

7. Abend

- Wir sind ab heute Abend in der Lage Unterprogramme zu definieren. Solche Unterprogramme werden Funktionen genannt und enthalten Code, den wir von verschiedenen Orten aus aufrufen werden.

Funktionen

Funktionen sind kleine Unterprogramme. Häufig verwendeter Code kann in Funktionen gepackt werden um ein Programm effizienter und übersichtlicher zu gestalten. Ein einfaches Beispiel zeigt uns auch gleich die Anwendung und Definition von Funktionen.

Nehmen wir an, dass wir ein Programm schreiben, dass häufig geometrische Berechnungen anstellt. Wir müssen an verschiedenen Stellen eine Kreisfläche aus einem Radius berechnen.

```
#include <iostream>

using namespace std;

// Um eine Funktion zu verwenden
// müssen wir dem Compiler bekannt
// machen, wie diese Funktion
// heisst und was sie für einen
// Rückgabetypen
double KreisFlaeche(double Radius);
// Eine Funktionsdeklaration beginnt
// mit dem Rückgabetypen, hier : double
// danach der Funktionsname
// hier : KreisFlaeche
// In Klammern stehen dann die
// Parameter die der Funktion übergeben
// werden. Danach ein Semikolon !

// main ist auch eine Funktion !
int main()
{
    char weiter = 'j';

    while('j' == weiter)
    {
        cout << "Bitte Radius eingeben : ";
        double r = 0.0;
        cin >> r;
        // die neue Variable ergebnis
        // erhält den Wert, der in der
        // Funktion KreisFlaeche berechnet
        // wird
        double ergebnis = KreisFlaeche(r);

        cout << endl;
        cout << "Die Kreisflaeche betraegt : ";
        cout << ergebnis << endl;

        cout << "weiter ? ";
```

```

    cin >> weiter;
}

return 0;
}

// Konstante für PI
const double PI = 3.14159;

// Hier wird die Funktion, die wir
// oben deklariert haben nun
// definiert
double KreisFlaeche(double Radius)
{
    double flaeche = Radius * Radius * PI;
    return flaeche;
}

```

Um zu zeigen, wie wir diese Funktion weiterverwenden können machen wir eine zusätzliche Funktion, die eine Ringfläche berechnet.

```

#include <iostream>

using namespace std;

// Funktionsprototypen
double KreisFlaeche(double Radius);
double RingFlaeche(double InnenRadius, double AussenRadius);

int main()
{
    char weiter = 'j';

    while('j' == weiter)
    {
        cout << "[1] fuer Kreis" << endl;
        cout << "[2] fuer Ring" << endl;

        int wahl = 0;
        cin >> wahl;

        double ergebnis = 0.0;

        switch(wahl)
        {
            case 1:
            {
                double r = 0.0;
                cin >> r;

                ergebnis = KreisFlaeche(r);
            }
            break;
            case 2:
            {

```

```

        doubleri = 0.0;
        double ra = 0.0;
        cin >> ri;
        cin >> ra;
        ergebnis = RingFlaeche(ri, ra);
    }
    break;
default:
    break;
}

cout << endl;
cout << "Das Ergebnis betraegt : ";
cout << ergebnis << endl;

cout << "weiter ? ";
cin >> weiter;
}

return 0;
}

// Konstante für PI
const double PI = 3.14159;

// Hier wird die Funktion, die wir
// oben deklariert haben nun
// definiert
double KreisFlaeche(double Radius)
{
    double flaeche = Radius * Radius * PI;
    return flaeche;
}

double RingFlaeche(double InnenRadius,
                    double AussenRadius)
{
    // Ringfleache mit Verwendung der
    // KreisFlaeche-Funktion
    double flaeche = KreisFlaeche(AussenRadius) -
                    KreisFlaeche(InnenRadius);
    return flaeche;
}

```

Parameterübergabe

Eine Funktion kann Parameter erhalten. Da sind Werte, die der Funktion übergeben werden können und in der Funktion zur Verfügung stehen. Ein Parameter hat auch einen Datentypen sowie ein Name, den der Parameter in der Funktion hat. Dieser Name ist nur in der Funktion gültig ! Hier ein kleines Beispiel um dies zu verdeutlichen :

```
#include <iostream>

using namespace std;

// Funktionsprototypen
void AusgabeEinesInt(int einInt);

int main()
{
    int test = 5;

    // Hier wird der Wert
    // der Variable test
    // an die Funktion übergeben
    AusgabeEinesInt(test);

    // Hier wird direkt der
    // Ganzzahlwert 99 an die
    // Funktion übergeben
    AusgabeEinesInt(99);

    return 0;
}

void AusgabeEinesInt(int einInt)
{
    // Hier in der Funktion
    // heisst der Wert der an
    // die Funktion übergeben wurde
    // immer einInt !
    cout << einInt << endl;
}
```

Rückgabewert

Wie wir bereits gesehen haben, kann bei einer Funktion, die eine Berechnung irgendeiner Art anstellt das Ergebnis dieser Berechnung als Rückgabewert verwendet werden. Die allgemeine Form sieht dabei etwa so aus :

```
Variable = Funktion(Parameter);
```

Hat eine Funktion einen Rückgabewert muss der Datentyp der bestimmt werden. So hat im Beispiel oben die Funktion "KreisFlaeche" den Rückgabotyp double. Die Zeile bedeutet also, dass der Variable der Wert zugewiesen wird, den die Funktion zurückgibt. Es gibt jedoch Funktionen, die keinen sinnvollen Rückgabewert haben. Diese Funktionen haben den Rückgabotypen void (Leer). Hier ein Beispiel für eine Funktion, die sinnvollerweise keinen Rückgabotypen hat :

```
void SagHallo()  
{  
    cout << "Hallo" << endl;  
}
```

Diese Funktion hat auch keine Parameter... wären doch alle Funktionen so einfach... In der Funktion selber wird, falls es einen Rückgabewert gibt, dieser mit return zurückgegeben. Dieses return muss nicht einmal am Ende der Funktion stehen, wie wir im folgenden Beispiel sehen :

```
// Diese Funktion findet heraus  
// ob der Benutzer die Funktion  
// noch einmal ausführen will  
bool Wiederholen()  
{  
    cout << "Funktion wiederholen ?" << endl;  
    string antwort;  
    cin >> antwort;  
    if(antwort == "Ja")  
    {  
        // Entscheidung ist  
        // getroffen, also  
        // Rückgabe von true  
        return true;  
    }  
    else  
    {  
        return false;  
    }  
}
```

Sichtbarkeit von Variablen

Variablen, die wir in unserem Code irgendwo deklarieren sind nicht überall sichtbar, das heisst wir können sie nicht von überall her ansprechen. Dies ist insbesondere bei Funktionen wichtig und soll darum an dieser Stelle vertieft werden.

Globale Variablen

Globale Variablen werden ausserhalb jeglicher Funktion definiert und sind darum grundsätzlich in jeder Funktion "sichtbar".

```
int eineGlobaleVariable = 0;

void eineFunktion()
{
    eineGlobaleVariable = 33;
}

int main()
{
    eineGlobaleVariable = 99;

    // Funktion aufrufen
    eineFunktion();

    return 0;
}
```

Die Variable "eineGlobaleVariable" wird ausserhalb von jeglicher Funktion definiert, also hier ausserhalb der Funktion "eineFunktion" sowie ausserhalb der Funktion "main" ! In beiden Funktionen können wir darauf zugreifen. **VERMEIDET GLOBALE VARIABLEN** ! Es ist zum Beispiel in der main-Funktion nicht unbedingt klar, dass der Aufruf der Funktion "eineFunktion" auch gleich die Variable "eineGlobaleVariable" verändert. Solche sogenannten "Seiteneffekte" sollten tunlichst vermieden werden, da sie eine gefährliche Fehlerquelle darstellen.

Im allgemeinen ist aber eine Variable nur innerhalb der umschliessenden geschweiften Klammern sichtbar :

```
{
    int a = 0;
    // ab hier ist die Variable sichtbar
    // ....
}
// ab hier ist die Variable nicht mehr sichtbar !
void Funktion()
{
    int lokaleVariable = 0;
    // hier ist lokaleVariable sichtbar
}
// hier ist lokaleVariable nicht mehr sichtbar!

int main()
{
```

```
int i = 55;

if(i < 100)
{
    int test = 0;
    // test nur innerhalb der
    // if Klammern
}
else
{
    int test = 666;
    // dieses "test" ist nur
    // im else Block sichtbar
}

return 0;
}
```