4. Kontrollstrukturen

4.1.Verzweigungen

4.1.1. If-else

4.1.1.1. Definition und Anwendung

Die Kontrollstruktur if-then-esle sieht folgendermassen aus:

Beispiel 1

```
if (ausdruck) {
   falls ausdruck == true, dann wird dieser code
ausgeführt
}
else {
   falls ausdruck == false, dann dieser code
}
```

Die Struktur funktioniert so: der Ausdruck bei der *if* –Anweisung wird ausgewertet; falls dieser *true* ist, dann wird der erste Teil des Codes ausgeführt, falls es *false* ist, dann der code im *else*-Block. Den else–Block kann man auch auslassen; in diesem Fall wird nichts gemacht und der Code nach der *if*-Anweisung ausgeführt.

Hier ein Beispiel:

Beispiel 2

```
#include <iostream.h>

void main()
{
   const float LIMIT=80;
   float geschwindigkeit;

   cout << "Bitte geben Sie die Geschwindigkeit ein: ";
   cin >> geschwindigkeit;

   if ( geschwindigkeit > LIMIT )
      cout << "Zu schnell gefahren." << endl;
   else
      cout << "Nicht zu schnell gefahren." << endl;
};
```

Im Ausdruck selbst benutzt man Anweisungen, die mit *true* oder *false* bewertet werden können (Ergebnisse von verschiedenen boolschen Operationen). Falls es sich um einen arithmetischen Ausdruck handelt, dann wird eine Null als Ergebnis als *false* interepretiert und alles andere als *true*.

Man kann *if*-Anweisungen auch verschachteln:

Beispiel 3

```
#include <iostream.h>

void main()
{
   const float LIMIT=80;
   float geschwindigkeit;
```

```
   cout << "Bitte geben Sie die Geschwindigkeit ein: ";
   cin >> geschwindigkeit;

   if ( geschwindigkei t> LIMIT )
      if ( geschwindigkeit > LIMIT+30 )
         cout << "Zu schnell gefahren. Punkte in Flensburg!" << endl;
      else
         cout << "Zu schnell gefahren. Verwarnungsgeld!" << endl;
   else
      cout << "Nicht zu schnell gefahren." << endl;
};
```

oder:

Beispiel 4

```
bool istSchaltjahr(int jahr)
{
   if ( jahr%400 == 0 ) return true;
   else if( jahr%100 ==0) return false;
   else if(jahr%4==0) return true;
   else return false;
};
```

4.1.1.2. das *else* dem richtigen *if* zuordnen

Wir wollen uns nun ein paar Beispiele ansehen:

Beispiel 5

```
   if(geschwindigkeit>LIMIT)
      if(geschwindigkeit>LIMIT+30)
         cout<<"Zu schnell gefahren. Punkte in Flensburg!"<<endl;
   else
      cout<<"Nicht zu schnell gefahren."<<endl;
```

Das else wird in diesem Fall dem ersten if zugeordnet (warum?). Falls aber das else dem zweiten gehört, müssen wir den code ein bisschen umformen. Dazu gibt es zwei Möglichkeiten:

Beispiel 6

```
   if(geschwindigkeit>LIMIT)
   {
      if(geschwindigkeit>LIMIT+30)
         cout<<"Zu schnell gefahren. Punkte in Flensburg!"<<endl;
   }
   else
      cout<<"Nicht zu schnell gefahren."<<endl;
```

oder

Beispiel 7

```
   if(geschwindigkeit>LIMIT)
      if(geschwindigkeit>LIMIT+30)
         cout<<"Zu schnell gefahren. Punkte in Flensburg!"<<endl;
      else
         ;
   else
      cout<<"Nicht zu schnell gefahren."<<endl;
```

4.1.1.3. Mehrere Bedingungen

Man kann im Ausdruck natürlich auch mehrere Bedingungen schreiben: es gilt dieselbe Regel. Falls der ganze Ausdruck *true* ist, dann wird der erste Code ausführt; falls es *false* ist, dann der *else*-Code. Hier ein Beispiel:

Beispiel 8

```
if ((var 1 == 0) && ( (var2 < 5) || (var3 > 10) ) ) ...
```

4.1.2. Switch-case

Hier ein Beispiel für einen einfachen Taschenrechner:

Beispiel 9

```
#include <iostream.h>

void main()
{
   double zahl1;
   double zahl2;
   int operation;
   cout<<"Primitives Taschenrechner-Programm"<<endl;
   cout<<"Zahl1: ";
   cin>>zahl1;
   cout<<"Zahl2: ";
   cin>>zahl2;
   cout<<"Operation (1=+,2=-,3=*,4=/): ";
   cin>>operation;
   switch(operation)
   {
   case 1:
      cout<<zahl1+zahl2<<endl;
      break;
   case 2:
      cout<<zahl1-zahl2<<endl;
      break;
   case 3:
      cout<<zahl1*zahl2<<endl;
      break;
   case 4:
      cout<<zahl1/zahl2<<endl;
      break;
   };
};
```

Der *switch-case*-Anweisung wird hinter dem Schlüsselwort *switch* in runden Klammern eine Variable übergeben. Diese Variable - und das ist eine große Beschränkung von *switch-case*-Anweisungen - muss von einem Standard-Datentyp sein. Es darf sich hierbei auch um kein Array handeln.

Die *break*-Befehle sind notwendig, damit der Computer die Abarbeitung der Befehle im *switch*-Block beendet und hinter die schließende geschweifte Klammer springt. Mit dem Weglassen dieser *break*-Befehle kann man dafür sorgen, daß für mehrere Konstanten derselbe Code ausgeführt wird.

Es ist auch möglich, einen Zweig anzugeben, der durchlaufen werden soll, wenn keiner der Werte zutrifft. Das Schlüsselwort hierfür heißt *default*.

Beispiel 10

```
   case 8:
   case 9:
   case 0:
      cout<<"Diese Ziffer ist nicht belegt."<<endl;
      break;
   default:
      cout<<"Das war noch nicht einmal eine Ziffer."<<endl;
   };
```

4.1.3. Bedingungsoperator

Die Definition des Operators sieht so aus:

Beispiel 11

```
Erg = (Ausdruck1) ? Ausdruck2 : Ausdruck3;
```

Hier ein Beispiel zu seiner Anwendung:

Beispiel 12

```
var2 = (var1>0) ? var1 : -var1;
```

Alternativ muss man hier den folgenden Code schreiben:

Beispiel 13

```
if (var1>0)
    var2 = var1;
else
    var2 = -var1;
```

4.2.Schleifen

4.2.1. for-Schleife

Sollen eine oder mehrere Anweisungen wiederholt ausgeführt werden, so werden hierfür Schleifen eingesetzt.

Die *for*-Schleife wird hauptsächlich immer dann eingesetzt, wenn bereits vor dem Eintritt in die Schleife bekannt ist, wie oft die in ihr enthaltenen Anweisungen ausgeführt werden sollen. Die *for*-Schleife hat folgenden Syntax:

Beispiel 14

```
for (Ausdruck1; Ausdruck2; Ausdruck3) {
    Aktion;
      }
```

Der erste Ausdruck in den Klammern ist der Initialisierungsausdruck. Dort wird der Typ der Schleifenvariable und ihr Anfangswert angegeben:

Beispiel 15

```
   for (int index = 0; ...)
```

oder aber auch:

Beispiel 16

```
   for (char buchstabe = ´a´; ...)
```

Der zweite Ausdruck bestimmt das Ende der Schleife, d.h. wann die Schleife verlassen wird:

Beispiel 17

```
   for (int index = 0; index < 100; ...) ...
   for (int index = 100; index > 4; ..) ...
```

Der erste Beispiel wird z.B. folgendermassen interpretiert: für die Variable index des Typs int mit Anfangswert 0 wird die Schleife solange durchgelaufen, bis index >= 100 wird. Dann wird es abgebrochen.

Der dritte Ausdruck gibt die Veränderung der Schleifenvariable an, z.B.:

Beispiel 18

```
   for (int index = 0; index < 100; index ++) ...
   for (int index = 0; index < 100; index += 10)
   for (int index = 100; index > 0; index --)
   for (int index = 1; index < 100; index *= 3)
   for (int index = 0; index < 1000; index = var_1*index + 56)
```

Man kann pro Ausdruck auch mehrere Anweisungen angeben, die aber durch Kommata getrennt werden. Z.B.:

Beispiel 19

```
for (int index = 0, bool done = false; (index < 100) && (!done); index ++) …
```

Hier ein vollständiger Beispiel:

Beispiel 20

```
#include <iostream>
#include <cstdlib>

int main()
{
  char Buffer[3];

  for (int i = 0; i < 3; ++i)
  {
    std::cout << "Geben Sie eine Zahl ein: " << endl;
    std::cin >> Buffer[i];
  }
}
```

4.2.2. While

Außer der beschriebenen *for*-Schleife kennt C++ noch die *while*-Schleife. Während die *for*-Schleife immer dann eingesetzt wird, wenn schon vor dem Eintritt in die Schleife die Abbruchbedingung bekannt ist, so wird die *while*-Schleife dagegen hauptsächlich dann eingesetzt, wenn die Abbruchbedingung erst innerhalb der Schleife selbst festgestellt werden kann.

Bemerkung. Jede *while*-Schleife kann durch eine *for*-Schleife umgetauscht werden; man muss nur darauf achten, wann der erste Durchlauf ausgeführt wird und auf die Endbedingung.

Allgemein benutzt man die *while*-Schleife so:

Beispiel 21

```
....
bool done = false;
while (!done)
{
    .... // Hier irgendwann done auf
    .... // auf true setzen damit Schleife
    ...  // beendet wird.
}
```

Und hier ein Beispiel:

Beispiel 22

```
#include <iostream.h>

void main()
{
   int i=1;
   while(i<=10)
   {
      cout<<"Schleifendurchlauf Nummer "<<i<<endl;
      i=i+1;
   };
};
```

4.2.3. Do-while

Diese Schleife wird immer mindestens einmal durchlaufen, da das Abbruchkriterium erst am Ende der Schleife abgeprüft wird. Allgemein sieht die Schleife so aus:

Beispiel 23

```
bool done = false;
do
{
    .... // Hier irgendwann done auf
    .... // auf true setzen damit Schleife
    ...  // beendet wird.
} while (!done);
```

Und hier ein Beispiel:

Beispiel 24

```
#include <iostream.h>
void main()
{
   int eingabe;
   int summe=0;
   do
   {
      cout<<"Bitte geben Sie eine Zahl ein: ";
      cin>>eingabe;
      summe=summe+eingabe;
   }while(eingabe!=0);
   cout<<"Die Summe dieser Zahlen ist: "<<summe<<endl;
};
```

4.3.Kontrolltransferanweisungen

4.3.1. Break

Die *break*-Anweisung darf nur innerhalb einer *for*- oder *while*-Schleife oder in einem *case*-Zweig der *switch-case* Verzweigung stehen. Sie bewirkt das sofortige Verlassen der

aktuellen Schleife oder der *switch-case* Verzweigung. Sind mehrere Schleifen verschachtelt, so wird nur die aktuelle Schleife abgebrochen, die äußeren Schleifen werden nicht beeinflusst. Hier ein Beispiel:

Beispiel 25

```
...
for (int index = 0; index < 20; index ++) {
   if (done) break;
   ...
}
```

In diesem Fall wird die Schleife vorzeitlich abgebrochen, falls *done* auf true gesetzt wird. Sonst wird die ganze Schleife (20 Mal) durchgelaufen.
Die Benutzung von break-Anweisungen in einem *switch-case* Block haben wir schon gesehen.

4.3.2. Continue

continue bewirkt dagegen den Abbruch nur des laufenden Schleifendurchlaufs. Das Programm geht dann mit dem nächsten Durchlauf weiter. Hier ein Beispiel:

Beispiel 26

```
for (int index = 0; index < 20; index ++) {
   if ((index == 5) || (index == 7)) continue;
   ...
}
```

In diesem Fall wird die Schleife bis zum Ende durchgelaufen, aber die Durchläufe, bei denen *index* gleich 5 oder 7 ist, nicht.

4.3.3. return

Die *return*-Anweisung bewirkt das Verlassen der aktuellen Funktion. Falls dies die main-Funktion ist: das Verlassen des Programms. Man benutzt es entweder zum frühzeitigen Verlassen der Funktion oder für Ergebniss-Übergabe. Mehr dazu bei den Funktionen.

4.4.Aufgaben

4.4.1. Verändern Sie wieder unseren Tacshenrechner: es soll so wenig Code enthalten, wie möglich. Benutzen Sie dafür die switch-case-Anweisung und die while-Schleife.

4.4.2. Schreiben Sie ein Programm, dass 20 Buchstaben einliest.

4.4.2.1. mittels einer for-Schleife

4.4.2.2. mittels einer do-while-Schleife

4.4.2.3. mittels einer while-Schleife

4.4.2.4. mittels einer for-Schleife. Falls ein ‚X' eingegeben wird, muss das Programm enden.

4.4.2.5. mittels einer for-Schleife. Falls ein ‚X' eingegeben wird, muss das Programm es übersprigen und mit dem nächsten Buchtaben weiterarbeiten.

4.4.3. Schreiben Sie ein Programm, das ein Menü simuliert: verschiedene Zahlen entsprechen verschiedenen Menüpunkten. Bei der Auswahl einer Zahl soll das Menüpunkt ausgegeben werden. Benutzen Sie dafür die switch-case-Anweisung. Probieren Sie aus, was passiert, falls Sie die break-Anweisungen weglassen.