5. Abend

Ziele :

- Wir kennen die Bedeutung und die Anwendung von Operatoren und wissen welche Operatoren vor anderen Vorrang haben. Wir sind dadurch in der Lage Berechnungen in unsere Programme einzubauen und zu verstehen in fremden Programmen was genau berechnet wird.
- Wir können ab heute Abend Teile/Blöcke von unserem Code nur unter gewissen Bedingungen ausführen lassen. Wir sind ab heute ebenso in der Lage Codeblöcke mehrmals als Teil einer Schleife ausführen zu lassen. Dadurch gewinnen wir die Möglichkeit unser Programm zu strukturieren und "intelligentere" Programme zu erstellen.

Operatoren für elementare Datentypen

Binäre arithmetische Operatoren

werden bei Berechnungen mit zwei (darum das Wort binär) Operatoren verwendet. Hier einige Beispiele :

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

int main()
{
    int a = 0;
    int b = 10;
    // Addition mit zwei
    // Variablen
    int c = a + b;

    // Addition mit zwei
    // konstanten Werten
    c = 33 + 34;
    // Subtraktion und
    // Multiplikation sind
    // genauso möglich

    // Division und Modulo-Op
    // sind ein wenig anders
    c = 12 / 5;
    // Division von zwei Ganzzahlen
    // ergibt als Ergebnis auch
    // eine Ganzzahl
    cout << c << endl;
    // Ausgabe : 2
```

```
    // Die Modulo-Op rechnet den
    // ganzzahligen Rest der
    // Division aus und funktioniert
    // nur mit Ganzzahlen
    c = 13 % 5;
    // = 2 Rest 3
    cout << c << endl;
    // Ausgabe : 3

    return 0;
}
```

Unäre Operatoren

wirken auf nur einen Operator

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

int main()
{
    int a(0);
    int b(10);

    // prä-inkrement, also auswerten
    // bevor der Ausdruck verwendet
    // wird
    int c = ++a;

    // post-inkrement, zuerst Ausdruck
    // verwenden, danach inkrementieren
    c = b++;

    return 0;
}
```

Zuweisungen

Mit einer Zuweisung kann einer Variablen ein Wert gegeben werden.
Mit einer zusammgesetzten Zuweisung kann man gewissen Ausdrücke ein wenig kürzer schreiben. Betrachte hierfür das Beispiel :

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

int main()
{
```

```
    // Die Variable a erzeugen
    // und direkt initialisieren
    int a = 0;

    // zuerst wird addiert (4+5)
    // das Ergebnis wird dann
    // der Variable a zugewiesen
    a = 4 + 5;
    // Ausgabe : 9
    cout << a;

    // Zusammengesetzte Zuweisung
    a += 5;
    // a wird gleich a + 5
    // die Zeile ist also gleichbedeutend
    // wie :
    a = a + 5;


    return 0;
}
```

Vergleichsoperatoren

dienen dem Vergleich zweier Operanden (Variablen oder Werte), siehe auch die boole'schen Ausdrücke von Abend 3. Das Ergebnis eines Vergleiches ist immer vom Datentyp **bool**.

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

int main()
{
    // Die Variable a erzeugen
    // und direkt initialisieren
    bool a = false;

    // zuerst wird der Vergleich
    // durchgeführt, danach wird
    // das Ergebnis der Variable
    // a zugewiesen.
    a = 4 < 5;

    // Zuerst wird 5 mit 6
    // verglichen. BEACHTE
    // ein Vergleich ist immer
    // mit ZWEI Gleichheits-
```

```
    // zeichen !
    a = 5 == 6;

    int b = 55;
    int x = 55;
    // Mit Klammern wird es
    // evtl. besser lesbar
    a = (b == x);

    return 0;
}
```

Logische Operatoren

Die logischen Operatoren && (logisches UND), || (logisches ODER) und ! (logisches NICHT) werden verwendet, wenn man zusammgesetzte Bedingungen formulieren will. Das Ergebnis einer logischen Operation ist auch immer true oder false, also vom Datentyp **bool.**

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

int main()
{
    // Die Variable a erzeugen
    // und direkt initialisieren
    bool a = false;

    // a wird true falls 5 gleich 6 ist
    // ODER
    // 4 kleiner als fünf ist
    a = (5 == 6) || (4 < 5);

    int b = 55;
    int x = 55;
    a = (b == x) && (2 < 1);
    // wird true falls b gleich x ist
    // UND
    // 2 kleiner als 1 ist

    bool z = !a;
    // falls a true ist wird z false
    // sonst wird z true

    return 0;
}
```

Kontrollstrukturen

Die if-Anweisung

Sie dient dazu einen Code-Block nur dann auszuführen, wenn eine Bedingung erfüllt ist. Die Bedingung ist dann erfüllt, wenn der **Ausdruck in der Klammer nach dem if ungleich 0** ist. Das heisst auch ein boole'scher Ausdruck kann in der Klammer stehen, denn **false=0, true=1.** Beispiel:

```
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{
    cout << "Gib den Dividenden ein" << endl;

    // neue Variable erzeugen
    double dividend;
    // und von der Tastatur
    // einlesen
    cin >> dividend;

    cout << "Gib den Divisor ein" << endl;

    double divisor;
    cin >> divisor;

    // Vergleich des divisors
    // mit 0 (0.0 als double)
    if(divisor != 0.0)
    {
        // Dieser Block wird
        // nur ausgeführt, falls
        // die if-Bedingung in der
        // Klammer nicht 0 (true) ist
        double ergebnis = dividend / divisor;

        cout << dividend << " / " << divisor;
        cout << " = " << ergebnis << endl;
    }
    cout << "Taste";
    getch();

    return 0;
}
```

Mit else kann ein Code-Block definiert werden, der abgearbeitet wird, wenn die Bedingung nicht erfüllt ist.
Beispiel:

```
#include <iostream>
using namespace std;
int main()
{
 // boole'scher Ausdruck
 if(3 == 4)
 {
  cout << "Das wird NIE !!!";
  cout << "ausgegeben" << endl;
 }
 else
 {
  cout << "Das wird ";
  cout << "ausgegeben" << endl;
 }
 return 0;
}
```

Die while-Schleife

Die while Schleife ist eine Schleife bei der zuerst ein Ausdruck geprüft wird und falls dieser ungleich 0 ist der Code im darauffolgenden Block ausgeführt wird. Beispiele:

```
int main()
{
    int a = 5;

    // solange a ungleich 0 ist
    while(a)
    {
        // a um eins erniedrigen
        a--;
    }

    a = 3;
    // und jetzt die ewige Schleife
    while(3 == a)
    {
        int z = 0;
    }

    return 0;
}
```

Die for-Schleife

Die for-Schleife wird häufig für Schleifen mit festen Zählern verwendet. Zum Beispiel um 100 mal "Hallo" auszugeben, oder 1000 mal eine Berechnung durchzuführen.
Hier die Syntax :
for(Ausdruck1; Ausdruck2; Ausdruck3)
Ausdruck1 heisst Initialisierung und wird nur einmal vor der Schleife ausgeführt.
Ausdruck2 heisst Laufbedingung, die Schleife wird ausgeführt fallst diese true ist.
Ausdruck3 heisst Reinitialisierung, die am Ende der Schleife jedesmal ausgeführt wird.
Eine for-Schleife kann problemlos in eine while-Schleife verwandelt werden:

```
int main()
{
    // Ausdruck1, Initialisierung
    int a = 0;

    // Ausdruck2, Laufbedingung
    while(a < 5)
    {
        // Code hier
        // einfügen

        //Ausdruck3, Reinitialisierung
        ++a;
    }

    // Hier als for
    for(int b = 0; b < 5; ++b)
    {
        // Code hier
        // einfügen
    }

    return 0;
}
```

Die do-while-Schleife

Bei dieser Schleife wird der Code im Schleifenblock mindestens einmal ausgeführt, da die Laufbedingung erst am Ende der Schleife geprüft wird.

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

    do
    {
        a--;
    }
    while(a > 0);

    return 0;
}
```