	0.25

c) Regeln

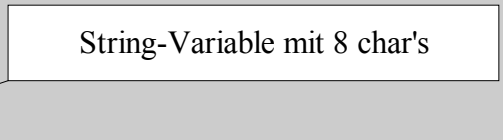
- Das erste Element hat den Index 0.
- Das letzte Element hat den Index Größe-1.

6.2. Strings

a) Beispiel:

Datei: „text.c“

```
#include <stdio.h>

int main(void)
{
  char  name[8];

  printf("Hallo, ich bin HAL, wer bist Du? ");
  scanf("%s", name);
  printf("Hallo, %s\n", name);

  return 0;
}
```



b) Belegung der String-Variblen

Bei Eingabe von „Dave“:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
text	'D'	'a'	'v'	'e'	0			
(ASCII:	68	97	118	101	0)			

c) Regeln

- Strings müssen unter C in char-Arrays gespeichert werden.
- Die Zeichen werden entsprechend dem ASCII-Code gespeichert.
- Jeder String ist mit einer 0 am Ende abgeschlossen.
- Ein char-Array der Größe 8 kann Strings mit maximal 7 Buchstaben oder Zeichen speichern.

6.3. Mehrdimensionale Arrays

a) Beispiel:

```
#include <stdio.h>

int main(void)
{
  int  feld_2d[3][5];
  int  i,k;

  z = 1;
  for(i=0; i<3; i++)
  {
    for(k=0; k<5; k++)
    {
      feld_2d[i][k] = z;
      z++;
    }
  }

  return 0;
}
```

b) Belegung

	[0]	[1]	[2]	[3]	[4]
feld_2d[0]	1	2	3	4	5
feld_2d[1]	6	7	8	9	10
feld_2d[2]	11	12	13	14	15

c) weitere Beispiele:

```
int    a[2][5][5];          //3dimensionales Array
float  x[3][100][10][2];    //4dimensionales Array
```

6.4. Arrays als Funktions-Parameter

a) Beispiel:

Datei: „zahlen.c“

```
#include <stdio.h>

int lies_ein(int arr[]);
void gib_aus(int arr[], int anz);

int main(void)
{
  int zahlen[10];
  int n;

  n = lies_ein(zahlen);
  gieb_aus(zahlen , n);
  return 0;
}

int lies_ein(int arr[])
{
  int  i,n;

  printf("Wieviele Zahlen möchtest Du eingeben? ");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  {
    printf("%dte Zahl? ", i+1);
    scanf("%d", &(arr[i]));
  }
  return n;
}

void gib_aus(int arr[], int anz)
{
  int   i;

  for(i=0; i<anz; i++)
  {
    printf("%dte Zahl = %d\n", i+1, arr[i]);
  }
}
```

Array “zahlen” wird **nicht** kopiert!
“lies_ein()” erhält direkten Zugriff auf Array “zahlen”!

b) Regeln:

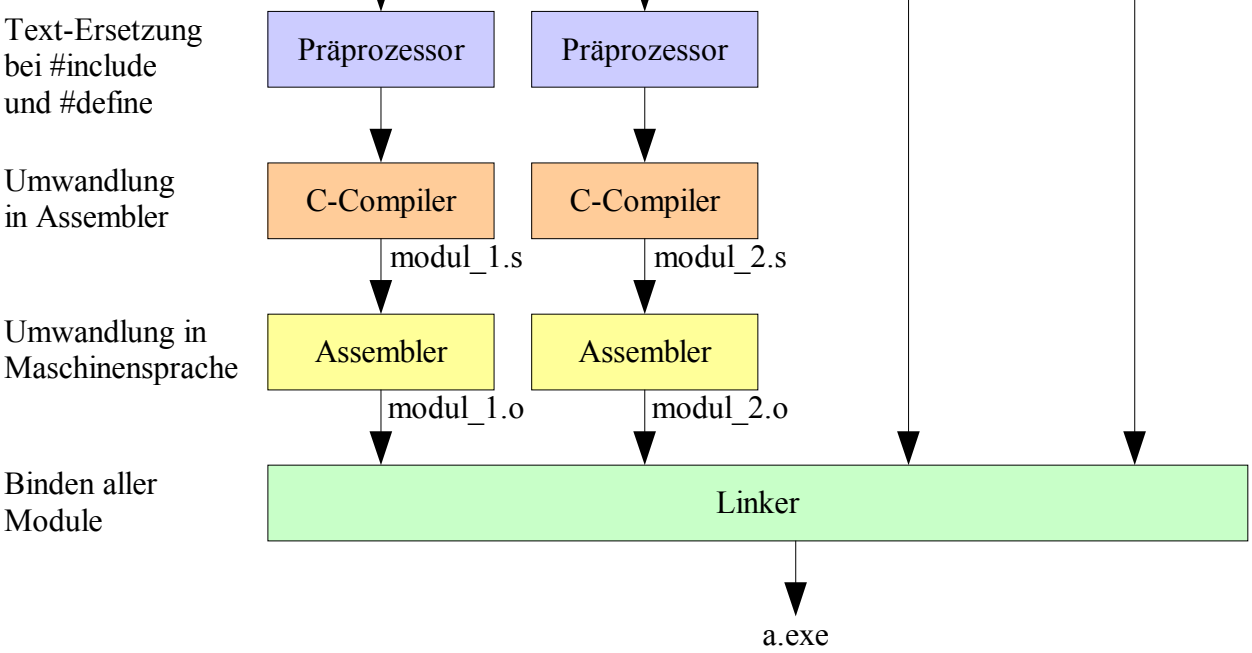
- Array werden beim Funktions nicht kopiert! Es findet kein Wert-Aufruf statt.
- Die Funktion erhält vollen Zugriff auf das Original-Array (=Adress-Aufruf)!

7. Der Übersetzungs-Vorgang

7.1. Kompiler-Aufruf

```
gcc      modul_1.c      modul_2.c      pre.o      -lLib
```

7.2. Ablauf



7.3. Anhalten des Übersetzungs-Vorganges

`gcc modul_1.c -E` = Anhalten nach Präprozessor; Ausgabe auf Standard-Output

`gcc modul_1.c -S` = Anhalten nach C-Compiler; Ausgaben in Datei modul_1.s

`gcc modul_1.o -c` = Anhalten nach Assembler; Ausgabe in Datei modul_1.o

7.4. Übersetzung eines Programms mit mehreren Modulen

Quelldateien: main.c, draw_func.c

Bibliotheken (=Libraries): gdi32 (Datei: libgdi32.a)

Kompiler-Aufruf:

```
gcc main.c draw_func.c -lgdi32
```

8. Dateien

8.1. Durchlesen einer Datei

Programm: „lesen.c“

```
#include <stdio.h>

int main(void)
{
  FILE*  f;                     //File-Handle-Variable
  int    z;

  f = fopen("daten.txt", "r");  //Datei "daten.txt" zum Lesen öffnen
  if(f == 0)
  {
    return -1;                  //Fehler beim Öffnen
  }

  while(!feof(f))               //wiederhole solange Datei-Ende
  {                             //  n i c h t   erreicht
    fscanf(f, "%d", &z);        //eine Zahl aus Datei lesen
    printf("%d\n", z);
  }

  fclose(f);                    //Datei schließen
  return 0;
}
```

8.2. Schreiben in eine Datei

Programm: „schreiben.c“

```
#include <stdio.h>

int main(void)
{
  FILE*  f;                   //File-Handle-Variable
  int    i;

  f = fopen("daten.txt","w");//Datei "daten.txt" zum Schreiben öffnen
  for(i=0; i<10; i=i+1)
  {
    fprintf(f, "%d", i);      //Variable i in Datei schreiben
  }

  fclose(f);                  //Datei schließen
  return 0;
}
```

8.3. Datei-Betriebsarten

Beim Öffnen einer Datei mit „fopen“ wird im zweiten Parameter die Datei-Betriebsart angegeben. Einige der wichtigsten Möglichkeiten sind:

`"r"`	Datei zum Lesen öffnen. Die Datei muss vorher existieren.
`"w"`	Datei zum Schreiben öffnen. Die Datei wird neu angelegt.
`"r+"`	Datei zum Schreiben und Lesen öffnen. Sie muss vorher existieren.
`"w+"`	Datei zum Schreiben und Lesen öffnen. Sie wird neu angelegt.
`"a"`	Datei zum Anhängen öffnen. Sie wird neu angelegt.

Beispiele:

```
f = fopen("file.txt", "w+");
```

Öffnet die Datei „file.txt“ zum Schreiben und Lesen. Die Datei wird neu angelegt, d.h. Eine bereits vorhandene Datei „file.txt“ wird vorher gelöscht.

```
f = fopen("file.txt", "a");
```

Öffnet die Datei „file.txt“ zum Anhängen von Daten. Wenn die Datei vorher existiert, dann bleiben die alten Daten erhalten.

8.4. Positinieren in Dateien

Beispiel 1: 10tes Zeichen der Datei „tabelle.txt“ durch „#“ ersetzen.

Datei: „pos1.c“

```
#include <stdio.h>

int main(void)
{
  FILE* f;
  f = fopen("tabelle.txt", "r+");
  fseek(f, 10, SEEK_SET);
  fprintf(f, "%c", '#');
  fclose(f);
  return 0;
}
```

8.5. Zusammenfassung

`FILE* f;`	= Datentyp für File-Handles
`f=fopen("file.dat", "r");`	= Datei „file.dat zum **Lesen** öffnen
`f=fopen("file.dat", "w");`	= Datei „file.dat zum **Schreiben** öffnen
`fclose(f);`	= Datei schließen (f = File-Handle)
`feof(f);`	= Datei-Ende erkennen (f = File-Handle)
`fprintf(f, "%d", 112);`	= Schreiben in Datei (wie printf)
`fscanf(f, "%d", &x);`	= Lesen aus Datei (wie scanf)