Schleifenanweisungen

Bisher: sequentielle Abarbeitung von Befehlen (von oben nach unten)

Nun: Befehle mehrfach ausführen (= Programmschleife):

`for`-Anweisung - wenn feststeht, wie oft

z.B.: eine Berechnung 10 mal durchführen

`while`-Anweisung - solange eine Bedingung erfüllt ist

z.B.: solange nicht das Ende der Datei erreicht ist, Daten aus der Datei lesen

`do - while`-Anweisung - mindestens einmal ausführen

z.B.: von der Tastatur einlesen, bis der Nutzer eine bestimmte Taste drückt

Die `for` - Schleife

Anwendung: eine Anweisung oder Anweisungsblock eine ganz bestimmte Anzahl mal ausführen.

Syntax:

```
for ( Ausdruck 1 ; Ausdruck 2 ; Ausdruck 3 )
  {
   Anweisungen ;
  }
```

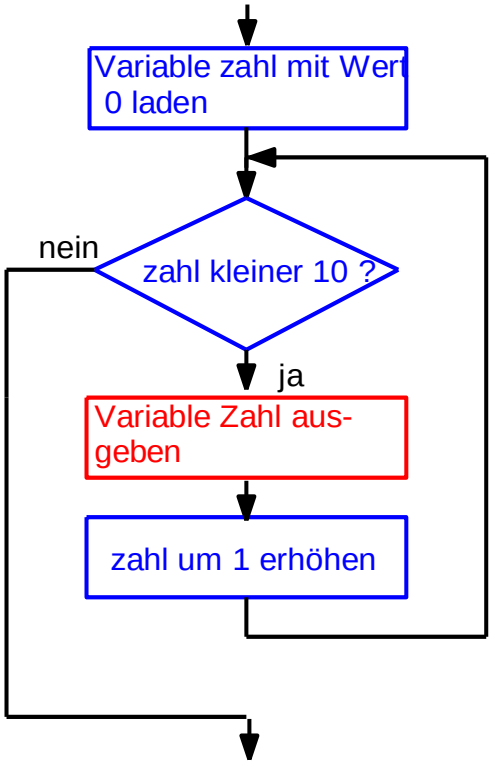
Ausdruck 1 = Initialisierungsausdruck, wird am Anfang einmal ausgeführt

Ausdruck 2 = Abbruchkriterium, die Schleife wird solange durchlaufen, solange der Ausdruck 2 wahr ist.

Ausdruck 3 = dieser Ausdruck wird bei jedem Schleifendurchlauf ausgeführt
normalerweise: einen Schleifenzähler um 1 erhöhen (inkrementieren)

Anweisungen = hier stehen die Anweisungen, die in jedem Schleifendurchlauf ausgeführt werden sollen

Beispiel: die Zahlen 0 bis 9 ausgeben




```
int zahl;

for (zahl=0; zahl < 10; zahl++)

  {

   printf("\n %d ",zahl);

  }
```

Wichtig:

- die Ausdrücke in der for-Klammer durch Semikolon trennen
- nach der for-Klammer kein Semikolon !

Beispiel (fehlerhaft wegen Semikolon !!!):

```
int zahl;
for (zahl=0; zahl < 10; zahl++);
 {
  printf("%d",zahl);
 }
```

ergibt:

10

Erklärung: das Semikolon hinter der `for`-Klammer ist eine “leere Anweisung”.
diese wird 10 mal ausgeführt.
danach: `printf`-Anweisung, gehört nicht mehr zur `for`-Schleife !
`zahl` hat dann den Inhalt “10”.

Die `while` - Schleife

Anwendung: Anweisungen ausführen, solange eine Bedingung erfüllt ist

Syntax:

```
while ( Ausdruck )
  {
   Anweisungen ;
  }
```

Ausdruck = solange dieser Ausdruck wahr ist, wird die Schleife abgearbeitet

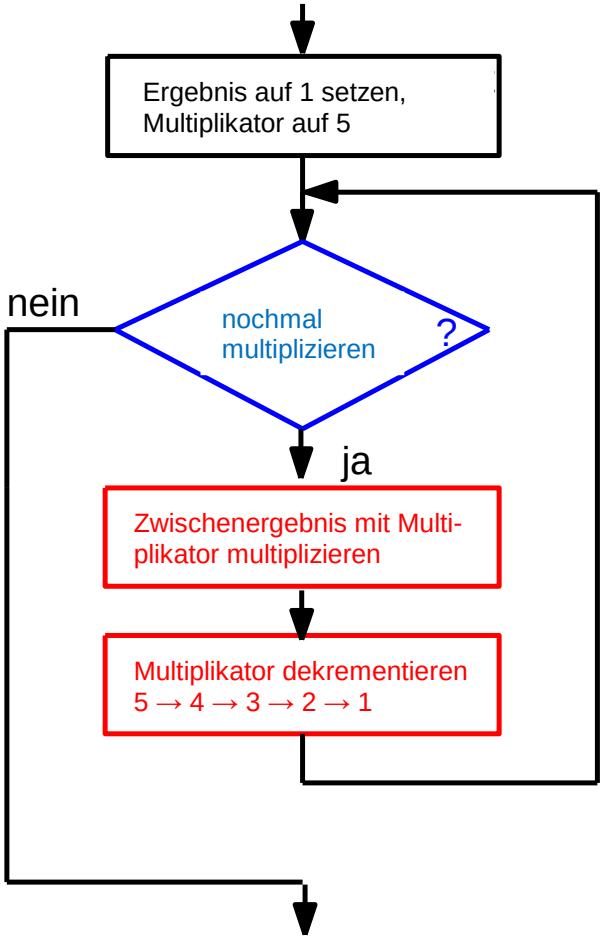
Anweisungen = hier stehen die Anweisungen, die in jedem Schleifendurchlauf ausgeführt werden sollen

wichtig:

wenn der Ausdruck sofort falsch ist, wird die Anweisung überhaupt nicht ausgeführt !

= “abweisende Schleife”

Beispiel: die Fakultät von 5 berechnen: **5! = 1 * 2 * 3 * 4 * 5**




```
int ergebnis = 1;
int n = 5;

while (n > 1)
  {
   ergebnis = ergebnis * n;
   n--;

  }
```

wichtig: die Variablen müssen initialisiert werden, sonst ist ihr Inhalt zufällig

Die **do** - **while** - Schleife

Anwendung: Anweisungen mindestens einmal ausführen und dann solange wie eine Bedingung erfüllt ist

Syntax:

```
do
  {
   Anweisungen ;
  } while ( Ausdruck );
```

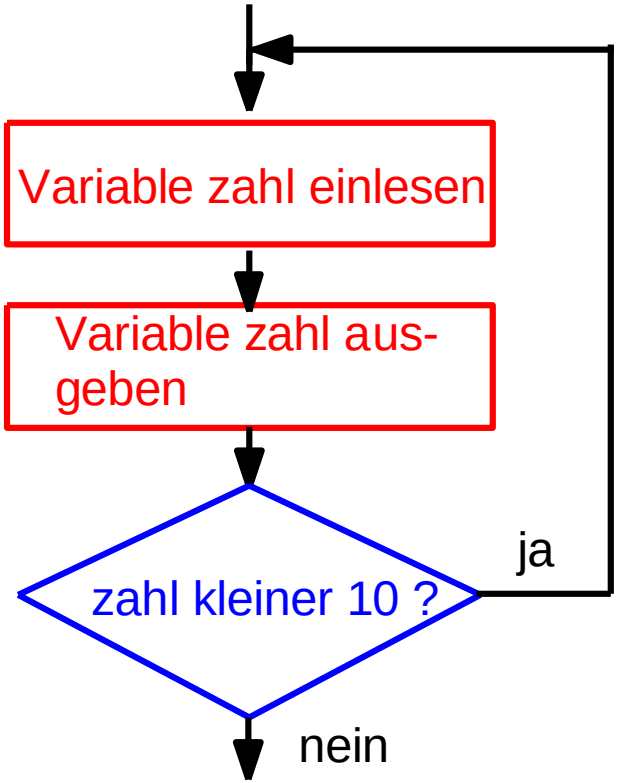
Ausdruck = solange dieser Ausdruck wahr ist, wird die Schleife abgearbeitet

Anweisungen = hier stehen die Anweisungen, die in jedem Schleifendurchlauf ausgeführt werden sollen

wichtig:

die Anweisung wird ausgeführt, bevor der Ausdruck geprüft wird.

Beispiel: Eine Zahl einlesen, bis eine Zahl größer oder gleich 10 eingegeben wird




```
int zahl;

 do
  {

    scanf("%d",&zahl);

    printf("%d",zahl);

 } while (zahl < 10);
```

wichtig: hier muß ein Semikolon hinter die while-Klammer !

Zusammenfassung: Schleifen

Es gibt drei Schleifen in C:

1) **`for`** -Schleife

2) **`while`** - Schleife

3) **`do - while`** - Schleife

prinzipiell kann z.B. eine **`for`**-Schleife eine **`while`**-Schleife ersetzen

Sinnvoll: **`for`**-Schleife, wenn Anzahl Durchläufe bekannt

→ hierfür besonders übersichtlich

sonst: **`while`**-Schleife

Operatoren

bereits bekannt:

`+`	Addition	`-`	Subtraktion
`++`	Zahl um 1 erhöhen (Inkrementieren)	`--`	Dekrementieren
`*`	Multiplikation		
`/`	Division		
`%`	Modulo Operation		

→ das sind “arithmetische” Operatoren

Unterscheidung:

unärer Operator Beispiel: zahl++;

→ wirkt auf einen Operanden (hier: zahl)

binärer Operator Beispiel: zahl + 5

→ wirkt auf zwei Operanden (hier: zahl, 5)

Vergleichsoperatoren

`>`	größer	
`<`	kleiner	
`>=`	größer oder gleich	
`<=`	kleiner oder gleich	
`==`	gleich	(ohne Leerzeichen !)
`!=`	ungleich	

sind binäre Operatoren, benötigen zwei Operanden

Beispiel: `x < 10`

→ Ergebnis dieser Vergleichsoperation ist wieder eine Zahl:

falls: x kleiner 10, dann ist das Ergebnis 1 (= wahr)

falls: x nicht kleiner 10, dann ist das Ergebnis 0 (= falsch)

Merke: die logischen Zustände “wahr” und “falsch” werden codiert:

wahr	entspricht	1	(oder Zahl ungleich 0)
falsch	entspricht	0	

Beispiel:

```
zahl = 5 > 4;
```
in `zahl` steht 1 (wahr)

```
zahl = 5 == 4;
```
in `zahl` steht 0, weil 5 nicht gleich 4 ist (falsch)

```
zahl = 4 * ( 3 == 3);
```
in `zahl` steht 4

Wichtig: Zuweisungsoperator (**=**) und Gleichheitsoperator (**==**) nicht verwechseln !

Beispiel: (fehlerhaft!!!)

```
while (zahl = 4)
    {
    }
```
ist kein Vergleich sondern Zuweisung !!!
daher: Endlos-Schleife

Algorithmus: Eine Summe berechnen

Aufgabe: Lese Zahlen von der Tastatur ein. Addiere die eingegebenen Zahlen so lange auf, bis eine 0 eingelesen wird.

```
int zahl, summe;

summe = 0;

do
  {
   printf("\n bitte geben Sie eine Zahl ein: ");
   scanf("%d",&zahl);
   summe = summe + zahl;
   printf("\n %d + %d = %d \n",summe-zahl,zahl,summe);
  } while (zahl != 0);
```

Die Variable `summe` muß vor der ersten Verwendung auf 0 gesetzt werden !
Die Schleife läuft, solange Zahlen ungleich 0 eingegeben werden
Es wird die Summe vor der Addition und nach der Addition ausgegeben