Programmierung mit C

Bedingte Anweisungen und Schleifen

Kontrollstrukturen

R|R|Z|N

- Mit Hilfe von Kontrollstrukturen kann der Programmablauf beeinflusst werden.
- In Abhängigkeit vom Wert einer oder mehrere Variablen wird der Programmablauf gesteuert.
- Es gibt folgende Möglichkeiten:
  - Auswahlanweisungen:
    - Bedingte Anweisungen; if – then - Anweisungen
    - Fallunterscheidungen; switch - Anweisungen
  - Schleifen (Iterationsanweisungen):
    - Kopfgesteuerte Schleifen; while - Schleife
    - Fußgesteuerte Schleifen; do – while - Schleife
    - Zählschleifen; for - Schleifen
- Schleifen können vom Programmierer unter bestimmten Bedingungen unterbrochen werden.

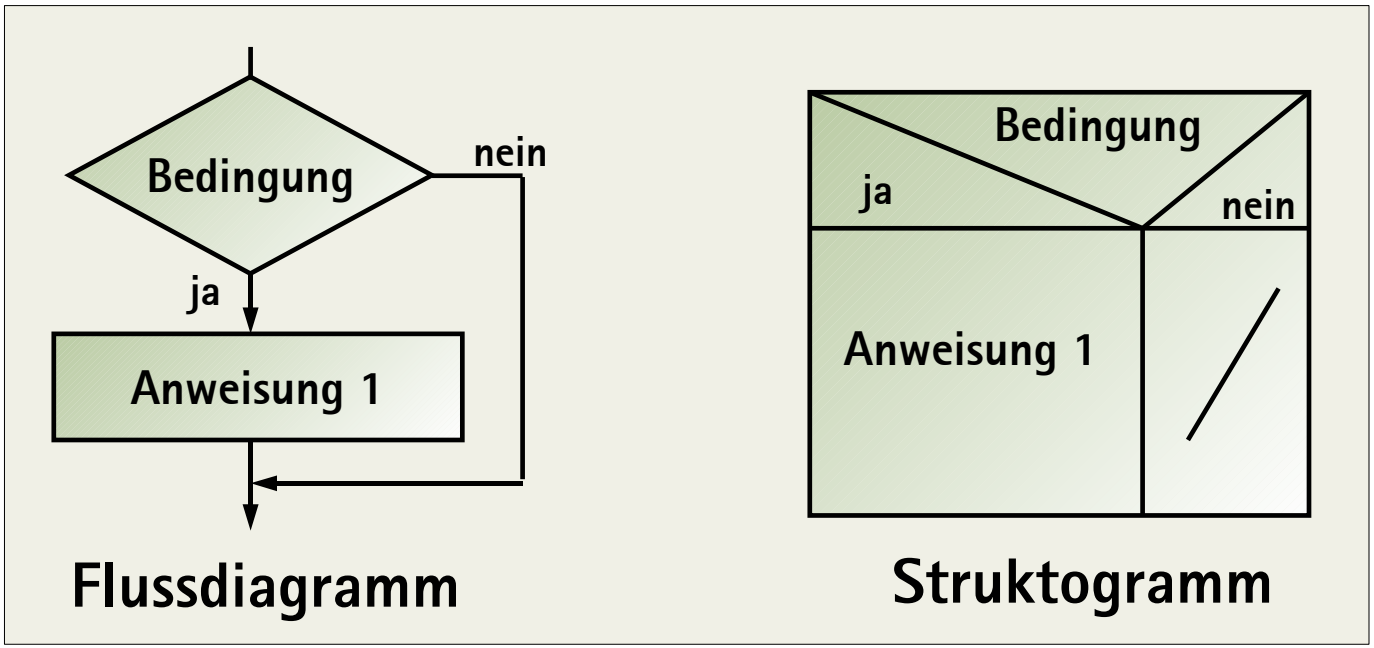
Bedingte Anweisung

R|R|Z|N

`if (Bedingung) { Anweisung }`

- `if (Bedingung)` fragt ab, ob ein Ausdruck wahr ist.
  - Ein Ausdruck, der den Wert wahr oder falsch liefert, wird als Bedingung bezeichnet.
  - Wenn die Bedingung wahr ist, werden die Anweisungen in der geschweiften Klammer ausgeführt.
- Die geschweifte Klammer
  - ... klammert mehrere Anweisungen zum Block.
  - ... kann auch nur eine Anweisungszeile enthalten.
  - ... kann leer sein.
  - ... kann wiederum einen Block von Anweisungen enthalten.
- Wenn Bedingung = wahr (true) Dann ...

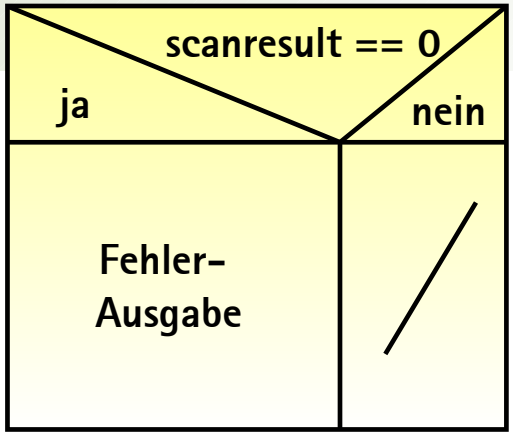
```
if (Ausdruck) { Anweisung }
```



Beispiel

```
int max            = 0;
int scanresult     = 0;

printf("Geben Sie eine Ganzzahl ein ");
scanresult = scanf("%d", &max);

if (scanresult == 0) {
    printf("Fehleingabe, Es wurde keine Zahl eingegeben \n\n");
}
```

Falls der Benutzer nur <ENTER> drückt, wird die Funktion scanf() nie beendet.



Weitere mögliche Fehler

R|R|Z|N

```
int max            = 0;
int scanresult     = 0;

printf("Geben Sie eine Ganzzahl ein ");
scanresult = scanf("%d", &max);

if (scanresult = 0) {
    printf("Fehleingabe, Es wurde keine Zahl eingegeben \n\n");
}
```

- Statt dem Vergleichsoperator „ist gleich“ wird der Zuweisungsoperator genutzt.
- `scanresult` wird der Wert 0 zugewiesen, unabhängig davon welchen Wert `scanresult` hat.
- Die Bedingung trifft immer zu. Das heißt, die printf()-Anweisung wird bei jedem Durchlauf ausgeführt.

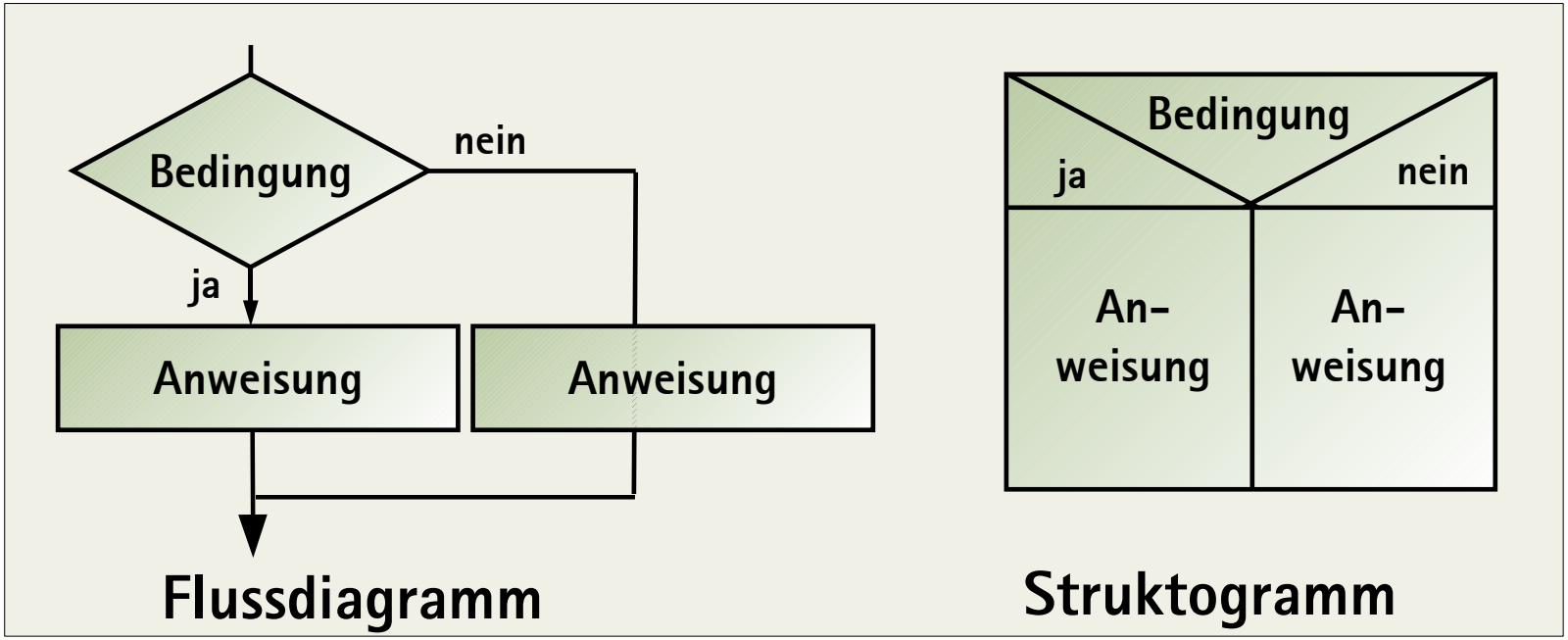
if-else-Anweisung

R|R|Z|N

```
if (Ausdruck) { Anweisung }
else { Anweisung }
```

- Wenn die Bedingung wahr ist, werden die Anweisungen in der geschweiften Klammer ausgeführt, die zu der if-Anweisung gehören. Andernfalls werden die Anweisungen ausgeführt, die zur else-Anweisung gehören.
- Wenn Bedingung = wahr (true) Dann ... Andernfalls ...

```
if (Ausdruck) { Anweisung }
else { Anweisung }
```



Beispiel

R|R|Z|N

```
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    int zahl_1 =0;
    int zahl_2 =0;

    printf("Zwei Zahlen eingeben");
    scanf("%d %d",&zahl_1, &zahl_2);

    if (zahl_1 >= zahl_2)
       printf("Erste Zahl ist groeser gleich zweiter Zahl\n");
    else
       printf("Erste Zahl ist kleiner als zweite Zahl\n");

    system("Pause");
    return 0;
}
```

Beispiel: Keine Eingabe

R|R|Z|N

```
#include <stdio.h>

int main(){
    int max = 0;
    int scanresult = 0;
    char zeichen;

    printf ("Bitte geben Sie eine Ganzzahl ein: ");
    zeichen = getc(stdin);

   if (zeichen == '\n'){
        printf("Keine Eingabe");
    }
    else{
        ungetc(zeichen, stdin);
        scanresult = scanf("%d", &max);
        printf("%d \n", max);
    }
return 0;
}
```

Erläuterung

```
#include <stdio.h>

int main(){
    int max = 0;
    int scanresult = 0;
    char zeichen;

    printf ("Bitte geben Sie eine Ganzzahl ein: ");
    zeichen = getc(stdin);

   if (zeichen == '\n'){
        printf("Keine Eingabe");
   }
   else{
        ungetc(zeichen, stdin);
        scanresult = scanf("%d", &max);
        printf("%d \n", max);
    }
return 0;
}
```

Hier wird ein Zeichen von der Standardeingabe eingelesen.

Falls das Zeichen <ENTER> enthält, wird eine Meldung ausgegeben.

Falls das Zeichen <> <ENTER> war, wird das zuletzt gelesene Zeichen zurück in den Stream geschrieben.

Anschließend wird die Eingabe vollständig mit scanf() eingelesen.

Bedingungsoperator

R|R|Z|N

```
(variable) = (Bedingung) ? if : else
```

- Mit Hilfe des Bedingungsoperators (Fragezeichen) kann die Anweisung if - else ersetzt werden.
- Die Variable wird mit Hilfe der Anweisungen des if- oder else-Zweigs berechnet.
- Falls die Bedingung wahr ist, wird der if-Zweig ausgeführt, andernfalls der else-Zweig.

Beispiel

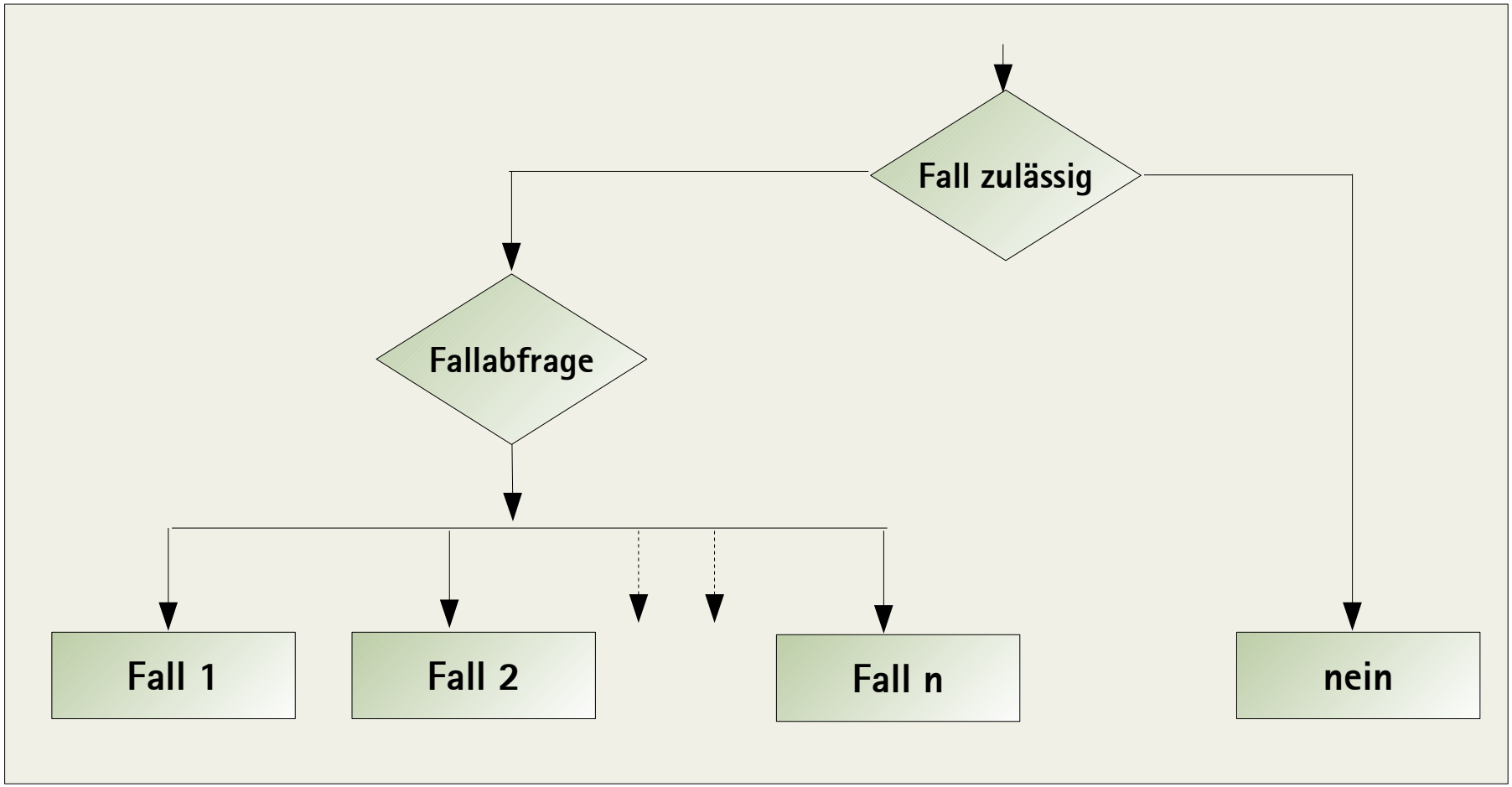
```
/*--- Berechnung mit Bedingungsoperator--*/
    ergebnis = (zahlB != 0) ? zahlA / zahlB : -1;

/*---- Berechnung mit if else Statement--*/
    if (zahlB != 0)
       ergebnis = zahlA / zahlB;
    else
       ergebnis = -1;
```

Anweisungsleiter

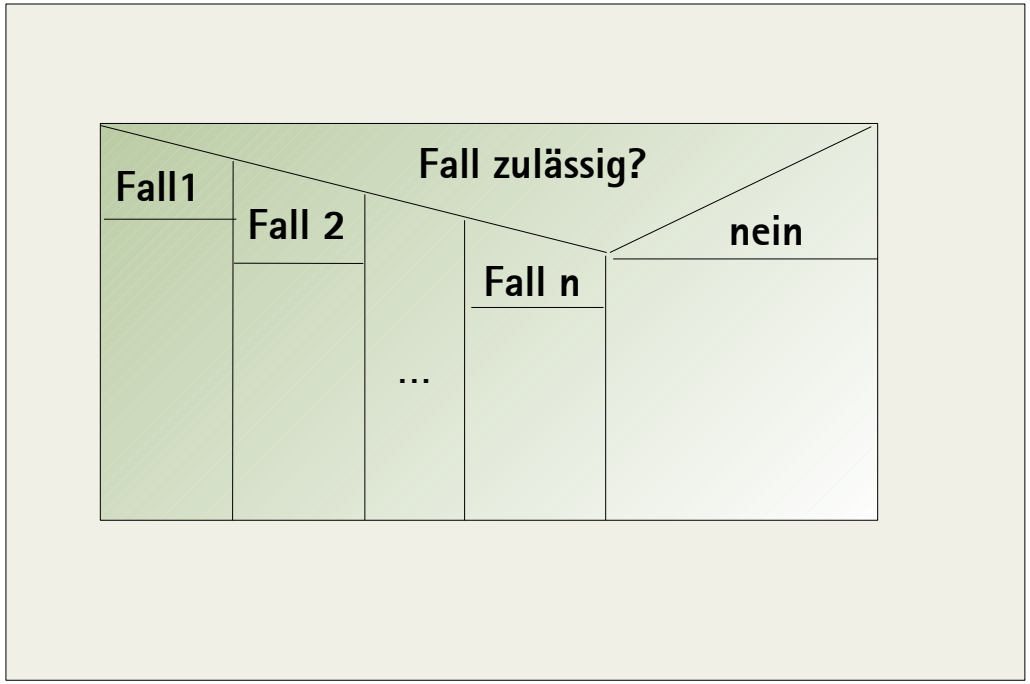

```
if (Ausdruck) { Anweisung }
else if (Ausdruck) { Anweisung }
else { Anweisung }
```

- Der if-Anweisung folgt eine weitere if-Anweisung
- Die Bedingungen werden von oben nach unten ausgewertet.
- Sobald eine Bedingung wahr ist, werden die dazugehörigen Anweisungen ausgeführt und der Rest der Leiter übergangen.
- Wenn keine der Bedingungen wahr ist, wird der else-Zweig ausgeführt.
- Falls Integer- oder Character-Werte auf Gleichheit geprüft werden, sollte eine switch-Anweisung genutzt werden.

Flußdiagramm



Struktogramm



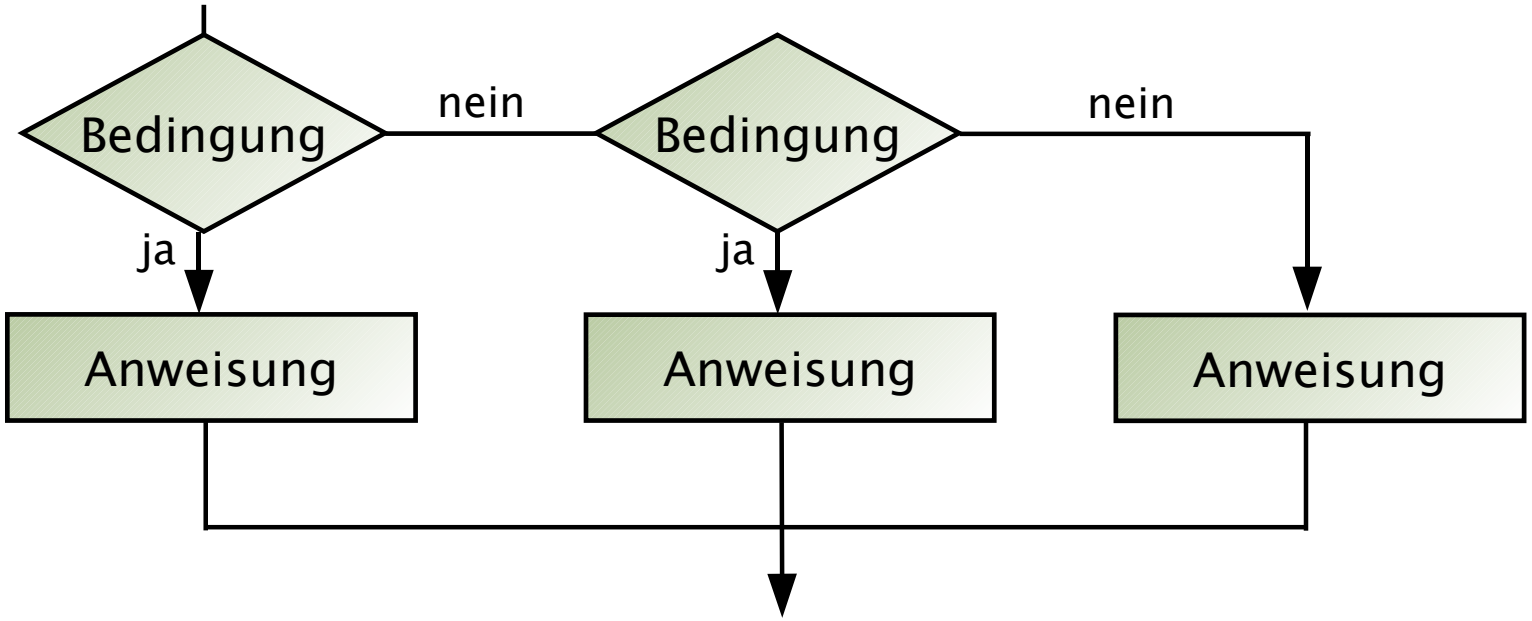
Beispiel

```
int zahlA;
int zahlB;
int check;          /* Anzahl der eingelesenen Zeichen */

printf("Eingabe (Ganzzahl / Ganzzahl):  " );
check = scanf("%d %d", &zahlA, &zahlB);
printf("Check = %d\n", check);

/* Es sind zwei Zeichen gelesen wurden */
if (check == 2) {
   printf("%d / %d = %d\n", zahlA, zahlB, zahlA / zahlB);

/* Das erste Zeichen wurde korrekt gelesen */
} else if (check == 1) {
   printf("Der zweite Wert ist nicht korrekt\n");

} else {
         printf("Keine korrekte Eingabe\n");
}
```

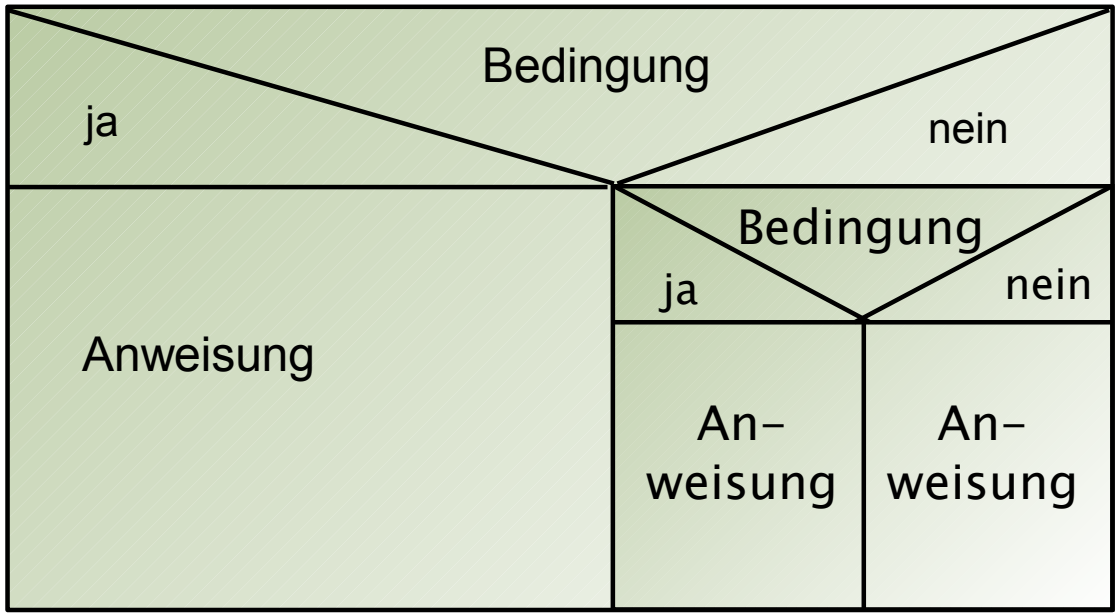
Verschachtelte if-Anweisung

R|R|Z|N

```
int zahlA;
int zahlB;
int result;

result = 0;
printf("Eingabe (Ganzzahl / Ganzzahl):  " );
check = scanf("%d %d", &zahlA, &zahlB);

result = zahlA % zahlB;
if (result == 0) {

    if (zahlA == zahlB) {
            printf("Beide Zahlen sind gleich\");
    } else {
          printf("Zahlen ohne Rest teilbar\n");
    }

} else {
          printf("Zahlen nicht ohne Rest teilbar\n");
}
```


Bedingung
nein
Bedingung
nein
ja
ja
Anweisung
Anweisung
Anweisung

Struktogramm



Logische Verknüpfungen

R | R | Z | N

- Folgende logische Operatoren sind vorhanden:
  - UND &&
  - ODER ||
  - Negation (NICHT) !
- Beispiele für logische Verknüpfungen:

```
if (( a == 5 ) && ( b >= 22 ) {  …  }
if (( a != 0) || ( b  >- 1 ) ) {  …   }
if ( !a ) {   …   }
```

- Versuchen Sie möglichst einfach zu verstehenden Code zu schreiben.

  if (a !=0) {   …   } ist einfacher zu verstehen als

  if (!a) {   ….   }, obwohl die Anweisung identischen Code erzeugt.

Beispiel: Nutzung des Not-Operators

R|R|Z|N

```
int zahl;
printf(„Bitte geben Sie eine Zahl ein:  " );
scanf("%d", &zahl);

if (!(zahl == 0)) {
    printf("Die Zahl ist ungleich als Null");

}

if (zahl != 0) {
    printf("Die Zahl ist ungleich als Null");

}
```

Beispiel: Nutzung des AND-Operators

```
#include <stdio.h>

int main()
{
  int zahl;

  printf("Geben Sie einen Wert zwischen 10 und 20 ein: ");
 scanf("%d", &zahl);

      if ( (zahl >= 10) && (zahl <= 20) ) {
      printf("Ausgabe Messpunkt %d \n", zahl);
      }

      else {
      printf("Falsche Eingabe! \n");
      }

      return 0;
}
```

Beispiel: Nutzung des OR-Operators

R|R|Z|N

```
#include <stdio.h>

int main()
{

   int zahl;
   printf("Geben Sie einen Wert zwischen 10 und 20 ein: ");
   scanf("%d", &zahl);

   if ( (zahl < 10) || (zahl > 20) )
   {
      printf("Falsche Eingabe! \n");
   }

   return 0;
}
```

- Beispiel: `(var1 != var2) && (var2 > 10)`
  - Zuerst wird die linke Bedingung `(var1 != var2)` ausgewertet.
  - Anschließend wird die rechte Bedingung `(var2 > 10)` ausgewertet.
  - Beide Ergebnisse müssen in diesen Fall wahr sein, andernfalls wird der else-Zweig augeführt.
- Sie können die Operatoren beliebig oft in beliebiger Mischung in einer Bedingung nutzen.
- Um die Lesbarkeit zu erhöhen, sollten die verschiedenen Elemente der Bedingung mit runden Klammern zusammengefasst werden.
- Falls verschiedene Operatoren gemischt werden, muss die Bindung der Operatoren beachtet werden.

Unäre Operatoren
Mathematische Operatoren

Shiftoperatoren
Vergleichsoperatoren

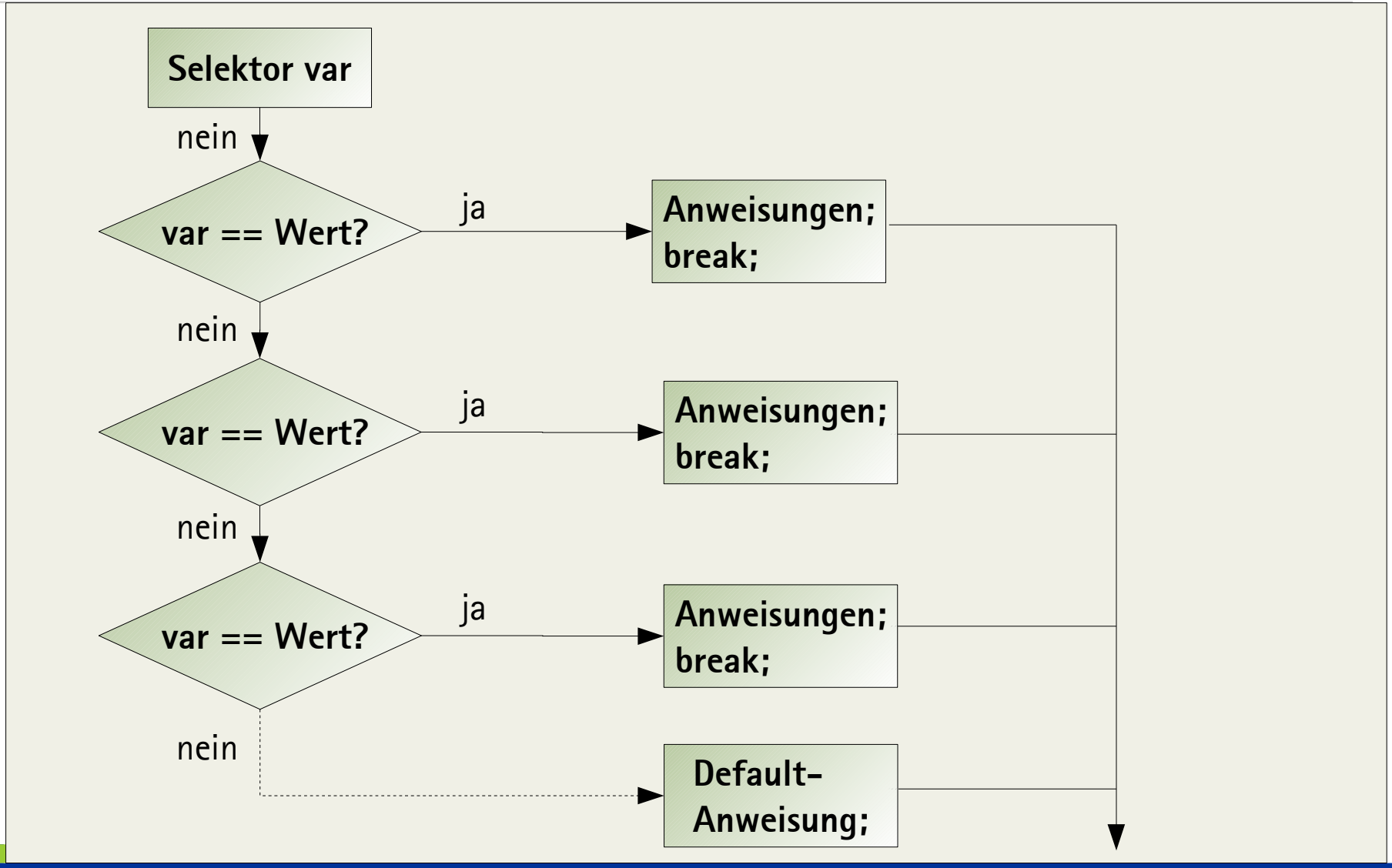
- Jede Ebene einer verschachtelten if-Anweisung wird mit Hilfe des Tabulators eingerückt.
- Klammern Sie zusammengehörige Blöcke, um die Lesbarkeit zu erhöhen.
- Vermeiden Sie unterschiedliche Datentypen auf beiden Seiten eines Vergleichsoperators.
- Verzichten Sie auf if-else-Leitern, wenn Sie switch-Anweisungen nutzen können.
- Kurze if-Anweisungen sollten mit Hilfe des Bedingungsoperators erstellt werden.

Fallunterscheidung mit switch

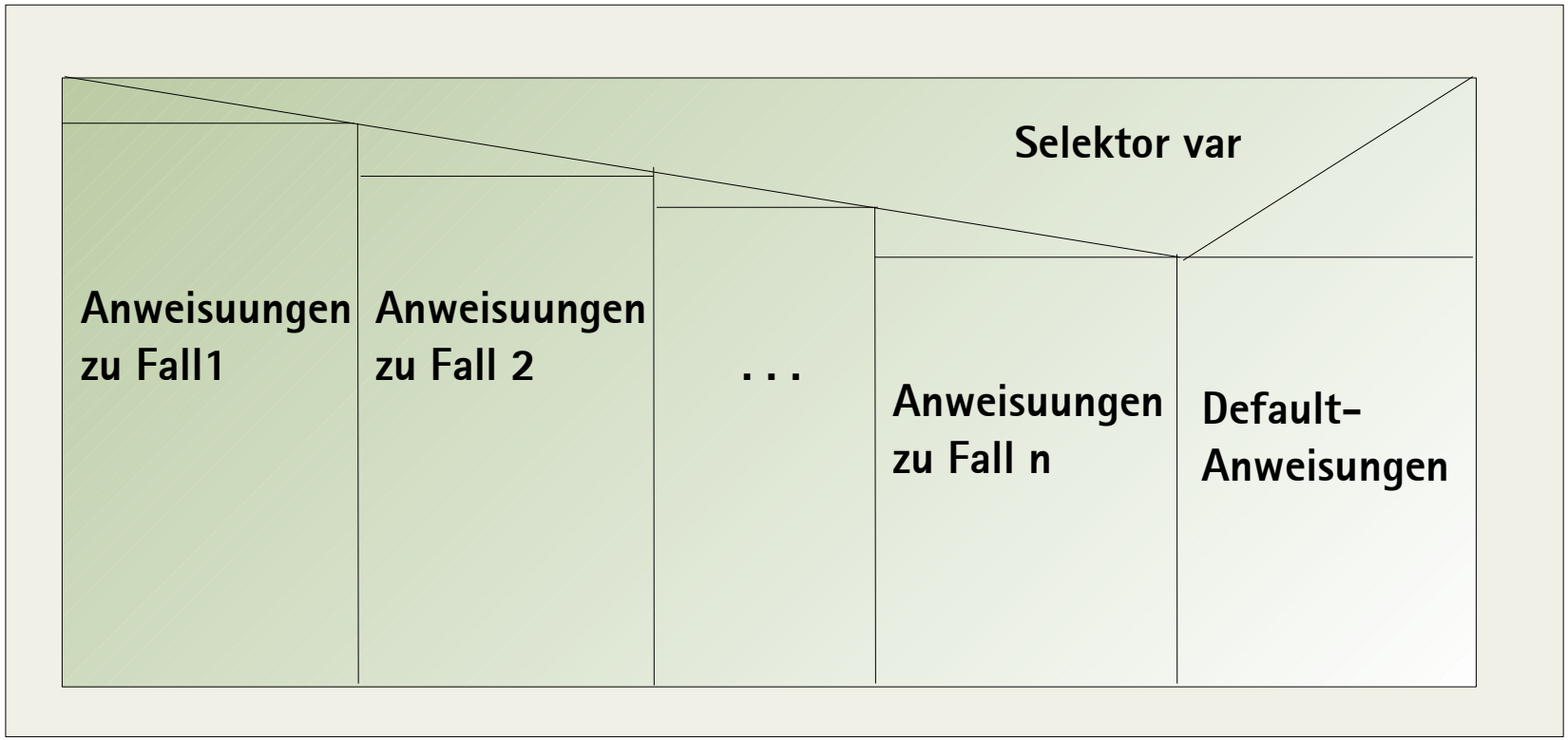
- Der Wert eines Ausdrucks wird mit vorgegebenen Werten verglichen.
- Der Wert eines Ausdrucks wird mit einer Liste von Konstanten vom Datentyp `char` oder `int` verglichen.
- Jeder Vergleich wird mit `case` (Fall) eingeleitet.
- Es ist nur ein Vergleich auf Gleichheit möglich.
- Falls der Wert des Ausdrucks mit einer Konstanten aus der Liste übereinstimmt, werden alle nachfolgenden Anweisungen bis zum Schlüsselwort `break` verarbeitet.
- Die Angabe von `default` ist optional. Hier kann der Standardfall beschrieben werden.

```
switch (Ausdruck) {
   case Konstante1:
        Anweisung;
        break;
   ...
   case Konstanten:
        Anweisung;
        break;
   default:
        Anweisung;
}
```

Flußdiagramm



Struktogramm



Beispiel

```
switch (check) {
   case 2:
        if (zahlB == 0) {
                printf("Fehler: Division durch Null\n");

        } else {
                printf("Ergebnis = %d\n", zahlA / zahlB);

        }
        break;

   case 1:
        printf("Der zweite Wert ist nicht korrekt\n");
        break;

   default:
        printf("Keine korrekte Eingabe\n");
}
```

Break

- ... ist eine Sprunganweisung.
- ... kann in Schleifen und switch-Anweisungen vorkommen.
- ... beendet einen Anweisungsblock.
- Wenn das Programm auf ein `break` trifft, springt das Programm in die Codezeile, die der switch-Anweisung folgt.
- ... ist in switch-Anweisungen optional. Falls aber am Ende einer case-Anweisung kein `break` vorkommt, werden alle nachfolgenden case-Anweisungen auch durchlaufen.

- Eine switch-Anweisung kann im Gegensatz zu einer if-Anweisung nur auf Gleichheit testen.
- Keine case-Konstante kann in ein und derselben switch-Anweisung den gleichen Wert besitzen.
- Eine switch-Anweisung kann in ein andere switch-Anweisung eingeschlossen werden.
- Zeichenkonstanten in einer switch-Anweisung werden in Integer umgewandelt.
- switch-Anweisungen werden häufig für Menüs oder Tastaturabfragen genutzt.

Beispiel für ein Menü

R|R|Z|N

```
#include <stdio.h>

int main(void)
{
char eingabe;

...

switch(eingabe)
{
    case 'E':  printf("Neuen Eintrag einfügen");
    case 'L':  printf("Eintrag löschen");
    case 'S':  printf("Telefonnummer suchen");
    case 'A':  printf("Alle Einträge anzeigen");
    case 'B':  printf("Programm-Ende");
    default:   printf("Falsche Eingabe");
}

...
return 0;
}
```

Schleifen (Iterationsanweisungen)

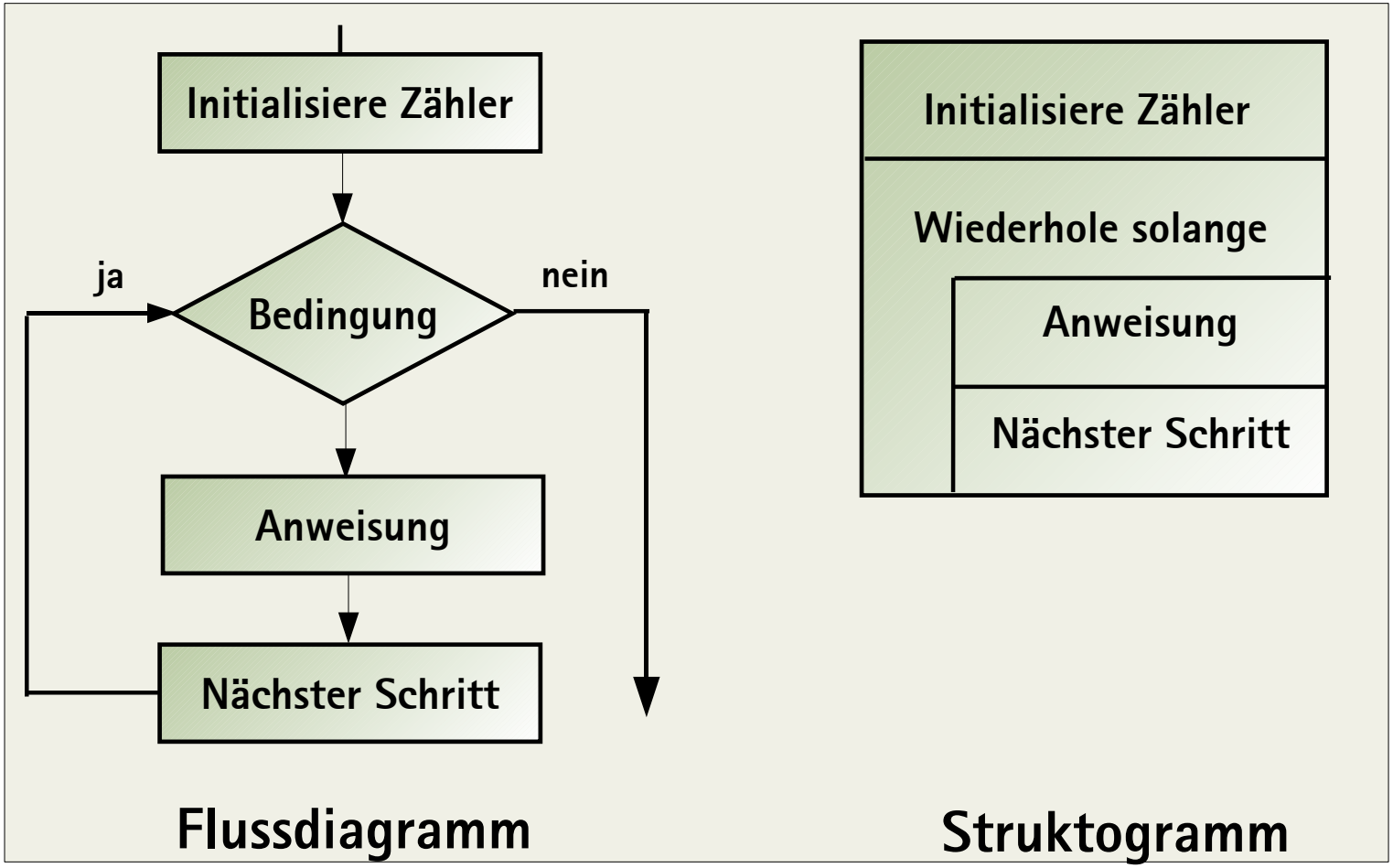
- ... sind Wiederholungen bestimmter Anweisungen.
- Die Anzahl der Durchläufe wird durch eine Bedingung bestimmt. Falls diese Bedingung nie wahr wird, wird die Schleife als Endlosschleife bezeichnet.
- ... können vorzeitig abgebrochen werden.
- … bestehen immer aus einem Schleifenkopf und einem Schleifenrumpf.
- Es gibt in der C-Programmierung
  - Kopfgesteuerte Schleifen (while-Schleifen)
  - Fußgesteuerte Schleifen (do - while-Schleifen)
  - Zählschleifen (for-Schleifen)

```
for (Initialisierung; Bedingung; Reinitialisierung)
    { Anweisung };
```

- Die for-Schleife ist eine Zählschleife.
- `Initialisierung` ist eine Anweisung, die eine Variable einen Anfangswert zuweist. Die Variable wird initialisiert und zählt die Schleifendurchläufe.
- `Bedingung` legt fest, wann die Schleife beendet wird. Solange die Bedingung wahr ist, werden die Anweisungen im Schleifenrumpf abgearbeitet. Andernfalls wird die Schleife abgebrochen.
- `Reinitialisierung` legt fest, wie die Variable nach jedem Schleifendurchgang verändert wird. Dem Zähler wird ein neuer Wert für den nächsten Schleifendurchlauf zugewiesen.
- Initialisierung, Bedingung und Reinitialisierung werden durch ein Semikolon voneinander getrennt, weil sie als Anweisung behandelt werden.

- Zuerst wird einmalig die Schleifenvariable (Zähler) initialisiert. Die Schleifenvariable zählt die Schleifendurchgänge.
- Anschließend wird die Bedingung überprüft. Wenn diese wahr ist, werden die Anweisungen im Schleifenrumpf ausgeführt.
- Dann wird die Schleifenvariable entsprechend der Angabe im Feld Reinitialisierung auf einen neuen Wert gesetzt und die Bedingung erneut überprüft.
- Falls die Bedingung falsch ist, wird die Schleife abgebrochen.

Flussdiagramm und Struktogramm



Beispiele

R|R|Z|N

```
int count;
float zaehler;

for (count = 0; count < 10;  count++) {
   printf("Schleifendurchlauf: %d\n", count);
}

for (count = 10; count > 0;  count -= 1) {
   printf("Schleifendurchlauf: %d\n", count);
}

for (zaehler = 5; zaehler > 0;  zaehler -= 0.25) {
   printf("Schleifendurchlauf: %.2f\n", zaehler);
}
```

```
int count;
float zaehler;

for (count = 0; count < 10;  count++) {
   printf("Schleifendurchlauf: %d\n", count);
}

for (count = 10; count > 0;  count
   printf("Schleifendurchlauf: %d\n"
}

for (zaehler = 5; zaehler > 0;
   printf("Schleifendurchlauf: %
}
```

In der ersten for-Schleife werden alle Zahlen von 0 bis 9 ausgegeben. Die Zählvariable wird mit 0 initialisiert und nach jedem Schleifendurchlauf um eins erhöht. Die Schleife wird solange durchlaufen, wie die Zählvariable kleiner als 10 ist.

```
int count;
float zaehler;

for (count = 0; count < 10;  count++) {
   printf("Schleifendurchlauf: %d\n", count);
}

for (count = 10; count > 0;  count -= 1) {
   printf("Schleifendurchlauf: %d\n", count);
}

for (zaehler = 5; zaehler                zaehler -= 0.25) {
   printf("Schleifendurchlau                 \n", zaehler);
}
```

In der zweiten for-schleife werden alle Zahlen von 10 bis 1 ausgegeben. Die Zählvariable wird bei jedem Schleifendurchlauf um eins herunter gezählt.

Beispiele

```
int count;
float zaehle

for (count =
   printf("S
}

for (count =
   printf("Schleifendu                d\n", count);
}

for (zaehler = 5; zaehler > 0;  zaehler -= 0.25) {
   printf("Schleifendurchlauf: %.2f\n", zaehler);
}
```

In der letzten for-Schleife wird eine Zählvariable als float definiert und mit fünf initialisiert. Nach jedem Schleifendurchlauf wird die Zählvariable um 0.25 heruntergezählt.

Die Veränderung des Zählers nach jedem Schleifendurchlauf muss zum Datentyp der Zählvariablen passen.

- Häufig werden die Anweisungen in der runden Klammer durch Kommata getrennt.

  Das Trennzeichen zwischen den Anweisungen in den runden Klammern wird vergessen.

  - falsch: `for ( count = 0  count < 10  count++ ) { }` ☒
  - falsch : `for ( count = 0, count < 10,  count++ ) { }` ☒
  - richtig: `for ( count = 0; count < 10;  count++) { }` ☑

- Am Ende des Schleifenkopfes wird ein Semikolon gesetzt.

  Die Anweisungen im Schleifenrumpf werden nicht ausgeführt.

  Die Schleife ist durch das Semikolon beendet.

  - falsch: `for ( count = 0; count < 10; count++ ); { }` ☒
  - richtig: `for ( count = 0; count < 10; count++ ) { }` ☑

- Die Abbruchbedingung ist falsch gewählt.

  Man hat eine Endlosschleife programmiert, weil die Abbruchbedingung immer erfüllt wird.

  - falsch: `for ( count = 2; count == 5; count += 2 ) { }` ☒
  - richtig: `for ( count = 2; count >=5; count += 2 ) { }` ☑
- Der Datentyp des Zähler passt nicht zu der Reinitialisierung der Variablen.
  - Deklaration einer Zählvariablen: `int count;`
  - falsch: `for (count = 2; count >= 5; count += 0.5) { }` ☒
  - richtig: `for (count = 2; count >= 5; count += 1) { }` ☑

Geschachtelte Schleifen

R|R|Z|N

```
int main(void)
{
    int count;
    int mal;

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

- Jede for-Schleife besitzt ihre eigene Schleifenvariable und eine Bedingung für den Abbruch.
- Arbeitsablauf:
  - Der erste Schliefendurchlauf der äußeren Schleife wird gestartet.
  - Die innere Schleife wird solange durchlaufen, bis die Bedingung falsch ist oder mit Hilfe von `break` abgebrochen wird.
  - Anschließend wird die Schleifenvariable der äußeren Schleife entsprechend der Reinitialisierung verändert.
  - Der nächste Schleifendurchlauf der äußeren Schleife beginnt.

- Die Schleifenvariable der inneren Schleife sollte nicht von der Schleifenvariablen der äußeren Schleife abhängig sein.
- Die Schleifenvariable der äußeren Schleife darf nicht durch Anweisungen in der inneren Schleife verändert werden. Eine Endlosschleife ist möglich.
- Falls die innere Schleife mit Hilfe von `break` abgebrochen wird, wird die äußere Schleife nicht automatisch abgebrochen. Es wird der aktuelle Schleifendurchlauf der inneren Schleife abgebrochen und der nächste Schleifendurchlauf der äußeren Schleife gestartet.

Zeichen für die Initialisierung nutzen

Als Zählvariable kann auch ein Character genutzt werden.

Der Character wird intern automatisch in einen Integer-Wert umgerechnet.

```
int main(void)
{
char zeichenGross;

for (zeichenGross = 'A';  zeichenGross <=  'C';  zeichenGross ++)
    {
    printf("Schleifendurchlauf: %.2c\n", zeichenGross );
    }
return 0;
}
```

Berechnungen in Schleifenköpfen nutzen

Berechnungen können in der Abbruchbedingung vorkommen.

Wenn der Zähler zum Quadrat größer als 30 ist, wird die Schleife abgebrochen. Sonst werden die Anweisungen im Schleifenrumpf ausgeführt. Der Wert des Zählers wird durch den Ausdruck im Schleifenkopf nicht verändert.

```
int main(void)
{
    int count;
    int ergebnis;

    for (count = 1; count * count <= 30;  count++) {
       ergebnis = count * count;
       printf("Ergebnis: %d\n", ergebnis);
    }
}
```

Berechnungen in Schleifenköpfen

Berechnungen in Schleifenköpfen können auch in der Reinitialisierung vorkommen.

Der Wert der Berechnung kann einer Variable zugewiesen werden, die nicht als Zählvariable genutzt wird.

```
int main(void)
{
    int count;
    int ergebnis;

    ergebnis = 50;
    for (count = 0;  ergebnis <= 75;
       ergebnis = ++count * 5) + 50) {
       printf("Ergebnis %d: %d\n", count, ergebnis);
    }
}
```

Reinitialisierung ersetzen

R|R|Z|N

```
int main(void)
{
int count;

for (count = 20;  count > 0; ) {
   printf("Ergebnis %d\n", count);
   count = count - 2;
}

count  = 20 ;
for ( ; count > 0; ) {
   printf("Ergebnis %d\n", count);
   count = count - 2;
}

}
```

Wenn innerhalb des Schleifenkopfs die Initialisierung, das Abbruchkriterium oder die Reinitialisierung weggelassen wird, muss das Semikolon für die Anweisung trotzdem gesetzt werden.

Die Reinitialisierung kann auch innerhalb des Schleifenrumpfes vorgenommen werden. Falls nicht, wird eine Endlosschleife erzeugt, weil das Abbruchkritierium nie erreicht wird.

Initialisierung ersetzen

R|R|Z|N

```
int main(void)
{
int count;

for (count = 20;  count > 0; ) {
   printf("Ergebnis %d\n", count);
   count = count - 2;
}

count  = 20 ;
for ( ; count > 0; ) {
   printf("Ergebnis %d\n", count);
   count = count - 2;
}

}
```

Die Initialisierung der Schleifenvariable kann außerhalb der Schleife vorgenommen werden. Falls die Initialisierung vergessen wird, besitzt die Schleifenvariable einen unbestimmten Wert.

Wenn die Abbruchbedingung entfernt wird, erzeugt man eine Endlosschleife. Die Schleife bricht nie ab.

Funktionen in Schleifenköpfen

R|R|Z|N

```
int main(void)
{
int count;

for ( printf("Bitte geben Sie eine Zahl ein: "); count != 5; ) {
   scanf("%d", &count);
}

}
```

Funktionen können in der Initialisierung oder Reinitialisierung genutzt werden.

Die Reinitialisierungsanweisung in der Kopfzeile der Schleife ist leer. Statt dessen wird die variable [count] im Schleifenkörper reinitialisiert.

Achtung die printf-Funktion wird nur einmal ausgeführt!

Mehrere Werte initialisieren

R|R|Z|N

Sie können auch mehrere Werte innerhalb eines Schleifenkopfes initialisieren. Die Initialisierung der verschiedenen Variablen wird durch ein Komma getrennt.

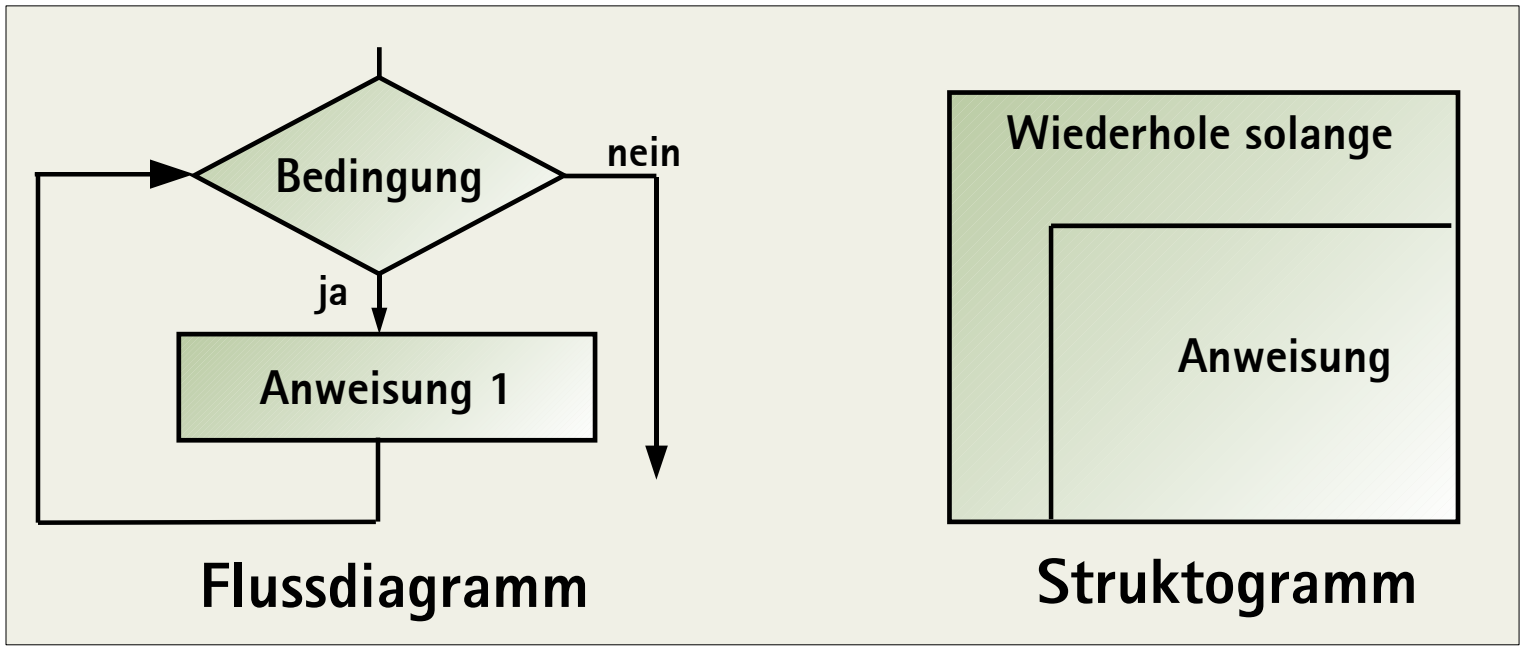
Es ist auch möglich, Abbruchkriterien und Reinitialisierung für verschiedene Variablen im Schleifenkopf anzugeben.

Die einzelnen Abschnitte müssen innerhalb der Anweisung durch Kommata getrennt werden.

```
int main(void)
{
int count;
int merkCount;

for (count = 1, merkCount = 2; count <= 10; count++)
{
printf("%d * %d = %d\n", count, merkCount, count * merkCount);
}

}
```

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

- Solange die Bedingung erfüllt ist, führe die Anweisungen aus.
- ... ist eine kopfgesteuerte Schleife.
- Die Bedingung steht im Schleifenkopf.
- Die Anweisungen stehen im Schleifenrumpf. Innerhalb der Anweisung muss die Schleifenvariable so verändert werden, dass die Bedingung zum Ende der Schleife falsch wird. Andernfalls ist eine Endlosschleife programmiert.


Bedingung
nein
ja
Anweisung 1
Flussdiagramm
Wiederhole solange
Anweisung
Struktogramm

Beispiel

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

/* Solange Bedingung erfüllt ist, führe aus... */
while (count <= 10) {
   ergebnis = ergebnis + count;
   printf("Schleifendurchlauf %d: %d\n", count, ergebnis);
   count++;     /* Reinitialisierung der Schleifenvariable */
   }
   return 0;
}
```

Die Schleifenvariable muss initialisiert werden. Das heißt die Schleifenvariable bekommt einen bestimmten Wert.

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

/* Solange Bedingung erfüllt ist, führe aus... */
while (count <= 10) {
   ergebnis = ergebnis + count;
   printf("Schleifendurchlauf %d: %d\n", count, ergebnis);
   count++;     /* Reinitialisierung der Schleifenvariable */
   }
   return 0;
}
```

Beispiel


```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

/* Solange Bedingung erfüllt ist, führe aus... */
while (count <= 10) {
   ergebnis = ergebnis + count;
   printf("Schleifendurchlauf %d: %d\n", count, ergebnis);
   count++;     /* Re    ialisierung der Schleifenvariable */
   }
   return 0;
}
```

Die Schleifenvariable wird daraufhin geprüft, ob die angegebene Bedingung erfüllt ist.

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Ini
ergebnis = 0;

/* Solange Bedingung erfüllt      führe aus... */
while (count <= 10) {
   ergebnis = ergebnis + count;
   printf("Schleifendurchlauf %d: %d\n", count, ergebnis);
   count++;     /* Reinitialisierung der Schleifenvariable */
   }
   return 0;
}
```

Wenn die Bedingung erfüllt ist, werden die Anweisungen im Schleifenrumpf durchlaufen. Der Schleifenvariablen wird im Rumpf ein neuer Wert zugewiesen.

Anschließend wird die Bedingung erneut geprüft.

Endlosschleifen

Wenn die Bedingung immer erfüllt ist, läuft eine while-Schleife endlos.
Wenn Sie einen Wert eins abfragen, trifft die Bedingung immer zu.
Eins oder ein Wert ungleich null steht für wahr (true).
Ein Wert gleich null steht für falsch (false).

```
#include <stdio.h>
int main(void)
{
long zaehler =0;

while ( 1 ) {  /* No way to escape */
   printf("Diese Meldung wird jetzt endlos ausgegeben. %ld\n
   ",zaehler);
     zaehler++;
}

printf("Diese Zeile wird niemals erreicht werden");
return 0;
}
```

Mögliche Fehler

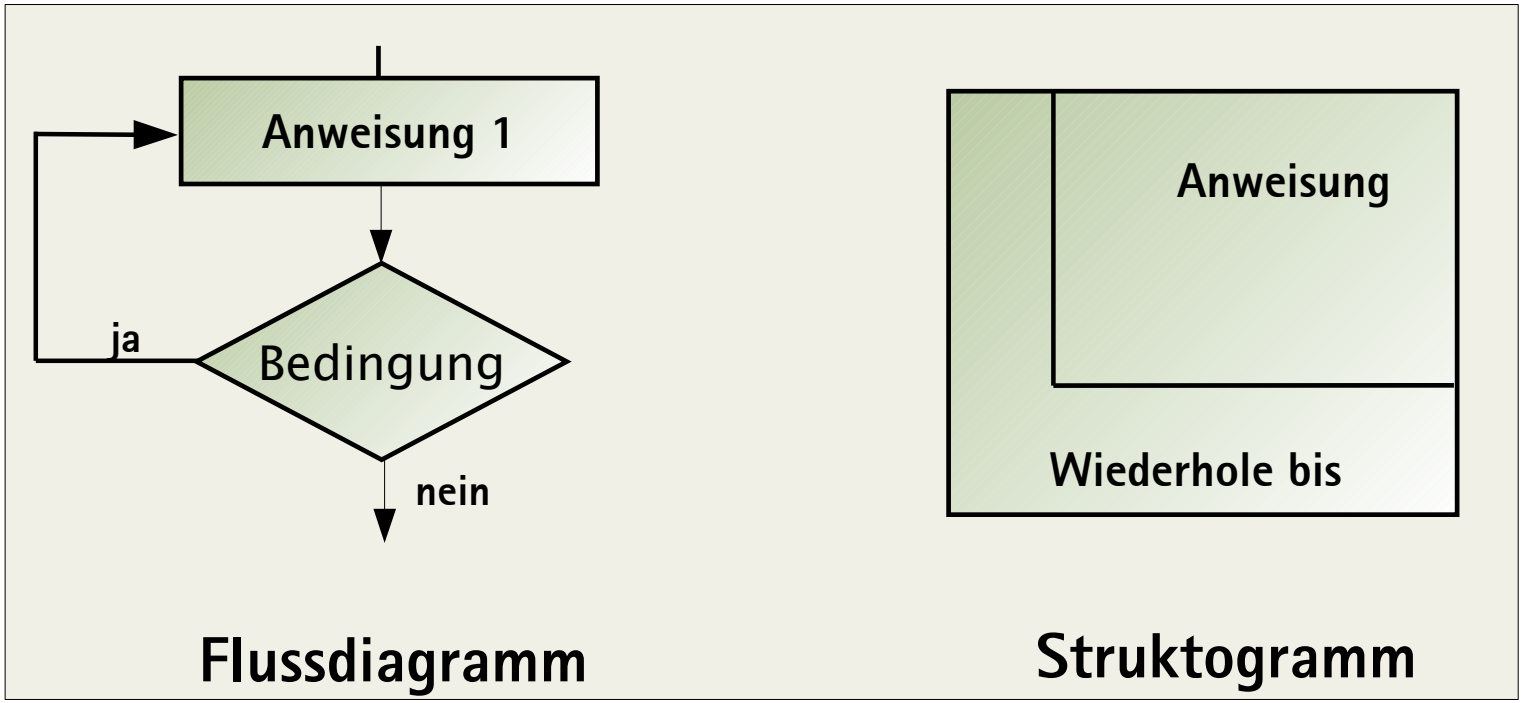
- Wenn der Schleifenkopf mit einem Semikolon abgeschlossen wird, wird eine Endlosschleife erzeugt. Der Schleifenvariablen wird kein neuer Wert zugewiesen und die Bedingung ist immer erfüllt.
  - falsch: `while (count > 10); { }` ☒
  - richtig: `while (count > 10) { }` ☑
- Vermeiden Sie, wenn möglich Überprüfungen auf Gleichheit. Die Berechnung innerhalb des Schleifenrumpfs muss exakt den angegebenen Wert erreichen, damit die Schleife abgebrochen wird.
  - ungünstig: `while (count != 10) { }` ☒
  - besser: `while (count <= 10) { }` ☑

- Überlegen Sie sich genau den Einsatz von logischen Operatoren.
- Beispiel: `while ((count++ < 5) || (zaehler++ < 5)) { }`
  - Die erste Bedingung ist solange wahr, wie `count` kleiner als 5 ist.
  - Die zweite Bedingung wird erst abgefragt (und `zaehler` inkrementiert), wenn die erste Bedingung falsch wird.
  - Die zweite Bedingung wird fünfmal durchlaufen und `count` zusätzlich weiter mit hochgezählt.
  - Die Schleife wird beendet, wenn `count` gleich 10 ist und `zaehler` gleich 5.

```
count= 1   zaehler= 0
count= 2   zaehler= 0
count= 3   zaehler= 0
count= 4   zaehler= 0
count= 5   zaehler= 0
count= 6   zaehler= 1
count= 7   zaehler= 2
count= 8   zaehler= 3
count= 9   zaehler= 4
count=10   zaehler= 5
```

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

- Bis die Bedingung erfüllt ist, führe die Anweisungen aus.
- Arbeitsablauf:
  - Beim ersten Durchlauf werden zuerst die Anweisungen im Schleifenrumpf durchlaufen.
  - Anschließend wird die Bedingung überprüft. Falls die Bedingung wahr ist, werden die Anweisungen nochmals durchlaufen. Andernfalls wird die Schleife nach dem ersten Durchlauf abgebrochen.
- Die do-while-Schleife wird mit einem Semikolon beendet. Die Bedingung wird als Anweisung interpretiert, die mit einem Semikolon abgeschlossen wird.
- ... sind eine fußgesteuerte Schleife.


Anweisung 1
ja
Bedingung
nein
Flussdiagramm
Anweisung
Wiederhole bis
Struktogramm

Beispiel

R|R|Z|N

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

do {
   ergebnis = ergebnis + count;
   printf("Ergebnisf %d: %d\n", count, ergebnis);
   count++; /* Reinitialisierung der Schleifenvariablen */
}
while (count <= 10); /* Ist die Bedingung erfüllt? */

return 0;
}
```

Beispiel

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

do {
   ergebnis = ergebnis + count;
   printf("Ergebnisf %d: %d\n", count, ergebnis);
   count++; /* Reinitialisierung der Schleifenvariablen */
}
while (count <= 10); /* Ist die Bedingung erfüllt? */

return 0;
}
```

Die Schleifenvariable muss initialisiert werden.

Das heißt die Schleifenvariable bekommt einen bestimmten Wert.

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

do {
   ergebnis = ergebnis + count;
   printf("Ergebnisf %d: %d\n", count, ergebnis);
   count++; /* Reinitialisierung der Schleifenvariablen */
}
while (count <= 10); /* Ist die Bedingung erfüll    */

return 0;
}
```

Die Anweisungen innerhalb des Schleifenrumpfs werden ausgeführt. Die Schleifenvariable bekommt im Rumpf einen neuen Wert zugewiesen.

Beispiel

```
int main(void)
{
int count;
int ergebnis;
count = 1;      /* Initialisierung der Schleifenvariablen */
ergebnis = 0;

do {
   ergebnis = ergebnis + count;
   printf("Ergebnisf %d: %d\n", count, ergebnis);
   count++; /* Reinitialisierung der Schleifenvariablen */
}
while (count <= 10); /* Ist die Bedingung erfüllt? */

return 0;
}
```

Wenn die Bedingung erfüllt ist, wird der Anweisungsblock erneut ausgeführt. Andernfalls wird die Anweisung nach der Schleife ausgeführt.

Beispiel: Mehrere Werte einlesen

R|R|Z|N

```
#include <stdio.h>
int main(void)
{
    float messwert;
    char antwort;
    antwort = 'y';
    messwert = 0;

    do{
        fflush(stdin);
        printf("Bitte geben Sie ein Wert ein: ");
        scanf("%f", &messwert);
        fflush(stdin);
        printf("Möchten Sie weitere Werte eingeben (y / n): ");
        scanf("%c", &antwort);
    }while(antwort == 'y');
    return 0;
}
```

Mit Hilfe von fflush() wird der Zeilenpuffer unter Windows geleert. Nicht gelesene Zeichen werden aus dem Puffer entfernt.

- `continue` beendet den aktuellen Schleifenlauf der Schleife und startet automatisch sofort einen neuen Schleifendurchlauf. Es wird ein Schleifendurchlauf übersprungen.
  - Bei while- und do-while-Schleifen wird nach Abbruch des aktuellen Schleifendurchlaufs die Bedingung überprüft.
  - Bei for-Schleifen wird die Schleifenvariable neu initialisiert und anschließend ein neuer Schleifendurchlauf begonnen.
- Die break-Anweisung beendet eine for-, while-, oder do-while-Schleife oder eine Fallunterscheidung vorzeitig. Die Schleife wird sofort verlassen.
- In verschachtelten Schleifen haben die Anweisungen `break` oder `continue` nur auf die zum Block gehörende Schleifen Auswirkungen.

Beenden

- `exit()` beendet das aktuelle Programm und übergibt die Kontrolle an das Betriebssystem.
  - Vollständige Deklaration: `void exit (int status);`
  - `exit(0)` wird zurückgegeben, wenn das Programm ohne Fehler beendet werden konnte.
  - `exit(1)` zeigt zum Beispiel einen Fehler an.
  - Die Headerdatei <*stdlib.h*> muss eingebunden werden.
- `return()` beendet eine Funktion.
  - Es wird an den Aufruf der Funktion zurückgesprungen.
  - Falls `return()` in der Hauptfunktion `main()` aufgerufen wird, wird die Kontrolle an das Betriebssystem abgegeben.
  - Vollständige Deklaration: `return wert;`

    `wert` ist abhängig vom Rückgabewert der Funktion.

Beispiel

R|R|Z|N

```
#include <stdio.h>
int main(void)
{
int zahl;
int ergebnis = 0;

while ( 1 ) {
   printf("Bitte geben Sie einen Wert ein: ");

   scanf("%d", &zahl);
   if (zahl == 0) {
        break;
   } else {
       ergebnis += zahl;
   }
}
printf("\nDie Summe betraegt: %d\n", ergebnis);

return 0;
}
```

Abbruch der Schleife