6. Abend

Ziele :

- Wir ergänzen unser Wissen über Kontrollstrukturen **(if-else; while; do-while; for-next)** mit der Möglichkeit bei mehreren Alternativen mit der **switch-**Anweisung lange if-else Ketten zu vereinfachen.
- Der C++-Compiler bietet die Möglichkeit gewisse Texte oder Zeichen automatisch zu ersetzen. Diese Möglichkeit werden wir ab jetzt mittels sogenannten Makros ausnutzen können.
- Wenn wir verschiedene Datentypen wie double und int in einer Berechnung mischen, wandelt der Compiler diese Datentypen implizit um. Wir werden die Regeln nach denen das geschieht kennenlernen.
- Um ganze Zeichenketten wie Namen oder Text zu speichern bietet der C++-Standard spezielle Hilfsmittel an, die wir ab heute Abend einsetzen können.

switch (Kontrollstrukturen)

Wollen wir aufgrund des Wertes einer numerischen Variable eine bestimmte Aktion aus einer Auswahl von möglichen Aktionen auswählen, können wir das mittels einer verketteten if-else-Anweisung.

```
#include <iostream>

using namespace std;

int main()
{
    // kleines Menu ausgeben
    cout << "[1] Tue dies" <<   endl;
    cout << "[2] Tue das"  <<   endl;
    cout << "[3] Tue jenes" <<  endl;
    cout << "[4] Tue nichts" << endl;

    int Eingabe = 0;

    // wir überprüfen die Eingabe des
    // Benutzers und fordern
    // ihn auf etwas einzugeben, solange
    // die Eingabe kleiner als 1 ist
    // oder die Eingabe grösser als 4 ist
    while((Eingabe < 1) || (Eingabe > 4))
    {
        cout << "Bitte Zahl zwischen 1 und 4 eingeben : " <<
endl;
        cin >> Eingabe;
```

```
    }

    if(1 == Eingabe)
    {
        cout << "Ich tue dies" << endl;
    }
    else if(2 == Eingabe)
    {
        cout << "Ich tue das" << endl;
    }
    else if(3 == Eingabe)
    {
        cout << "Ich tue jenes" << endl;
    }
    else if(4 == Eingabe)
    {
        cout << "Ich tue nichts" << endl;
    }
    else
    {
        // das ist eigentlich
        // nicht möglich
        cout << "Hä ?" << endl;
    }

    return 0;
}
```

Bei numerischen Wertevergleichen wie diesen ist dies meistens einfacher mittels einer **switch-case** möglich. Bei unserem Beispiel oben ist der Ausdruck, den wir prüfen "Eingabe". Dieser Ausdruck kommt bei einer **switch-**Anweisung in die Klammer nach dem switch :

switch(Eingabe)

danach kommen die einzelenen Konstanten, mit denen man den Ausdruck in Klammern vergleicht. Am besten wir betrachten das am umgewandelten Beispiel von oben:

```
#include <iostream>

using namespace std;

int main()
{
    // kleines Menu ausgeben
    cout << "[1] Tue dies" <<   endl;
```

```
    cout << "[2] Tue das"  <<   endl;
    cout << "[3] Tue jenes" <<  endl;
    cout << "[4] Tue nichts" << endl;

    int Eingabe = 0;

    // wir überprüfen die Eingabe des
    // Benutzers und fordern
    // ihn auf etwas einzugeben, solange
    // die Eingabe kleiner als 1 ist
    // oder die Eingabe grösser als 4 ist
    while((Eingabe < 1) || (Eingabe > 4))
    {
        cout << "Bitte Zahl zwischen 1 und 4 eingeben : " <<
endl;
        cin >> Eingabe;
    }

    switch(Eingabe)
    {
        // falls Eingabe = 1
        case 1:
            cout << "Ich tue dies" << endl;
            // mit break verhindern, dass
            // die unteren Anweisungen
            // auch asgeführt werden.
            break;
        case 2:
            cout << "Ich tue das" << endl;
            break;
        case 3:
            cout << "Ich tue jenes" << endl;
            break;
        case 4:
            cout << "Ich tue nichts" << endl;
            break;
        default:
            // sollte eigentlich
            // nie geschehen
            cout << "Hä ?" << endl;
            break;
    }

    return 0;
}
```

Beachte die Doppelpunkte nach den Konstanten (1, 2, 3, 4) und nach dem **default.** Ist die Eingabe also zum Beispiel 2 wird in den Code verzweigt, der direkt nach dem Doppelpunkt hinter der 2 kommt. Damit nicht noch mehr Code ausgeführt wird, der danach kommt, wird mit dem **break** aus der **switch** herausgesprungen. Die Anweisung nach dem default wird ausgeführt, wenn keine der Konstanten vor dem Doppelpunkt gleich der Anweisung in der switch-Klammer ist.
Wir können uns einmal ansehen was geschieht, was geschieht wenn das break weggelassen wird und dass es möglicherweise sogar Sinn machen kann.

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

    cin >> Eingabe;

    switch(Eingabe)
    {
        case 5:
        case 0:
            cout << "Du gehst aufs Ganze" << endl;
        case 4:
            cout << "Hallo 4" << endl;
        case 3:
            cout << "Hallo 3" << endl;
        case 2:
            cout << "Hallo 2" << endl;
        case 1:
            cout << "Hallo 1" << endl;
        default:
            cout << "Tschau" << endl;
            break;
    }

    return 0;
}
```

Hier wird die 5 gleich behandelt wie die 0. Zusätzlich wird bei keinem case aus der switch-Anweisung herausgesprungen, so dass alle darauffolgenden Anweisungen auch ausgeführt werden.

Textersetzung mit #define (Makros)

Wir können mittels der **#define** "Präprozessor"-Direktive im Code gewisse Dinge ersetzen lassen. Der Präprozessor gehört zum Compiler und ersetzt alle **#include** mit den jeweiligen angegebenen Header-Dateien sowie alle **#define** mit den gewählten Bausteinen.

```
// überall wo nachher PI im Code
// steht, wird das vom Präprozessor
// durch den Wert 3.14159 ersetzt
#define PI 3.14159

int main()
{
    cout << PI << endl;

    return 0;
}
```

Im Code wird also "PI" ersetzt durch das was beim **#define** angegeben ist.
Häufig werden auf diese Art Konstanten definiert, das PI ist ein gutes Beispiel dafür.
ICH RATE DAVON EHER AB. Wenn ich Pi als Konstante brauche, dann sollte Pi auch als solche definiert werden, denn dadurch wird auch gleich der Datentyp für Pi festegelegt :

```
// PI besser als double-Konstante
const double PI = 3.14159;
```

Es ist auch möglich bei solchen Makros Parameter zu verwenden :

```
#define Ausgabe(x) (cout << x)
int main()
{
   Ausgabe(55);
   return 0;
}
```

Aber auch hier : Wenn Du eine Funktion brauchst, dann definiere eine !

Typumwandlungen

Buch Kapitel 8

Die Standardklasse string

Namen, Texte und andere Zeichenketten bestehen aus aneinandergereihten Zeichen (z.B. vom Typ char). Diese Zeichenketten bezeichen wir ab jetzt als **string**'s.
Bis jetzt waren wir eigentlich nur in der Lage einzelne Zeichen (mit **char**) oder einzelne Zahlen (**int** oder **double**, etc.) zu verwalten. Das Verwalten von strings ist aber so wichtig, dass im C++ Standard ein sogenannte **Klasse** dafür definiert wurde, die Klasse string.
Um einen string zu verwenden können wir string wie einen anderen Datentypen behandeln. Wir müssen einfach die string-Bibliothek mittels #include <string> (using namespace std; danach nicht vergessen !) verfügbar machen.
Hier ein kleines Programm das uns einige Möglichkeiten des strings zeigt.

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    // eine Variable vom Datentyp
    // string definieren
    string einString;

    // einen anderen String
    // erzeugen und initialisieren
    string name = "Xaver";
    // oder so
    string Nachname("Baumann");

    // Einem string einen anderen
    // zuweisen
    string test = name;

    // Eine ganze Zeile von der
    // Konsole in einen String
    // einlesen
    string Eingabe;
    getline(cin, Eingabe);
    // Und Ausgeben
    cout << Eingabe << endl;

    // Strings können auch mit "+"
    // zusammengesetzt werden
    string ganzerName;
    ganzerName = name + " " + Nachname;
    cout << ganzerName << endl;
```

```
    // oder so
    test += " ";
    test += Nachname;
    cout << test << endl;

    // strings können auch verglichen
    // werden !
    if(test == ganzerName)
    {
        cout << "Aha !" << endl;
    }

    // Wir können auch in einen string etwas einfügen !
    ganzerName.insert(6, "M. ");
    cout << ganzerName << endl;

    // oder wieder daraus löschen
    // Von der Stelle 6 an 3 Zeichen löschen
    ganzerName.erase(6, 3);
    cout << ganzerName << endl;

    // Man kann auch auf einzelne Zeichen im string
    // zugreifen
    // Hier greifen wir auf das 5 Zeichen zu, denn
    // die Zahl in der eckigen Klammer gibt
    // den Index des Zeichens an, auf das wir
    // zugreifen wollen. Dabei hat das erste
    // Zeichen im string den index 0
    char einZeichen = ganzerName[4];

    // Länge des strings herausfinden
    int laenge = ganzerName.size();

    return 0;
}
```