II. 3 Kontrollstrukturen

II. 3. 1 if-Anweisung

Die einfachste Kontrollstruktur ist die `if`-Anweisung. Die Syntax lautet:

```
if (bedingung)
{
}
```

Ist die Bedingung innerhalb der runden Klammern wahr, d.h. `true`, dann wird der Anweisungsblock innerhalb der geschweiften Klammern ausgeführt. Ist die Bedingung `false`, so wird der Anweisungsblock übersprungen und das Programm führt die Anweisungen hinter der geschweiften Klammer aus.

Beispiel:

```
if (augenzahl == 6)
{
     cout<<"Du hast eine sechs gewürfelt, prima!" << endl;
}
augenzahl = wuerfeln();
```

Übung:
Entwickeln Sie ein Struktogramm nach Nassi-Shneidermann für ein Programm, das Sie eine Zahl über die Tastatur eingeben läßt, und das je nach Eingabezahl ausgibt, ob die Zahl kleiner 100, gleich 100 oder größer 100 ist.
Programmieren Sie dann den dazugehörigen Code.

if - Anweisung

Besteht der Anweisungsblock hinter einer Kontrollstruktur nur aus einer Anweisung, dann können die geschweiften Klammern weggelassen werden.

Ich empfehle Ihnen aber, auf diese Möglichkeiten zu verzichten, da hier leicht Fehlerquellen entstehen können. Beachten Sie auch, daß eine Funktion mit Rückgabewert auch wirklich ein `return` durchläuft.

Was tut folgende Funktion ?

```
int max (int zahl1, int zahl2)
{
     if (zahl1>=zahl2)
          {
               return(zahl1);
          }
     if (zahl1<zahl2)
          {
               return(zahl2);
          }
     return(-99);
}
```

Was tut folgende Funktion ?

```
int max (int zahl1, int zahl 2)
{
     if (zahl1 >= zahl2)
          {
          return(zahl1);
          }
     return (zahl2);
```

if ... else - Anweisung

Die `if` – Anweisung kann zu einer `if ... else` – Anweisung erweitert werden.

Beispiel:

```
if (a>=b)
     {
          cout<<"a ist größer gleich b"<<endl;
     }
else
     {
          cout<<"a ist kleiner als b"<<endl;
     }
```

Ist die `if`-Bedingung true, wird der darauf folgende Anweisungsblock ausgeführt. Ist die Bedingung false, wird der `else` (engl. sonst) – Anweisungsblock ausgeführt. Es wird auf jeden Fall einer der beiden Anweisungsblöcke ausgeführt, aber nie beide.

`if`–Anweisungen können auch verschachtelt werden.

Wie sieht das Struktogramm folgendes Programms aus ? Was tut das Programm ?

```
if(a>=b)
     {
          if (a>b)
               {
                    cout<<"a ist größer b"<<endl;
               }
          else
               {
                    cout<<"a ist gleich b"<<endl;
               }
     }
else
     {
          cout<<"a ist kleiner als b"<<endl;
     }
```

Logische Operatoren

Mit logischen Operatoren können mehrere Bedingungen in einer Abfrage verknüpft werden. Es gibt den **UND-Operator** und den **ODER-Operator**.

Der &&-Operator (UND-Operator, AND-Operator) hat folgende Syntax:

```
(bedingung1) && (bedingung2)
```

Die Wahrheitstabelle des &&-Operators lautet:

bedingung1	bedingung2	bedingung1&&bedingung2
false	false	false
false	true	false
true	false	false
true	true	true

Beispiel:

```
if ((x>=20)&&(x<=40))
{
}
```

Der Anweisungsblock hinter `if` wird nur ausgeführt, wenn x größer gleich 20 und kleiner gleich 40 ist.

Der &&-Operator ist kommunikativ, d.h. b1&&b2 ist gleichbedeutend mit b2&&b1.

Aufgaben:

a.) Wann wird der if-Block ausgeführt? `if ((y==-2)&&(x>0)&&(z==x)) { }`

b.) Wann wird der if-Block ausgeführt? `if ((x>0)&&(x<0)&&(z==0)) { }`

c.) Wann wird der if-Block ausgeführt? `if (!(x>=0)&&!(y>=0)) { }`

Logische Operatoren

Der ||-Operator (ODER-Operator, OR-Operator) hat folgende Wahrheitstabelle:

bedingung1	bedingung2	bedingung1 || bedingung2
false	false	false
false	true	true
true	false	true
true	true	true

Beispiel:

```
if ((zahl<20) || (zahl>40))
     {
          cout << zahl;
     }
```

Regel:
Der Compiler untersucht die Einzelbedingungen einer Verknüpfung von links nach rechts.

Der OR-Operator ist auch kommunitativ, d.h. es gilt b1 || b2 ist gleichbedeutend mit b2 || b1.

Bedingungen können auch mit AND- und OR-Operatoren zusammengesetzt werden. Achten Sie dabei auf eine richtige Klammerung!

Beispiel:
Wann wird der if-Anweisungsblock ausgeführt?

```
if (((x<20) || (x>40)) && (x!=0));
     {     }
```

Übungen

Aufgabe 1

Entwerfen Sie bitte ein Struktogramm für ein (!) Programm mit zwei Variablen (z.B. `max, moritz`).

Für den Fall, daß beide Zahlen nicht gleich sind, soll deren Summe ermittelt und durch 2 dividiert werden.

Für den Fall, daß beide Zahl gleich sind, soll die Meldung erscheinen :

"Es gibt nichts zu tun"

Für den Fall, daß beide Zahl gleich sind und kleiner Null sind, soll die Meldung erscheinen :

"Es gibt nichts zu tun, die Zahlen sind kleiner Null"

Bedingungen, Rückgabewert

Um die Kompatibilität von C++ zu C zu gewährleisten, gilt folgende Regel:

> Eine Bedingung ist falsch, wenn sie den Wert 0 hat. Eine Bedingung ist wahr, wenn sie einen Wert ungleich 0 hat.

Obwohl die bool-Variablen `true` und `false` feste Werte haben (1 und 0), ist eine Bedingung also auch wahr, wenn sie z.B. den Wert 3.2 hat.

Beispiel:

```
if(x)
{
}
```

wird ausgeführt, wenn x ungleich Null ist. Wann wird `if(!x) { }` ausgeführt?

> Das Ergebnis einer Entscheidung kann der Rückgabewert einer Funktion sein. Sie sollten bei der Übergabe von wahr und falsch immer 1 oder 0 benutzen, auch wenn es im Falle von wahr auch anders geht (s.o.).

Beispiel:

```
bool positiv(int zahl)
{
     if (zahl > 0)
          return(1);          // wahr
     else
          return(0);          // falsch
}
```

Die `return`-Anweisung kann Ausdrücke verarbeiten. Können Sie die Funktion `positiv` vereinfachen?

Übung

Aufgabe:

Schreiben Sie eine Funktion `isSchaltjahr`, der eine Jahreszahl übergeben wird und die einen wahren Wert zurückliefert, wenn es sich um ein Schaltjahr handelt. Falls es kein Schaltjahr ist, soll ein falscher Wert zurückgeliefert werden.

Ein Jahr ist kein Schaltjahr, wenn die Jahreszahl nicht durch 4 teilbar ist. Ein Jahr ist ein Schaltjahr, wenn die Jahreszahl durch 4, nicht aber durch 100 teilbar ist. Ein Jahr ist ebenfalls ein Schaltjahr, wenn die Jahreszahl durch 4, durch 100 und durch 400 teilbar ist.

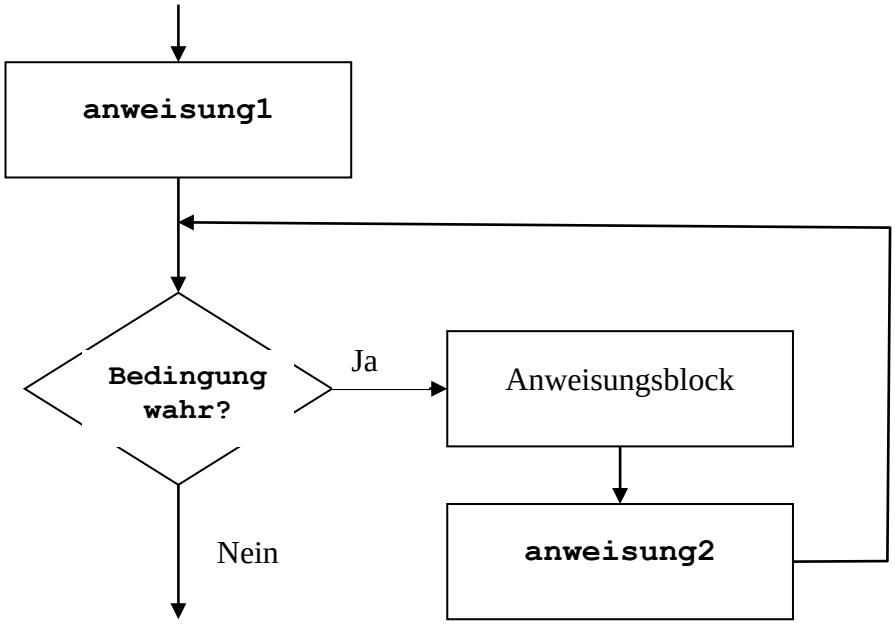
a.) Entwickeln Sie ein Struktogramm für eine `main`-Funktion, mit der Sie die Funktion `isSchaltjahr` überprüfen können.
b.) Entwickeln Sie ein Struktogramm für die Funktion i`sSchaltjahr`.
c.) Schreiben Sie den Code auf Papier.
d.) Lassen Sie Ihren Code von Ihrem Tauschpartner korrigieren.
e.) Programmieren Sie dann exakt diesen korrigierten Code.

II. 3. 2 Schleifen

Die grundlegende Form einer Schleife in C++ ist

```
for (anweisung1; bedingung; anweisung2)
{
}
```

Die `for`-Schleife als Flußdiagramm sieht folgendermaßen aus:



Übung:

Wie sehen die Symbole nach DIN 66001 aus?

Wie sieht das Struktogramm nach Nassi-Shneidermann für die `for`-Anweisung aus?

Schleifen

Welche Zahlen werden auf dem Bildschirm ausgegeben?

```
#include <iostream>
using namespace std;

int main()
{
     int x;
     for (x=1; x<=3; x++)
          {
               cout << "Wert x = " << x << endl;
          }
}
```

Es ist wie bei der `if`-Anweisung möglich, die geschweiften Klammern wegzulassen, wenn der Anweisungsblock nur eine Anweisung erhält.

Übung:
Schreiben Sie eine Schleife, die die Zahlen von 10 bis –40 herunterzählt und die Zahlen auf dem Bildschirm ausgibt.

An jeder Stelle der `for`-Anweisung können berechenbare Ausdrücke stehen.

Nennen Sie Beispiele!

Mehrere Anweisungen in `for`

Die beiden Anweisungen in `for` können jeweils aus mehreren Einzelanweisungen bestehen. Sie sind durch Kommata getrennt. Die Syntax lautet:

```
for (anw1a,anw1b,anw1c; bedingung; anw2a,anw2b,anw2c)
{
}
```

Beispiel:

```
for (x=zahl1,y=zahl2; x<=zahl2; x++,y--)
{
}
```

Verzicht auf Initialisierung

Es kann auf die Initialisierung der Zählvariablen verzichtet werden. Was liefert folgende Funktion?

```
#include <iostream>
using namespace std;

int main()
{
     int x;

     cout <<"Zahl eingeben :" <<endl;
     cin >> x;

     for ( ; x , x--)
          cout << "Wert x = " << x << endl;
}
```

Gleichzeitige Definition und Initialisierung

Man kann auch die Zählvariable in der `for`-Anweisung definieren. Beispiel:

```
for (int x=1; x<=10; x++)
{
}
```

Übung:
Schreiben Sie ein konventionelles Zählbeispiel, bei dem Sie testen, wie global oder lokal die Zählvariable x ist. Dann vergleichen Sie es mit dem obigen Beispiel.

Aufgaben

1.) Was machen folgende Schleifen?

a.) `for (x=1; x<100; x++)`
b.) `for (x=2; x>1; x++)`
c.) `for (x=1; x<20;)`
d.) `for ( ; 1 ;)`
e.) `for (;x<=8,x++) x-- ;`

Übungen

2.) Schreiben Sie ein Programm, das die Summe von 1+2+3+ ... +10 berechnet und ausgibt. Ändern Sie dann die 10 in eine 1000.

3.) Schreiben Sie das obige Programm um, so daß Sie nach dem Anfangs- und Endwert gefragt werden.

4.) Schreiben Sie ein Programm, das Ihnen die Summe 1+3+5+ ... +99 berechnet.

5.) Geben Sie im Programm 4.) einen Start- und Endwert ein. Ist der Startwert eine gerade Zahl, erhöhen Sie diese um 1. Ist der Endwert eine gerade Zahl, erniedrigen Sie diese Zahl um 1.

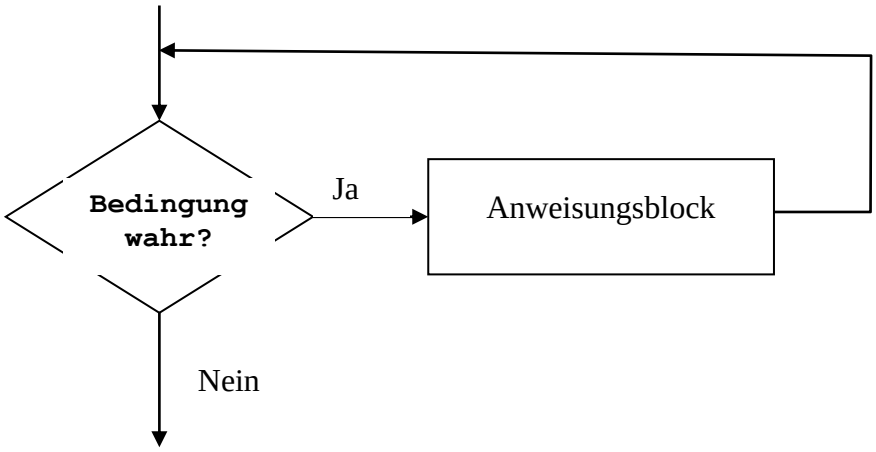
while

Die `while`-Schleife hat eine Struktur wie die `if`−Anweisung:

```
while (bedingung)
{
}
```

Ist die Bedingung wahr, wird der Anweisungsblock hinter `while` ausgeführt und dann wieder (!) die Bedingung geprüft, ob diese immer noch wahr ist. Solange also die Bedingung wahr ist, wird der Anweisungsblock ausgeführt. Ist die Bedingung falsch, so fährt das Programm hinter dem `while`-Anweisungsblock fort.

Das Flußdiagramm sieht folgendermaßen aus:



Übung:

Wie sehen die Symbole nach DIN 66001 aus?

Wie sieht das Struktogramm nach Nassi-Shneidermann für die `while`-Schleife aus?

while

Beispiel:

```
int main()
{
     int zahl=0;

     while (zahl != 200)
          {
               cout << "Bitte die Zahl 200 eingeben:";
               cin >> zahl;
          }
}
```

Übung:

Schreiben Sie folgende `for`-Anweisung in eine `while`-Schleife um:

```
for (x=5; x<=25; x++)
     cout << " x= " << x << endl;
```

while ohne Anweisungsblock

Oft wird while ohne einen dazugehörigen Anweisungsblock benutzt, z.B. um auf ein bestimmtes Ereignis zu warten (s. Beispiel n.S.)

while, do while

Was bewirkt folgendes Programm?

```
#include <iostream>
using namespace std;

int getzahl (void);

int main()
{
     while (getzahl() != 250);
}

int getzahl (void)
{
     int zahl;

     cout <<"Bitte 250 eingeben:" ;
     cin >> zahl;
     return(zahl);
}
```

do while Anweisung

Die Syntax der `do while`-Anweisung sieht folgendermaßen aus:

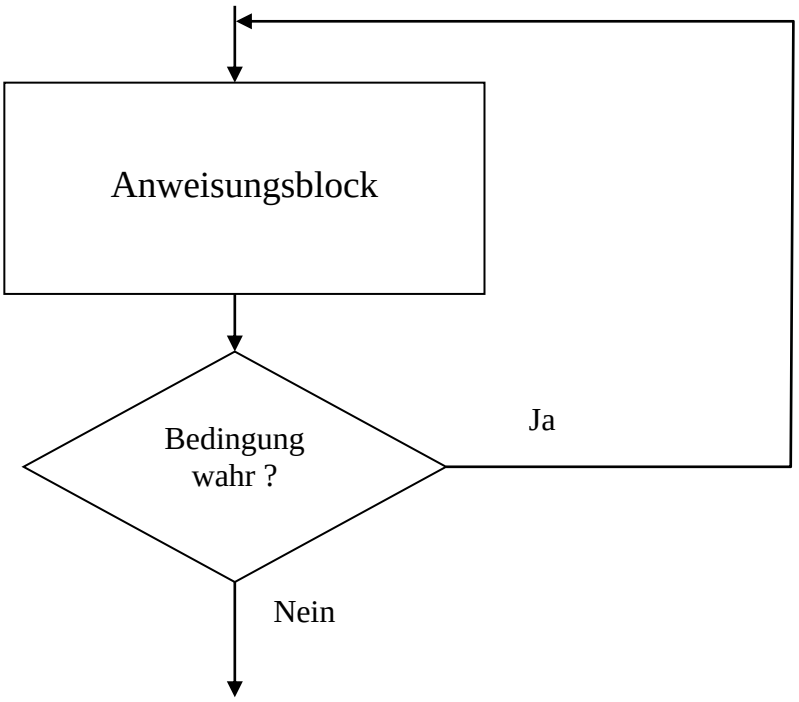
```
do
     {
     }
while (bedingung);
```

Zuerst wird auf jeden Fall der Anweisungsblock zwischen `do` und `while` ausgeführt. Ist dann die Bedingung wahr, wird wieder zum `do` gesprungen und der Anweisungsblock ausgeführt; und zwar solange, bis die Bedingung falsch ist.

do while

Übung:

Zeichnen Sie das Flußdiagramm der `do while`-Schleife.



Beachten Sie das Semikolon hinter der `while`-Bedingung !

Übung:

Schreiben Sie folgende `for`-Anweisung in eine `do while`-Schleife um:

```
for (x=5; x<=25; x++)
     cout << " x= " << x << endl;
```

Übung

Augaben

1.) Wie wirken folgende Schleifen?

a.) `while(1)`
b.) `do { } while (0);`
c.) `while(x)`

2.) Ein Programm soll entwickelt werden, das Sie nach einer bestimmten Zahl z1 fragt. Danach soll eine zweite Zahl so lange eingegeben werden, bis diese Zahl gleich z1 ist.
a.) Stellen Sie das Struktogramm dar.
b.) Realisieren Sie den Code des Programms mit einer `do while`-Schleife.
c.) Programmieren Sie Ihr Ergebnis.

3.) Der Mathematiker Leibnitz hat herausgefunden, daß die Zahl ; folgendermaßen berechnet werden kann:

; = (1-1/3+1/5-1/7+1/9-1/11+1/13- ...)*4

Schreiben Sie ein Programm, das nach diesem Verfahren ; berechnt. Vor der Berechnung soll gefragt, wie weit die Reihe gebildet werden soll, damit keine unendliche Schleife entsteht.

continue

Die `continue`-Anweisung kann nur innerhalb von `for-, do-` und `while-` Anweisungen benutzt werden.

Die `continue`-Anweisung springt zum Kopf der innersten `for-, do-` oder `while-` Anweisung, in der sie vorkommt. Danach arbeitet die Schleife normal weiter.

Übung:

Welche Ausgabe erzeugt das folgende Programm?

```
#include <iostream>
using namespace std;

int main()
{
     int x,y;
     for (x=1; x<=3;x++)
     {
          for (y=1; y<=5; y++)
               {
                    if((x==2) && (y==3))
                         {
                         cout << "continue" << endl;
                         continue;
                         }
                    cout << x << ":" <<y << endl;
               }
     }
}
```

break

Der `break`-Befehl ist gegensätzlich zum `continue`-Befehl.

Die `break`-Anweisung springt **hinter** den innersten `do-, for-, switch-` oder `while`-Anweisungsblock, in dem sie steht.

Das vorige Beispiel ein wenig verändert:

```
#include <iostream>
using namespace std;

int main()
{
     int x,y;
     for (x=1; x<=3;x++)
     {
          for (y=1; y<=5; y++)
               {
                    if((x==2) && (y==3))
                         {
                         cout << "break" << endl;
                         break;
                         }
                    cout << x << ":" <<y << endl;
               }
     }
}
```

Was gibt das Programm aus?

switch

Mit dem `switch`-Befehl kann eine Mehrfachselektion, d.h. eine Fallunterscheidung vorgenommen werden. Zusammen mit der `case`-Anweisung wird die Wirkungsweise am einfachsten verständlich. (case (engl.) = Fall)

```
switch (variable)
{
     case 1:

     case 17:

     case -8:
}
```

Falls variable den Wert 1, 17 oder –8 hat, wird die `case`-Marke angesprungen und die nachfolgenden Anweisungen alle (!) ausgeführt.

Übung:
Was gibt nachfolgendes Programm aus je nach Eingabewert aus? Zeichnen Sie das Struktogramm nach Nassi-Shneidermann.

```
int zahl;
cout << "Zahl eingeben: " << endl;
cin >> zahl;

switch (zahl)
{
    case 1:   cout << "1. Text." << endl;

    case 17:  cout << "2. Text." << endl;

    case -3:  cout << "3. Text" << endl;
}
cout << "4. Text" << endl;
```

switch

`case` ist nur eine Sprungmarke. Alle (!) nachfolgenden Anweisungen werden ausgeführt.

Damit nur die zur case-Anweisung zugehörigen Befehle ausgeführt werden, ist die break-Anweisung notwendig.

Übung:

Was gibt dieses Programm aus? Zeichnen Sie nun das Struktogramm.

```
int zahl;
cout << "Zahl eingeben: " << endl;
cin >> zahl;

switch (zahl)
{
    case 1:   cout << "1. Text." << endl;
              break;

    case 17:  cout << "2. Text." << endl;
              break;

    case -3:  cout << "3. Text" << endl;
              break;
}
cout << "4. Text" << endl;
```

Die `break`-Anweisung springt **hinter** den innersten `do-, for-, switch-` oder `while`-Anweisungsblock, in dem sie steht.

default

Es können mehrere `case`-Anweisungen das gleiche Sprungziel haben.

Beispiel:

```
switch (x)
{
     case 0:
     case 1:   cout << " 0 oder 1 gewählt" << endl;
               break;

     case 2:   cout << " 2 gewählt" << endl;
               break;
}
```

Die **`default`**-Marke ist eine besondere Sprungmarke für den Fall, daß die Switch-Variable auf keinen case-Fall zeigt.

Existiert für einen Wert keine `case`-Anweisung, wird – falls vorhanden – die `default`-Sprungmarke angesprungen.

```
switch (x)
{
     case 0:
     case 1:   cout << " 0 oder 1 gewählt" << endl;
               break;

     case 2:   cout << " 2 gewählt" << endl;
               break;

     default:  cout << "Falsche Zahl gewählt" << endl;
}
```

Übung

Aufgabe:

Sie sollen ein "Kneipen-Entscheidungsprogramm" entwickeln unter folgenden Annahmen:

- Sie geben Ihren zur Verfügung stehenden Geldbetrag ein.
- Es gibt drei Kneipen A, B, C mit folgenden Bierpreisen:

  A: 4,80 DM
  B: 4,20 DM
  C: 5.80 DM

- Berechnen Sie die Biere, die Sie in A, B und C trinken können und geben Sie die Ergebnisse aus.
- Dann sollen Sie gefragt werden, für welche Kneipe Sie sich entscheiden.
- Falls Sie Autofahrer sind, trinken Sie in Ihrer gewählten Kneipe 2 Bier, sonst 5.
- Wieviel Geld geben Sie dabei aus? Das Ergebnis soll angezeigt werden.