4. Ablaufsteuerung (Kontrollstrukturen)

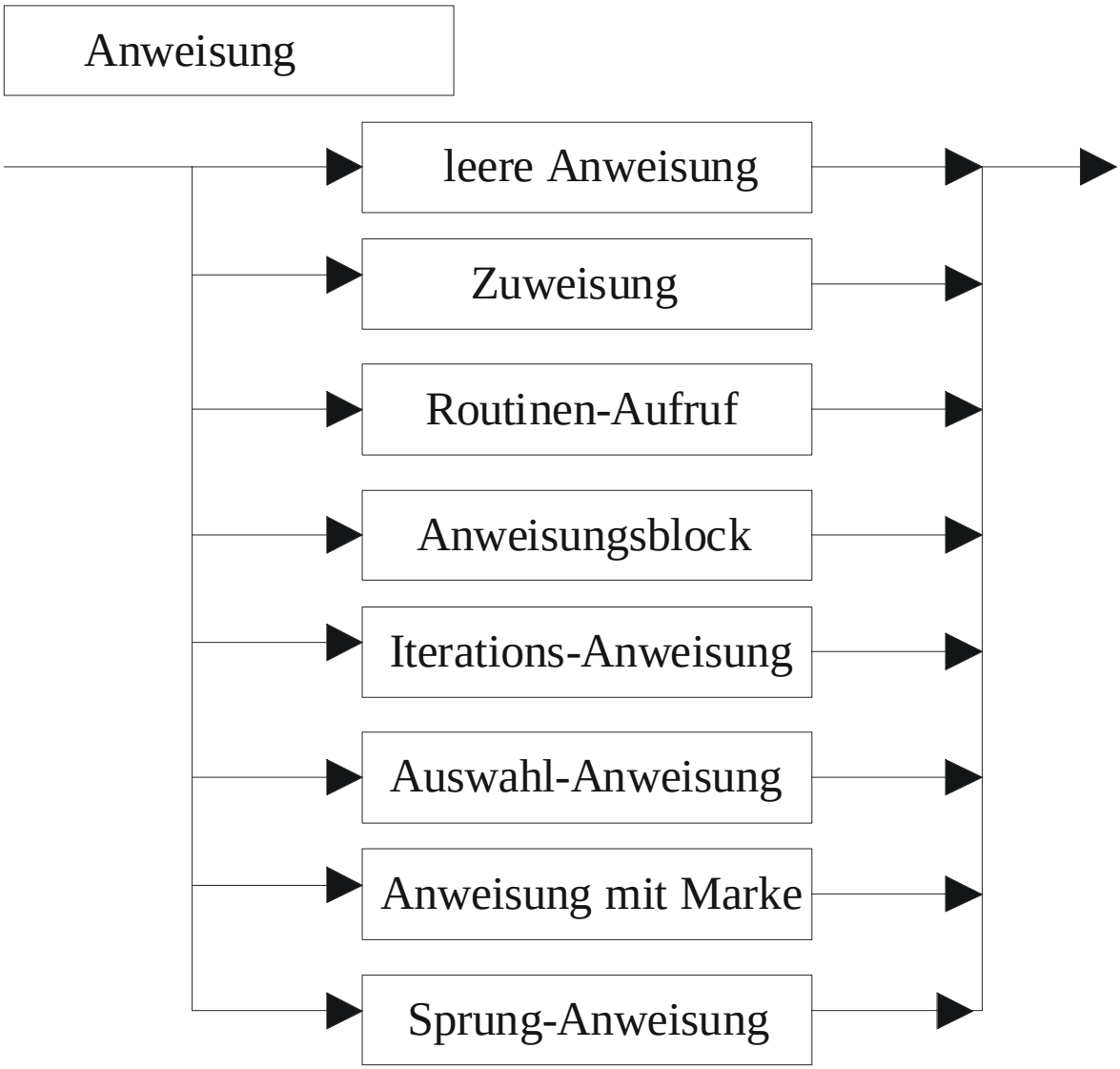
4.1 Anweisungen

4.2 Selektion (bedingte Anweisung)

4.3 Iteration

4.1 Anweisungen

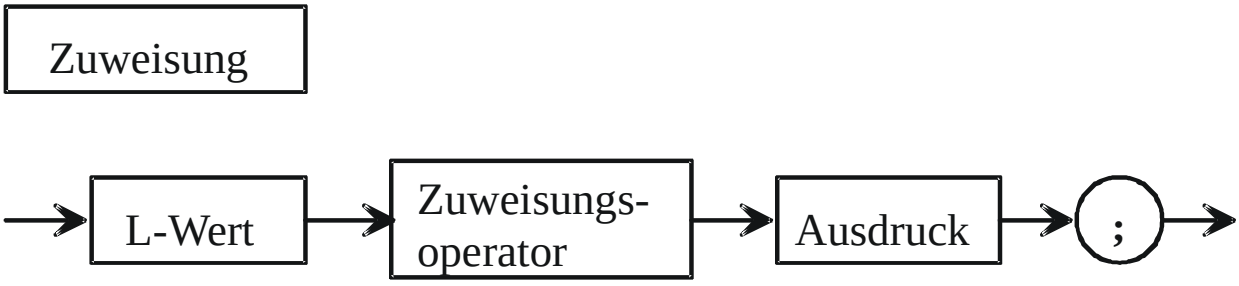
Ein Programm besteht aus einer Folge von Anweisungen.



Wertzuweisung (1)

("Ergibt-Anweisung", assignment statement)

- Berechnung des rechts stehenden Ausdrucks
- Wertzuweisung an die links stehende Variable



Empfehlung:

Der Ausdruck sollte denselben Datentyp haben wie die Variable. Wenn dies nicht der Fall ist, sollte ein explizit typecast durchgeführt werden.

Wertzuweisung (2)

Beispiel

```
summe += 4; /* identisch zu summe = summe + 4; */
produkt *= i;   /* identisch zu produkt = produkt * i; */
```

Dadurch, dass Zuweisungen syntaktisch selbst Ausdrücke sind, sind "Mehrfachzuweisungen" in C möglich!

Beispiel

```
a = b = c = 0.0;
if ((c=getc(datei)) !=  EOF) ...
            ...
```

[Beachte die Klammerung wegen der Vorrangregel]

Die Initialisierung mehrerer Werte und die Verwendung als Operand sind typische Anwendungen.

Wertzuweisung ungleich Vergleichsoperator!

i = i + 1 ist eine Wertzuweisung an i; neuer Wert ist alter Wert + 1

i == i + 1 ist ein boole'scher Ausdruck, der immer `false` ist! Syntaktisch korrekt! Kann ersetzt werden durch 0 (`false`).

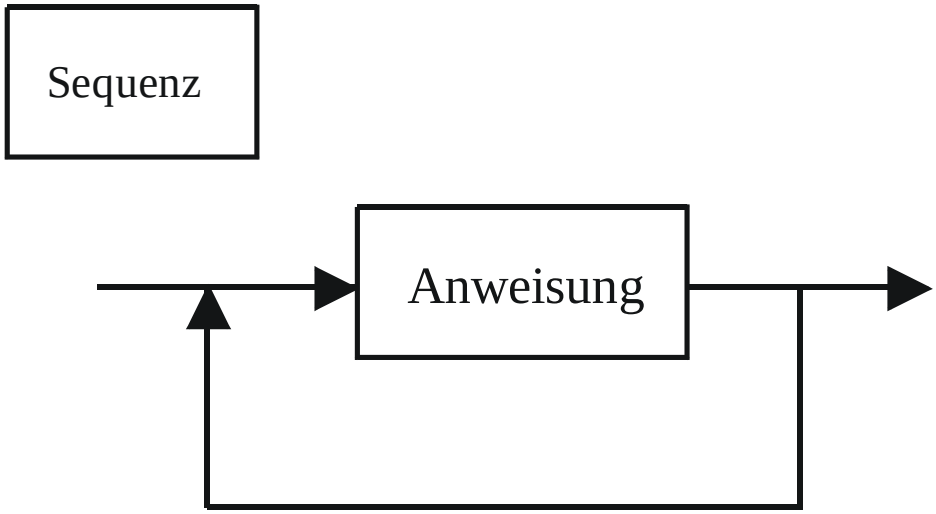
Beispiele für Wertzuweisungen

Deklaration der Variablen:

```
int  i, j, k;
double   x, y, z;
int  fertig;


i = j/k+i;
z = x/y;

fertig = (z < j) && !(x == y);

i = j/x; /* implizite Typkonvertierung */

i = z/x+15;  /* implizite Typkonvert. */

fertig = 1&&x;    /* zugewiesen wird 0 */
                  /* oder 1!           */
```

Folgen von einfachen Anweisungen (Sequenz)

Anweisungen, sequentiell aufgeschrieben, getrennt durch ";"



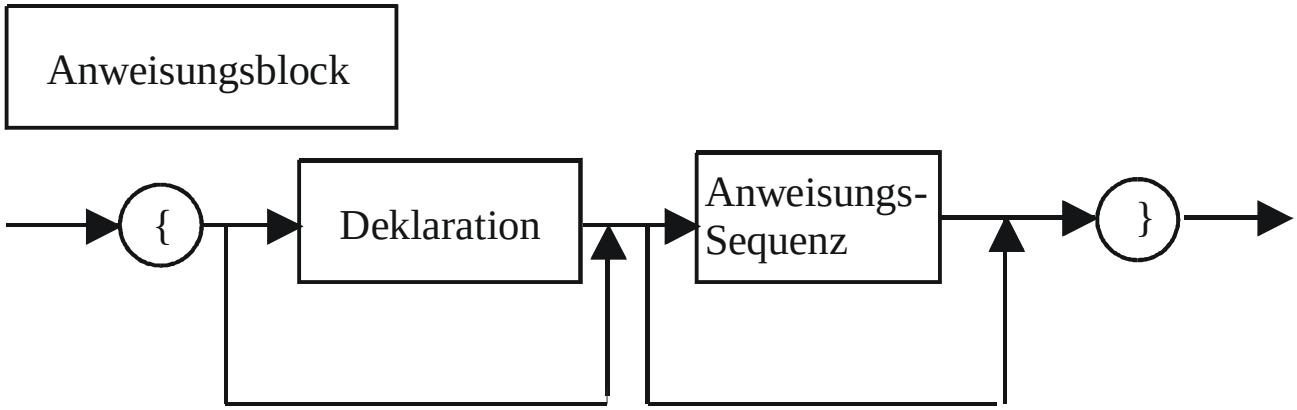
Beispiele

```
a = b; x = y+3; z = 17;
a = 3*87-5;
wahr = (x < 5) || (y > 20);
```

Ein erstes Programm

```
/* Programm : Produkt */

void main ()
{
float x, y, prod;

x = 3.5;
y = 27.8;
prod = x*y;
}
```

Anweisungsblock



Ein Anweisungsblock kann überall da stehen, wo eine einfache Anweisung stehen kann, insbesondere

- in Abhängigkeit von einer Bedingung
- im Rumpf einer Schleife.

Der ausführbare Teil eines C-Programms kann als Anweisungsblock aufgefasst werden. Siehe auch rekursive Verwendung von "Anweisung" im Syntax-Diagramm.

In C kann zu Beginn jedes Anweisungsblocks eine Variabelendeklaration eingefügt werden. Die hier deklarierten Variablen sind jedoch nur innerhalb dieses Blockes gültig!

4.2 Selektion (bedingte Anweisung)

Anmerkung 1

Als "Anweisung" tritt häufig eine zusammengesetzte Anweisung (ein Anweisungsblock) der Form

```
{
        Anweisung;
        Anweisung;
        .
        .
        .
        Anweisung;
}
```

auf, die dann als Ganzes von der Bedingung abhängt.

Anmerkung 2

Eine bedingte Anweisung der Form

```
if    Ausdruck
        Anweisung;
        Anweisung;
else Anweisung;
```

ist syntaktisch falsch!

Geschachtelte if-Anweisungen

Die Zugehörigkeit der else- Klausel zur if-Klausel wird üblicherweise durch Einrückung deutlich gemacht. Die Einrückung ist aber nicht signifikant für den Compiler!

Regel: Ein `else` gehört immer zu dem letzten `if`, das noch keine `else`-Klausel hat.

Beispiel: Knobelspiel

Zwei Spieler geben unabhängig voneinander gleichzeitig je eine nicht-negative ganze Zahl an (etwa durch Ausstrecken von Fingern auf Kommando oder durch verdecktes Aufschreiben).

Nennen beide Spieler die gleiche Zahl, so endet das Spiel unentschieden; andernfalls gewinnt, falls die Summe der genannten Zahlen gerade ist, der Spieler, der die kleinere Zahl genannt hat, und sonst (falls also die Summe ungerade ist) derjenige, der die größere Zahl genannt hat.

Erste Verfeinerung

```
/* PROGRAMM knobelspiel */
void main () {
/* Variablen deklarieren   */
/* ... */
if     (eingabe fehlerhaft)
       {meldung}
else
       {Entscheidung}
}
```

Schrittweise Verfeinerung (1)

Die Entscheidung lässt sich schreiben als

```
if      {eingaben gleich}
        {unentschieden}
else    {ermittle sieger}
```

Der Sieger kann ermittelt werden mit

```
if      {summe gerade}
        {kleinerer siegt}
else    {grösserer siegt}
```

Um anzugeben, welcher der beiden Spieler (der erste oder der zweite) gewonnen hat, kann man den Sieg des Spielers mit der kleineren Zahl beschreiben als

```
if      {erster spieler kleinere zahl}
        {erster spieler siegt}
else    {zweiter spieler siegt}
```

Schrittweise Verfeinerung (2)

Der Sieg des Spielers mit der größeren Zahl wird beschrieben als

```
if      {erster spieler grössere zahl}
        {erster spieler siegt}
else    {zweiter spieler siegt}
```

Programm "Knobelspiel" in Pseudo-Code

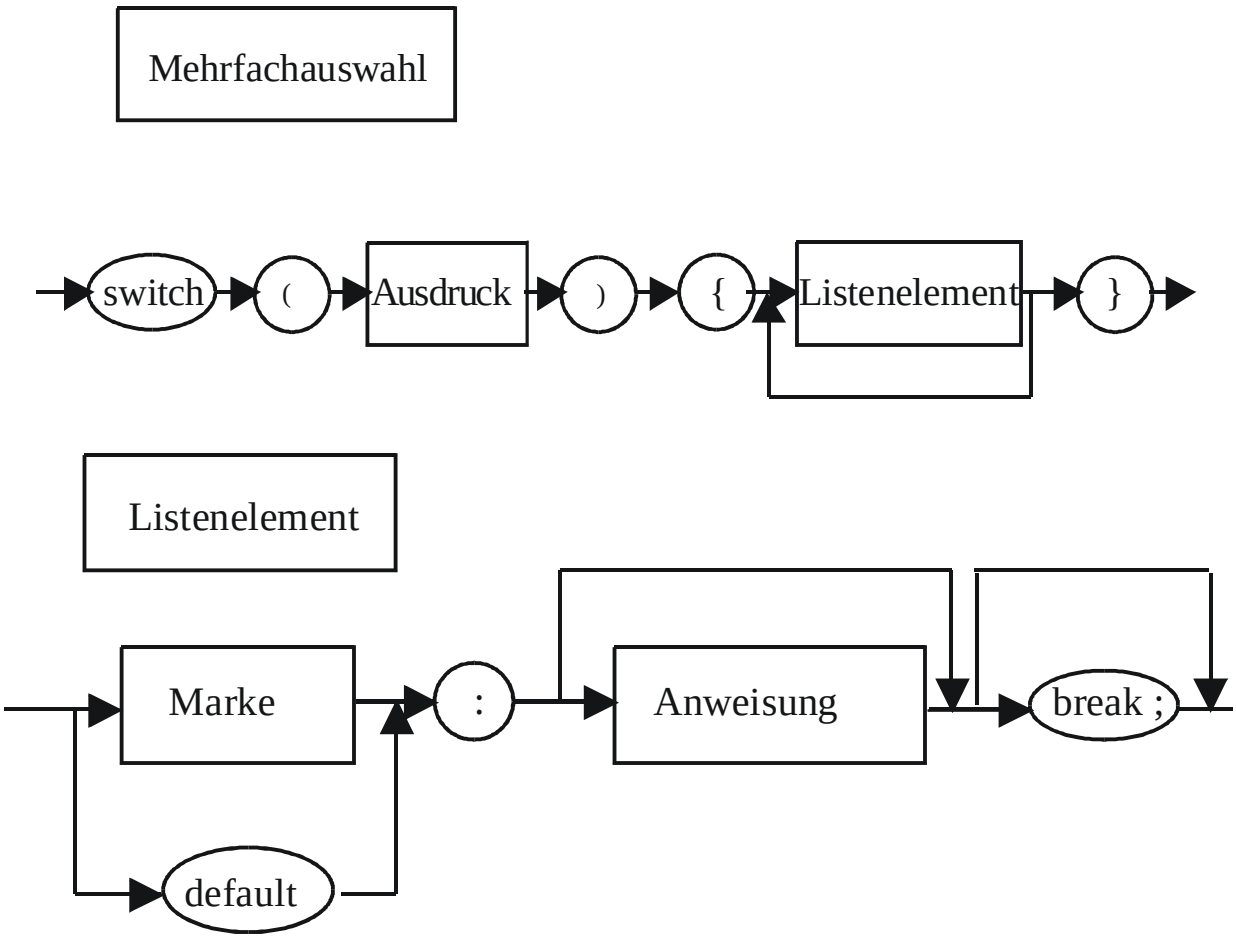
```
/* PROGRAMM knobelspiel */
/* Variablendeklaration  */
void main ()
{ {eingabe}
  if{eingabe fehlerhaft}
    {meldung}
  else
    if{eingaben gleich}
      {unentschieden}
    else
      if {summe gerade}
        if {erster Spieler kleinere zahl}
          {erster spieler siegt}
        else {zweiter spieler siegt}
      else {summe ungerade}
        if{erster spieler grössere zahl}
          {erster spieler gewinnt}
        else{zweiter spieler siegt}
} /* main */
```

Programm "Knobelspiel" in C

```
/* PROGRAMM knobelspiel */
void main ()
{ int k,l;
  printf ("Bitte die beiden Knobelwerte
           eingeben:");
  scanf ("%d \n", &k); scanf ("%d \n", &l);
  /* "call by reference" Erklärung folgt
                                    später */
  if (( k < 0) || (l < 0))
    printf ("unzulässige Eingabe");
  else
    if(k == l)
        printf ("unentschieden");
    else
        if(((k + l) %2) !=0) /* ungerade */
                             /* Summe */
            if(k < l)
              printf ("1. Spieler siegt");
        else printf ("2. Spieler siegt");
    else
        if(k > l)
            printf ("1. Spieler siegt");
        else printf ("2. Spieler siegt");
} /* main */
```

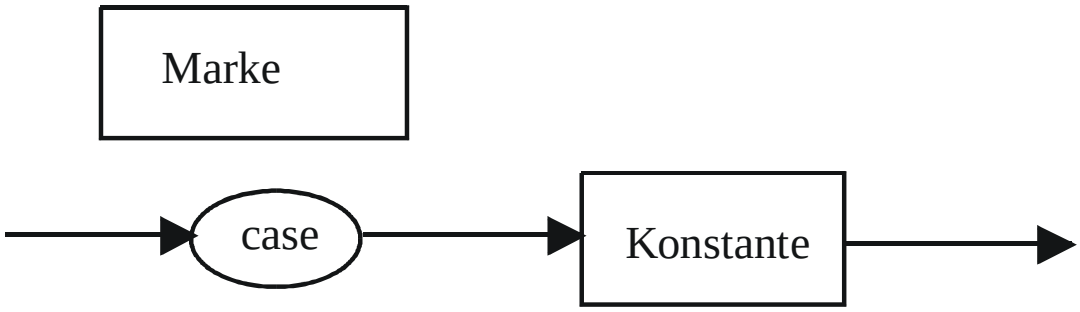
Mehrfach-Selektion (Fallunterscheidung) (1)

Die Mehrfachselektion ist eine Auswahlanweisung für die Auswahl zwischen mehreren Alternativen. Es wird untersucht, ob ein Ausdruck einen von mehreren konstanten ganzzahleigen Werten besitzt, und entsprechend verzweigt.



Mehrfach-Selektion (Fallunterscheidung) (2)

Die `default`-Klausel kann dabei maximal einmal am Ende der Liste verwendet werden.



Beispiel Mehrfach-Selektion (1)

```
void main ()
{
typedef enum {sieben, acht, neun, zehn,
bube, dame, koenig, as} spielkartentyp ;

spielkartentyp karte;
Int wert;

switch (karte)
    {
    case sieben   :   wert =  7; break;
    case acht     :   wert =  8; break;
    case neun     :   wert =  9; break;
    case zehn     :
    case bube     :
    case dame     :
    case koenig   :   wert = 10; break;
    case as       :   wert = 11; break;

    default       :   wert =  0;
    }
}
```

Beispiel Mehrfach-Selektion (2)

Merke:
Die Anweisungen in einem `switch`-Block werden sequentiell abgearbeitet. Die `break`-Anweisung bewirkt ein Verlassen des Blocks. Einen `case`-Block ohne eine `break`-Anweisung abzuschließen bewirkt, dass alle darauf folgenden `case`-Blöcke bis zu einem `break` oder dem Ende der `switch`-Anweisung ausgeführt werden.

Beispiel

```
int x = 0, y;

switch (x) {
       case   0 : y = 1;
       case   1 : y = 2; break;
       case   2 : y = 3; break;
       default          :
}
```

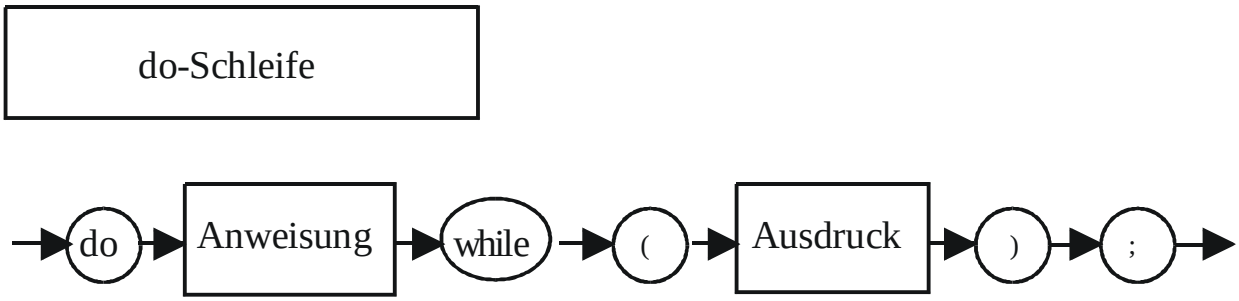
Der Programmierer wollte für den Fall x = 0 der Variablen y den Wert 1 zuweisen. So aber erhält y den Wert 2!

4.3 Iteration (Schleife)

Es gibt drei Varianten von Iterationsanweisungen:

- do-Schleife
- while-Schleife
- for-Schleife

do-Schleife

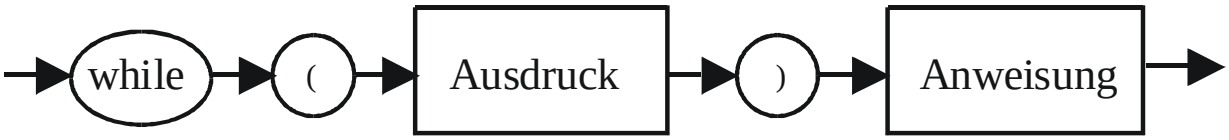


Merke:

Die Abbruchbedingung wird jeweils **nach** der Ausführung der Anweisungen geprüft.

while-Schleife

Bedingungsschleife

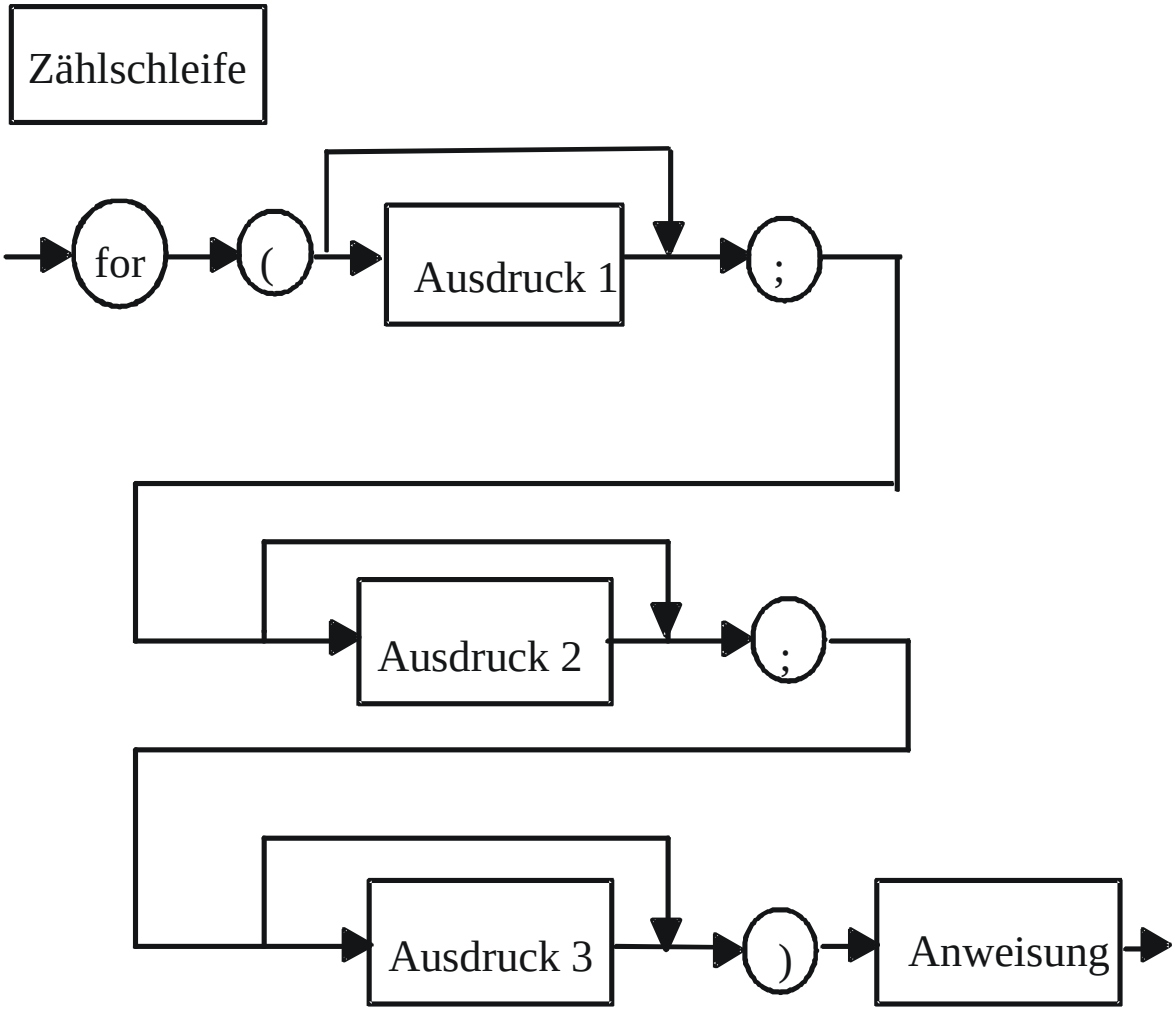


Merke:

Die Abbruchbedingung wird jeweils **vor** der Ausführung der Anweisungen geprüft.

for-Schleife ("Zählschleife") (1)

Die for-Schleife sieht allgemein wie folgt aus:



for-Schleife ("Zählschleife") (2)

- Ausdruck1 wird nur einmal berechnet und dient zur Initialisierung der Schleife (in der Regel Anfangszuweisung an Kontrollvariable).
- Ausdruck2 ist in der Regel eine Abbruchbedingung. Sie wird vor jedem Schleifendurchlauf berechnet. Die for-Schleife bricht ab, falls Ausdruck2 = 0 ist.
- Ausdruck3 wird nach jeder Wiederholung berechnet und gibt deshalb eine Reinitialisierung für jeden Durchlauf an (in der Regel eine Kontrollvariablen-änderung).

Analogie zwischen for- und while-Schleife

Eine for-Schleife der folgenden Syntax:

```
for (expr1; expr2; expr3)
      Anweisungsblock;
```

ist äquivalent zur folgenden while-Schleifen-Konstruktion:

```
expr1;
    while (expr2) {
        Anweisungsblock;
        expr3;
    }
```

Die for-Schleife ist immer dann sinnvoll, wenn die Anzahl der Wiederholungen im Voraus bekannt ist.

Beispiel für eine for-Schleife

```
main()
{
    int kalt, warm, temp;
    printf ("Bitte Ober- und Untergrenze
    für Temperatur eingeben (in Grad \
    Celsius)");
    scanf ("%d %d", &warm, &kalt);
    printf ("Celsius Fahrenheit \n");
    for (temp = kalt; temp < warm, temp++)
      printf ("%d %d\n", temp, (9*temp/5+32));
}
```

`continue` in Schleifen

`continue` bewirkt, dass der Schleifenblock nicht weiter abgearbeitet wird, sondern sofort mit dem nächsten Schleifendurchlauf begonnen wird.

Beispiel

```
#define N 10
int i, a[N];

for (i=0; i < N; i++)
{  /* negative Elemente werden nicht */
   /* bearbeitet */
   if (a[i] < 0)
      continue;
   /* Bearbeitung positiver Elemente */
   ...
}
```

`break` in Schleifen

`break` bewirkt, dass die ganze Schleife verlassen wird.

Beispiel

Im obigen Beispiel würde `break` an der Stelle von `continue` bewirken, dass beim ersten negativen Element die Bearbeitung des gesamten Arrays abgebrochen wird.

`while`-Schleife mit Abbruch in der Mitte

Man könnte sich auch eine Iteration vorstellen, bei der die Abbruchbedingung an beliebiger Stelle zwischen Schleifenbeginn und Schleifenende steht:

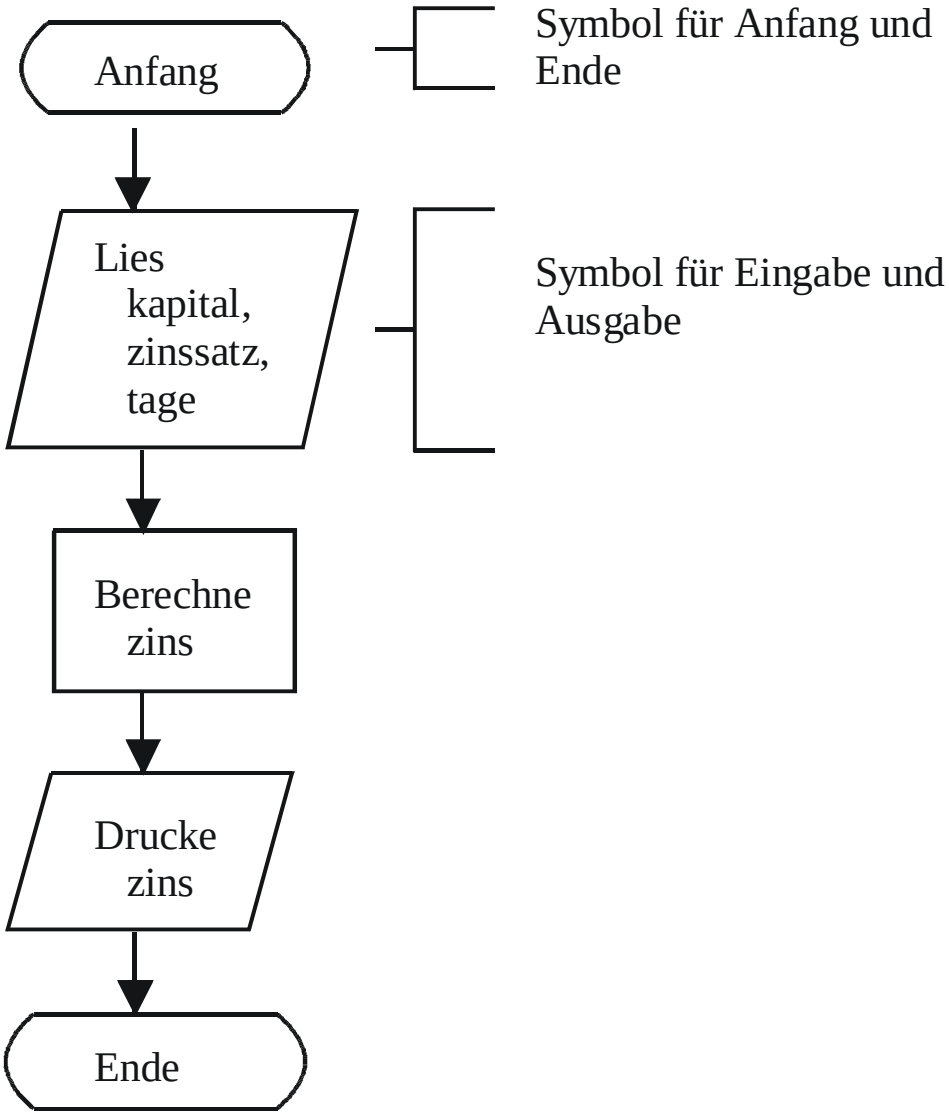
beginloop
    statement; statement;
    if (expr) then quitloop;
    statement; statement
endloop;

Eine solche Anordnung der Abbruchbedingung ist in C mit der `while`-Schleife möglich:

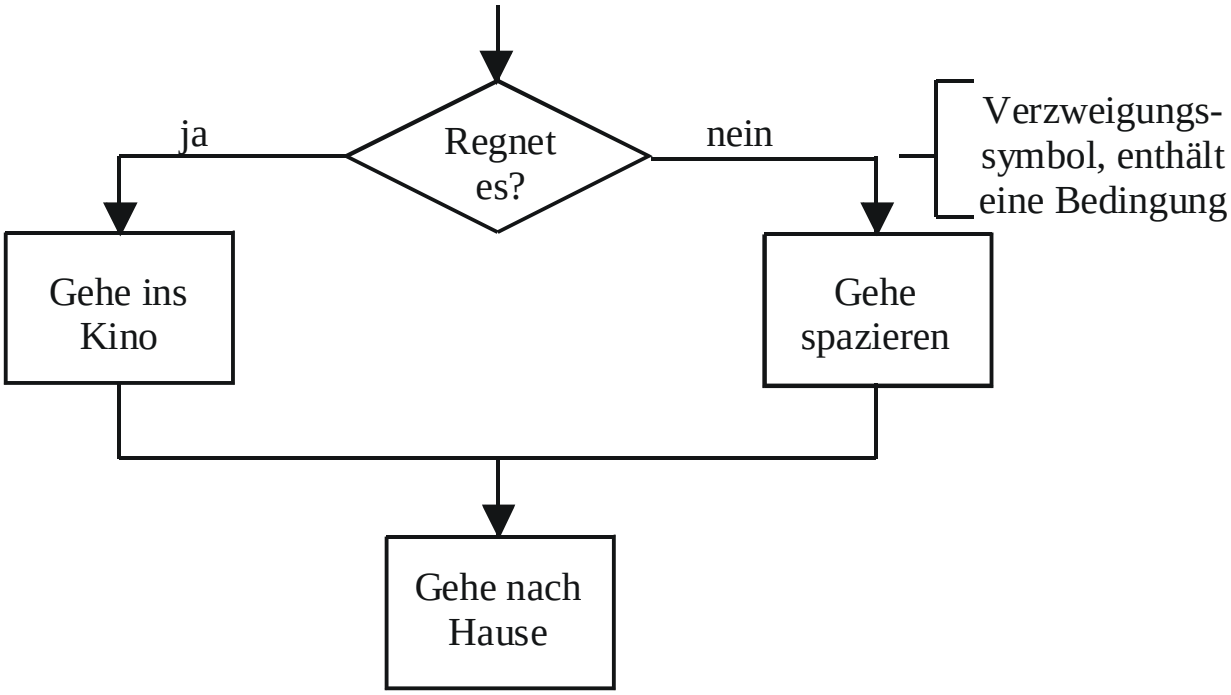
```
while (1) {    /* entspricht  while (true) */
        Anweisung; Anweisung;
        if (Ausdruck) break;
        Anweisung; Anweisung;
        }
```

Flussdiagramm (Programmablaufplan)

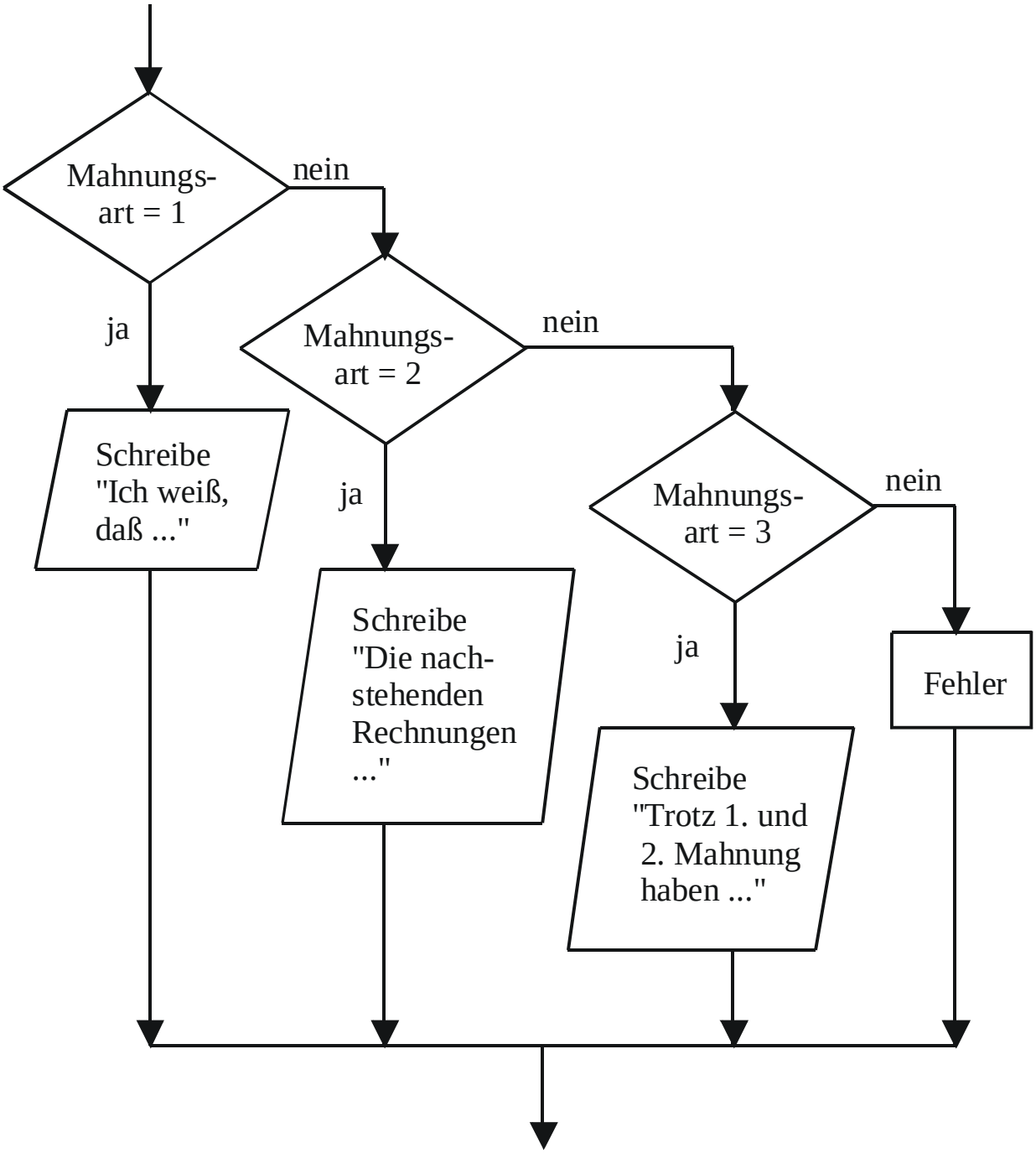
Ein Hilfsmittel zur Visualisierung von Abläufen (Kontrollstrukturen). Es gibt genormte Symbole für verschiedene ausführbare Anweisungen.



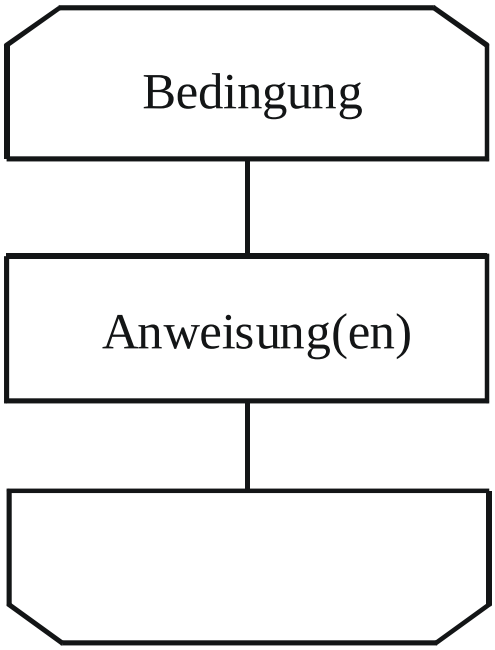
Selektion im Flussdiagramm



Mehrfach-Auswahl im Flussdiagramm



Iteration im Flussdiagramm



Die Bedingung wird vor der Ausführung der Schleife geprüft. Das Diagramm entspricht also einer `while` - Schleife.