Kapitel 13

Funktionen

Funktionen

- Eine Funktion ist die Zusammenfassung von mehreren Anweisungen unter einem bestimmten Namen.
- Funktionen verwendet man, um mehrfach benötigte Anweisungsfolgen nur einmal niederzuschreiben und von unterschiedlichen Stellen aufzurufen.
- Mit Funktionen kann ein großes Programm in viele kleine Teile (Unterprogramme) aufgeteilt werden.

Funktionsdefinition

- Eine Funktion besitzt einen Namen und einen Datentyp.
- Eine Funktion kann einen Wert zurückgeben.
- Der Datentyp der Funktion entspricht dem Datentyp des Rückgabewertes.

Funktionsdefinition

```
Typ  Funktionsname(Parameter)
{
     [Definition lokaler Variablen]
     [Anweisungen]
     [return [Ausdruck];]
}
```

Typ: Datentyp des Rückgabewertes
Funktionsname: Name der Funktion
Parameter: Liste der übergebenen Parameter

Funktionsdefinition

- Nach dem Funktionsnamen steht eine Liste von Parametern.
- Über diese Parameter werden Werte mit der Funktion ausgetauscht.
- Jeder dort angegebene Parameter besteht aus einem Variablennamen und dem entsprechenden Datentyp.
- Will man mehrere Parameter übergeben, werden diese durch Komma getrennt.

```
double  Berechnung(int zahl, double radius)
```

return -Anweisung

- Wird bei der Programmausführung auf eine *return* –Anweisung oder auf das Ende des Funktionsblocks getroffen, so wird in die aufrufende Funktion zurückgesprungen.
- Wenn die Funktion nicht vom Typ *void* ist, wird mit der *return* –Anweisung ein Wert zurückgegeben.

```
return [Ausdruck];
```

return -Anweisung

- Der Wert des Ausdrucks entspricht dem Rückgabewert.
- Der Ausdruck wird meist eingeklammert.
- Der Typ der Funktion muss normalerweise dem Typ des Ausdrucks entsprechen.
- Stimmt der Typ der Funktion nicht mit dem Typ des Ausdrucks überein, wird in den Typ der Funktion umgewandelt, falls dies möglich ist.

Datentyp *void*

- Der Datentyp *void* wird für Ausdrücke benötigt, die keinen Wert darstellen.
- Eine Funktion *fname*, die keinen Wert zurückgibt und der kein Wert übergeben wird, wird wie folgt deklariert:

```
void fname(void);
```

Oder auch:

```
void fname();
```

Funktionsdeklaration

- Bevor man eine Funktion aufrufen kann, muss sie vorher definiert und eventuell deklariert sein.
- Eine Funktionsdeklaration sieht wie folgt aus:

```
Typ  Funktionsname(Parameter);
```

- Man beachte das Semikolon am Ende der Zeile.
- Man nennt die Deklaration einer Funktion auch Funktionsprototyp.

Funktionsdeklaration

- Wenn an eine Funktion Parameter übergeben werden, so müssen bei der Deklaration der Funktion mindestens die Datentypen der Parameter angegeben sein.
- Die Parameternamen bei der Deklaration und der Definition einer Funktion müssen nicht zwangsweise übereinstimmen.
- Die Reihenfolge und Anzahl der Datentypen der Parameter bei der Funktionsdeklaration und Funktionsdefinition **muss** übereinstimmen.

Deklaration von Funktionen

- Wenn der Quellcode auf mehrere Dateien verteilt ist, müssen die Schnittstellen von Funktionen und Klassen, die von jeweils anderen Teilen aus benutzt werden, öffentlich bekannt gegeben werden.

```
// Funktionsdeklaration Fehlerausgabe
void FehlerAusgabe(void);
```

Deklaration von Funktionen

- Die Deklarationen werden sehr häufig in Header-Dateien abgelegt.

Definitionsdatei	Test.cpp
Deklarationsdatei (Header-Datei)	Test.h

Definition von Funktionen

- Die Definition von Funktionen wird später noch ausführlich beschrieben.
- In der Definition einer Funktion stehen die Anweisungen der Funktion.

```
// Funktionsdefinition Fehlerausgabe
void FehlerAusgabe(void)
{
        cout << "Es ist ein Fehler aufgetreten!" << endl;
}
```

Funktionsaufruf

- Zum Aufruf einer Funktion schreibt man an die Aufrufstelle den Funktionsnamen gefolgt von der eingeklammerten Parameterliste und einem abschließenden Semikolon.

```
Funktionsname(Parameter);
```

- Die Parameter beim Aufruf einer Funktion werden auch Aktualparameter genannt.

Funktionsparameter

- Es werden zwei Arten der Übergabe von Parametern unterschieden:
  - *call-by-value*
  - *call-by-reference*

call-by-value

- Der Parameter, der übergeben werden soll, wird kopiert und die Kopie wird an die Funktion weitergereicht.
- Der Parameter kann ein Ausdruck, eine Konstante oder eine Variable sein.
- Der Wert des ursprünglichen Parameters bleibt hierbei unverändert.
- Diese Art der Parameterübergabe sollte man auf Parameter beschränken, die nicht viel Speicher (z.B. einfache Datentypen) benötigen.

call-by-reference

- Die aufgerufene Funktion arbeitet statt mit einer Kopie des Parameters mit dem Original.
- Hierzu wird eine Referenz auf den Parameter übergeben.
- Ändert sich der Wert des Parameters nun innerhalb der Funktion, so ändert er sich automatisch auch im aufrufenden Programm

call-by-reference

- Der zu übergebene Parameter muss eine Variable sein.
- Um einen Parameter als Referenz zu kennzeichnen, wird bei der Deklaration und der Definition der Funktion nach dem Datentyp des Parameters der Operator & angefügt.

```
Typ  Funktionsname(Typ  &Parametername);
```

Eindimensionale Arrays als Parameter

- Um ein eindimensionales Array an eine Funktion zu übergeben, wird nur dessen Anfangsadresse übergeben.
  (Ein Array wird daher immer als *call-by-reference* übergeben.)
- Eine Funktion, die ein Array übergeben bekommt, kennt nur diese Anfangsadresse.
- Die Länge des Feldes ist **nicht** implizit bekannt und muss mit übergeben werden.

Mehrdimensionale Arrays als Parameter

- Die Größe des übergebenen Arrays muss, bei der Definition bzw. Deklaration einer Funktion mit einem mehrdimensionalen Array als Parameter, angegeben werden.
- Der Compiler ist damit in der Lage, innerhalb der Funktion die Position der einzelnen Arrayelemente korrekt zu berechnen.