II. Grundlagen C++

II.1.) Das erste Programm

Im folgenden wird der **Ansi-Standard** von C++ verwendet. Unterschiede zu Compilern, die den Ansi-Standard nicht verstehen, werden später aufgezeigt.

In Ansi-C++ sieht das kleinste mögliche Programm folgendermaßen aus:

```
int main ()
{
}
```

Die `main`-Funktion ist die Hauptfunktion. C++ Programme beginnen ihre Ausführung immer mit der `main`-Funktion. Das int vor dem `main` und das leere Klammerpaar () hinter dem `main` sind eine Besonderheit des Ansi-C++, bei älteren Compilern (auch C) ist

```
void main (void)
```

der Form `int main ()` identisch. `void` heißt soviel wie „leer“. Vor dem Funktionsnamen wird deklariert, von welchem **Typ der Rückgabewert** einer Funktion ist.

`void` testfunktion

bedeutet nun, daß die Funktion testfunktion keinen Rückgabewert liefert.

Hinter dem Funktionsnamen stehen in Klammern die **Werte, die der Funktion übergeben** werden. Auch hier bedeutet

```
main(void)
```

daß der Funktion main keine Werte übergeben werden.

Beispiel:

Bei der sinus-Funktion sin(x) wird der Eingabewert x übergeben. Der Typ des Rückgabewertes ist ein reale Zahl.

Da der GNU C/C++ Compiler den Ansi-Standard versteht, wird also weiterhin die Form

```
int main ()
{
}
```

verwendet.

Regel:
Achten Sie beim Tippen auf Groß- und Kleinschreibung, denn der Compiler achtet darauf.

Übung:
Was passiert, falls Sie die `main`-Funktion folgendermaßen schreiben: `Main` ?

Hinter dem Funktionsaufruf `int main ()` des Programmes

```
int main ()
{
}
```

stehen die geschweiften Klammern, die die Anweisungen der Funktion umschließen und nie fehlen dürfen.

Regel:
Geschweifte Klammern fassen Anweisungen zu einem Block zusammen.

Die übliche Schreibweise für C/C++ Programme wird in Ansätzen schon hier deutlich: **Zusammengehörige geschweifte Klammern sollten immer in der gleichen Spalte stehen.**

Das erste sinnvolle Programm

Das obige Minimalprogramm wird zwar vom Compiler richtig übersetzt; da es aber keine Anweisungen beinhaltet, bewirkt es auch nichts. Dies soll nun geändert werden: das folgende Programm soll eine Datenausgabe auf dem Bildschirm vornehmen.

```
#include <iostream>

using namespace std;

int main ()
{
    cout << "Hello World!";
}
```

Betrachten wir zunächst das Hauptprogramm. Die Anweisung `cout` (gesprochen: c out) ist der sogenannte Standard-Ausgabe-Stream. Als Standard-Ausgabe wird im allgemeinen der Bildschirm verwendet. Der <<-Operator schiebt bildlich gesprochen die Daten in den Ausgabestrom.
In diesem Fall sind die Daten die **Stringkonstante** "Hello, World!". Stringkonstanten sind eine Folge von Zeichen, die als Text aufgefaßt werden.

Regel:
Stringkonstanten stehen immer innerhalb doppelter Anführungszeichen.

Hinter der Anweisung `cout` steht ein Semikolon. Allgemein gilt:

Regel:
Anweisungen, denen kein Anweisungsblock folgt, werden mit einem Semikolon abgeschlossen.

Präprozessor

Abgesehen von der `cout` Anweisung ist noch der `#include`-Befehl zum Minimalprogramm dazugekommen. `include` ist kein direkter Bestandteil von C++, sondern ein Befehl des **Präprozessors**. Der Präprozessor ist Teil des Compilers.

Regel:
Präprozessorbefehle erkennt man am # in der ersten Spalte. Präprozessorbefehle dürfen nicht nach rechts eingerückt werden, sondern müssen in der ersten Spalte beginnen.

Ohne `include`-Befehl des iostreams ist dem Compiler die Funktion `cout` unbekannt. In Ansi-C++ muß aber jede Funktion, die aufgerufen werden soll, bekannt sein. Durch die "Einbindung" des iostreams sind dem Compiler typische i/o-Funktionen (input/output-) bekannt. Die Funktionsweise von **Compiler** und **Linker** wird später mit Beispielen behandelt. Hier reicht zunächst, daß der Programmierer dem Compiler explizit sagen muß, daß die Funktion `cout` irgendwo existiert.

Übung:
Kompilieren Sie das obige Programm ohne den Präprozessorbefehl `#include <iostream>`. Was meldet der Compiler?

Namensbereiche

Die letzte Änderung gegenüber dem Minimalprogramm ist die Zeile

```
using namespace std;
```

Übung:
Löschen Sie die Zeile using namespace std; aus dem Programm. Welches Ergebnis liefert

```
#include <iostream>

int main ()
{
    cout << "Hello World!";
}
```

In C++ kann eine Gruppe von Namen (das können Namen von Konstanten, Funktionen, Klassen usw. sein) zu einem Namensbereich (**namespace**) zusammengefaßt werden. `cout` gehört zum Namensbereich `std` (für standard). Alternativ zur Anweisung

```
using namespace std;
```

kann auch

```
std::cout << "Hello, World!";
```

programmiert werden.

Übersetzen des ersten Programmes

Nachdem das Programm

```
#include <iostream>

using namespace std;

int main ()
{
    cout << "Hello World!";
}
```

mit einem Texteditor z.B. unter dem Namen PROG01.C in ein entsprechendes Verzeichnis gespeichert wurde, wird nun mit Hilfe des GNU C/C++ Compilers im MS-DOS Fenster unter Windows 95 das Programm compiliert und gelinkt. Die Begriffe Compiler und Linker werden später erläutert, wenn schon einige Programme erstellt worden sind.

Zunächst werden zwei Möglichkeiten der Übersetzung durch den GNU-Compiler genannt. Der Befehl

gxx PROG01.C [Return]

erzeugt (hoffentlich) eine EXE-Datei namens a.exe. Das Programm wird dann mit a [Return] gestartet. Die andere Möglichkeit erzeugt explizit ein EXE-File mit einem gewünschten Namen (z.B. PROG01.exe), die Befehlsfolge lautet

gxx –o PROG01.EXE PROG01.C [Return]

Das ausführbare Programm wird dann mit prog01 [Return] gestartet.

Vorsicht:
Existieren bereits EXE-Dateien mit den angegebenen Namen wie z.B. a.exe oder prog01.exe in dem Verzeichnis, so werden diese automatisch mit der neuen Version überschrieben, falls der Compilier-Vorgang erfolgreich war.

Übungen:

Probieren Sie verschiedene Varianten des Programmes aus. Verändern Sie die Schreibweise einzelner Programmteile.

Tip: Gewährleisten Sie durch Veränderungen der Stringkonstante, daß Sie auch wirklich jeweils ein neu übersetztes Programm ausführen!

Welche Fehlerquellen stellen Sie fest? Schreiben Sie die Regeln mit Ihren eigenen Worten auf:

ϒ

ϒ

ϒ

ϒ

ϒ

ϒ

ϒ

ϒ

II.2.) Einfache Funktionen, Variablentypen, Ein-/Ausgabe

Das nächste Programm soll als Repräsentant eines "großen" Programmes dienen.

```
#include <iostream>

using namespace std;

int main ()
{
    cout << "Dies ist der 1. Text!";
    cout << "Dies ist der 2. Text!";
}
```

Übung:
Führen Sie das obige Programm aus. Wie sieht die Bildschirmausgabe aus?

Ändern Sie die Ausgabezeilen in

```
cout << "Dies ist der 1. Text!"\n;
cout << "Dies ist der 2. Text!" << endl;
```

und führen das Programm aus. Wie sieht die Bildschirmausgabe jetzt aus?

`endl:` `endl` ist ein sogenannter Manipulator. `endl` im Ausgabestream ruft eine Funktion namens `endl` auf, die das Steuerzeichen `\n` ausgibt und anschließend einen `flush` durchführt.

`\n:` newline. Das Steuerzeichen `\n` im Textstring sorgt für ein Carriage Return, d.h. die Ausgabeposition geht an den Anfang der nächsten Zeile.

`flush:` Normalerweise ist die Ausgabe gepuffert. Es wird erst eine gewisse Menge von Zeichen gesammelt, bis sie auf dem Bildschirm ausgegeben werden. `flush` läßt die Ausgabe sofort durchführen.

`\b:` backspace

`\f:` formfeed

Übung:
Führen Sie das Programm mit den verschieden Steuerzeichen aus.

Die folgende Tabelle listet alle Steuerzeichen auf:

Ganzzahlige Variable

Neben den Stringkonstanten werden zunächst **ganzzahlige Variable** vorgestellt. In C++ gibt es verschiedene Typen von ganzzahligen Variablen, die sich in ihrem Speicherplatzbedarf und damit auch ihrem Zahlenbereich unterscheiden. Die folgenden Bytegrößen sind nach der Ansi-Norm Mindestgrößen, d.h. ein Ansi-Compiler muß auf jeden Fall die angegebene Größe bereitstellen.

Regel:
Das Definieren einer Variablen ist eine Anweisung und wird daher mit einem Semikolon abgeschlossen.

Beispiele, die innerhalb der `main`-Funktion eingebaut werden:

```
int a;
int b;
long c;
long d;

a=4;
b=3;
c=135647;
d=2673287;
```

Variablendefinitionen des gleichen Typs lassen sich zusammenfassen:

```
int a,b;
long c,d;

a=4;
b=3;
c=135647;
d=2673287;
```

oder noch kürzer:

```
int a=4, b=3;
long c=135647, d=2673287;
```

Initialisierung

Bevor eine Variable verwendet wird, muß ihr auf jeden Fall erst einmal ein Wert zugewiesen werden. Das erste Zuweisen eines Wertes an eine Variable heißt **Initialisierung.**

Regel:

Eine nicht initialisierte Variable hat einen nicht vorhersagbaren Wert.

Übung:

Schreiben Sie ein Programm mit den obigen Variablendefinitionen und geben Sie die Werte mit `cout` aus. Was passiert, falls Sie eine Variable nicht initiieren oder z.B. eine unbekannte Variable `e` ausgeben wollen?

Eingabe

Die Eingabe von Werten erfolgt genau wie die Ausgabe über einen sogenannten Stream, den Eingabestream `cin`.

Der Eingabeoperator sieht folgendermaßen aus: `>>`

Geben wir einen Wert ein und weisen ihn der Variablen `a` zu, sieht das so aus:

```
cin >> a;
```

Beispiel:

```
#include <iostream>

using namespace std;

int main ()
{
    int a;

    cout << "Bitte einen Wert eingeben:";
    cin >> a
    cout << "Sie haben " << a << "eingegeben." << endl;
}
```

Die Variable `a` muß hier nicht explizit initialisiert werden. Die Zuweisung des Anwenders kann als Initialisierung betrachtet werden.

Mit `cin` können auch mehrere Werte innerhalb einer Anweisung eingelesen werden:

```
#include <iostream>

using namespace std;

int main ()
{
    int a,b;

    cout << "Bitte zwei Werte eingeben:";
    cin >> a >> b
    cout << "Sie haben " << a << "und ";
    cout << b << " eingegeben." << endl;
}
```

Übung:

Wie können Sie beim obigen Beispiel die Zahlen eingeben?

Ändern Sie das Programm, indem Sie die Eingabe benutzerfreundlicher gestalten.

Überschreiten des Zahlenbereiches

Übung:

Schreiben Sie ein Programm, in dem Sie eine `unsigned int` Variable `a` definieren und dieser als Benutzer einen Wert zuweisen. Dieser Wert soll dann wieder mit `cout` gezeigt werden.

Wie lautet die Bildschirmausgabe bei den Eingabewerten
a.) 10465 ?
b.) 65536 ?
c.) –1 ?

Die "ungewöhnlichen" Ausgabewerte kommen dadurch zustande, daß die Eingabewerte nicht im Zahlenbereich des Variablentyps `unsigned int` liegt. Der Wertebereich ist quasi wie ein Ring. Überschreitet ein Wert den Bereich auf der einen Seite, kommt er auf der anderen wieder heraus. Das gilt auch für die anderen ganzzahligen Variablen.

Falsche Eingaben haben im allgemeinen keine Laufzeitfehler zur Folge. Sie als Programmierer haben in C++ dafür Sorge zu tragen, daß die Eingaben in den von Ihnen gesteckten Grenzen liegen bzw. die Zahlenbereiche richtig gewählt sind.

Übung:

a.) Entwerfen Sie ein Struktogramm für ein Programm, das Sie nach 2 ganzzahligen Zahlen fragt, die Summe davon berechnet, nach einer neuen Zahl fragt, und diese Zahl mit der Summe multipliziert. Das Ergebnis soll ausgegeben werden.
b.) Schreiben Sie das zugehörige Programm dazu.

Formatierte Ausgabe

Neben den bisher bekannten Manipulatoren wie `\n, endl` usw. gibt es Manipulatoren für das Format der Zahlenausgabe. Die Anweisung

```
cout << 3 << 5 << - 8 << 17 << endl;
```

produziert z.B. folgende unschöne Ausgabe: 35-817. Mit dem Befehl `setw` (für setwidth, engl. für "Setze Breite") lassen sich Ausgaben in Tabellenform organisieren.

Übung:
Probieren Sie folgendes Programm aus, beachten Sie dabei den zweiten Präprozessorbefehl `#include <iomanip>`. Wie sieht die Ausgabe aus?

```
#include <iostream>
#include <iomanip>

using namespace std;

int main ()
{
    cout << setw(5) << 3 << setw(5) << 5;
    cout << setw(5) << -8 << setw(5) << 17 << endl;
    cout << setw(5) << 155<< setw(5) << 1;
    cout << setw(5) << 2 << setw(5) << -2 << endl;
}
```

Formatierte Ausgabe

Regel:
`setw` wirkt sich nur auf die unmittelbar folgende Ausgabe aus.

Mit `setfill` können die freien Plätze ausgefüllt werden. Probieren Sie

```
cout << setfill ('*');   oder  cout << setfill ('v');
```

vor den anderen `cout`-Anweisungen.

Die Zahlen wurden bisher rechtsbündig innerhalb des `setw`-Platzes ausgegeben, mit

```
cout << left;            // z.B. anstelle von setfill
```

erfolgt die Ausgabe links. Mit `right` wird die Ausgabe wieder rechtsbündig. Mit

```
cout << showpos;
```

werden die Vorzeichen explizit dargestellt. Mit

```
cout << hex;
```

werden die Zahlen im Hexadezimalsystem ausgegeben.

Übung:
Probieren Sie verschiedene Varianten von den Ausgabe-Manipulatoren aus. Notieren Sie kurz die Ergebnisse.

Die folgende Tabelle listet noch einige weitere Manipulatoren auf:

Variablentyp bool

In C++ wurde als Weiterentwicklung zu C der Variablentyp `bool` eingeführt. In der Boolschen Algebra gibt es nur zwei Werte: `1` und `0`, bzw. `true` und `false`. Nicht nur die Zuweisung von `1` und `0` sind möglich, sondern auch die Zuweisung von `true` und `false`.

```
bool x,y;

x=1;
y=false;
```

Übung:
Schreiben Sie auf Papier (!) den Programmcode für ein Programm, das die Werte von `true` und `false` ausgibt. Dann tauschen Sie Ihr Ergebnis mit Ihren Nachbarn und programmieren deren Code-Vorschlag. Falls Korrekturen notwendig sind, streichen Sie sie bitte als Fehler an.

Übung: Lesen von Programmcode

Übung:
Sie erhalten ein Reihe von Code-Beispielen. Falls Sie Fehler entdecken, korrgieren Sie den Code.

Aufgabe 1

```
#include <iostream>

using namespace;

void main()
{
  Cout << "Hallo!";
}
```

Aufgabe 2

```
int zahl1;

main()
{

  zahl1=19;

  cout << "Die Zahl lautet << zahl1 << endl

}
```

Aufgabe 3

```
#include <iostrea>

using namespce std;

int(main)
{
  cout << "Hello World!";
}
```

Übung: Lesen von Programmcode

Aufgabe 4

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hallo, C++ Programmierer!\m";
}
```

Aufgabe 5

```
#include <iostream>

using namespace std;

int main()
{
  int a,=2 b=4,c,d;
  float x,y,z;
  bool z=2;

  x=4.3, y=5.6;

  c=a+b;
  d=c-b;

  x=2*x;
  z=x/y;

  cin << set(5) << a <<" b= "<< b
  cout << set(3) " c= "<< c <<" d= "<<d << endl;
  cout << x<" y="<<y<<" z="<<z<< endl;
}
```

Kommentare

Kommentare im Programm dienen dazu, Informationen im Programm unterzubringen, die nicht Bestandteil des tatsächlichen Programms sind und auch vom Compiler ignoriert werden.

Kommentare sollen den geschriebenen Programmcode auch nach Monaten oder Jahren des Benutzens sowie auch anderen Programmierern verständlich erklären. Bei größeren Programmen sind dies insbesondere Erklärungen zu Funktionen. Im Laufe des Lehrgangs C++ werden daher die Anforderungen an eine Kommentarstruktur wachsen.

Als Weiterentwicklung von C kennt C++ den einzeiligen Kommentar mit `//`

```
// Beispiel
cout << "Hello, World!";      // Textausgabe
```

Wie in C gibt es auch die Konstruktion mit `/*  */` , die sich über mehrere Zeilen hinziehen kann. Alles zwischen `/*` und `*/` ignoriert der Compiler.

```
/*
Programm       : Beispielprogramm
Autor              : Axel Fenske
Letzte Änderung    : 06.11.1998            */


#include <iostream>      /* Einbinden von iostream*/
```

Bei Präprozessor-Befehlen können manche C++-Compiler Schwierigkeiten mit `//`-Kommentaren haben. In diesem Fall benutzen Sie die "alte" Schreibweise mit `/* */`.

Regel:
Kommentare dürfen nicht verschachtelt werden.

Daher ist `/* Innerhalb eines Kommentares /* ist ein */ Kommentar */` falsch.

Operatoren: +,-,*,/,% und +=,-=,*=,/=,%=

Neben den Grundrechenarten und den zugehörigen Operatoren `+,-,*,/` für Addition, Subtraktion, Multiplikation und Division gibt es noch für die ganzen Zahlen den **Modulo**-Operator `%`.

Regel:
Der Modulo-Operator % bestimmt den Rest einer Division von Ganzzahlen.

Beispiel:
13 % 4 ergibt 1, weil ....

Schon in C wurden Operatoren entworfen, die neben dem Rechenoperator auch die Zuweisung enthalten. So bedeutet

```
zahl += 5;
```

das gleiche wie

```
zahl = zahl + 5;
```

Die folgende Tabelle listet die zusammengesetzten Operatoren auf:

Operatoren: +=,-=,*=,/=,%= und ++,--

Vorsicht:
Auf der rechten Seite kann auch eine Berechnung stehen, diese muß dann vor der Zuweisung ausgeführt werden!

Übung:
Welchen Wert ergibt `wert* = 3 + value;` ?

C++ verfügt wie C noch über spezielle Operatoren zum **Inkrementieren** (`++`) und **Dekrementieren** (`--`) von Variablen.

Definition:
Eine Variable wird inkrementiert, wenn man ihren Wert um 1 erhöht. Eine Variable wird dekrementiert, wenn man ihren Wert um 1 vermindert.

Typischerweise werden Zähler um 1 erhöht oder um 1 vermindert. Beim typischen Addieren oder Subtrahieren sollten weiterhin die Operatoren `+, -, +=, -=` benutzt werden.

```
wert = 3;
wert ++;
wert ++;
wert --;       //  Wie groß ist Wert jetzt?
```

Operatoren: ++,--

Übung:
Entwerfen Sie ein Struktogramm für ein Programm, das Sie eine ganze Zahl über die Tastatur eingeben läßt. Diese Zahl soll auf drei verschiedene Arten um 1 erhöht und dann um 1 vermindert werden. Jedesmal soll die Zahl auf dem Bildschirm ausgegeben werden.

Der ++-Operator und der −−-Operator wurden bisher als **Postinkrement**-Operator und als **Postdekrement**-Operator vorgestellt. Daneben gibt es noch das **Präinkrement** und das **Prädekrement**.

Definition:
Bei den Postoperatoren wird die Variable erst benutzt und dann inkrementiert oder dekrementiert. Die Präoperatoren inkremtieren bzw. dekrementieren vor der Benutzung.

Operatoren: ++,--

Übung:

Schreiben Sie folgendes Programm:

```
#include <iostream>

using namespace std;

int main ()
{
    int stack = 3, erg;

    erg = stack;
    cout << erg << "," << stack << endl;
    erg = stack++;
    cout << erg << "," << stack << endl;
    erg = stack--;
    cout << erg << "," << stack << endl;
}
```

Wie lautet die Ausgabe? Wandeln Sie die Post-In- und Post-Dekrement-Operatoren um in Prä-Operatoren. Wie lautet jetzt die Ausgabe?

Übungen

Aufgabe 1
Welchen ganzzahligen Wert ergibt `17/3` ?

Aufgabe 2
Welchen ganzzahligen Wert ergibt `17%3` ?

Aufgabe 3
Welchen Unterschied gibt es zwischen den Ausdrücken `3*4+5` und `5+4*3` ?

Aufgabe 4
Welchen Unterschied gibt es zwischen den Ausdrücken `(3*4)+5` und `5+(4*3)` ?

Aufgabe 5
Ist `x+=2` und `x++2` das gleiche ?

Aufgabe 6
Worin liegt der Vorteil von `++` gegenüber `+=1` ?

Aufgabe 7
Welcher Wert wird ausgegeben ?

```
a=3;
a+=5;
--a;
a=a+a;
cout << a << endl;
```

Aufgabe 8

Welcher Wert wird ausgegeben ?

```
a=3;
a+=5+a++;
--a;a++;
a=a+a;
cout << a << endl;
```

Aufgabe 9

Welcher Wert wird ausgegeben ? (schwer)

```
a=12;
a+=++a+a++;
a=a+a;
cout << a << endl;
```

Aufgabe 10

Versuchen Sie die folgenden Zuweisungen so umzuschreiben, daß nur jeweils einer der Operatoren `+=, -=, *=, /=` verwendet wird.

```
1)  x=x+1;
2)  a=a-8;
3)  c=c-3;
4)  s=r*s*t;
5)  a=4*b+a;
6)  a=a*4+b;
7)  a=a*(4+b);
8)  c=c-3; c=c-6;
9)  d=d*5; d=d*e;
10) h++; i=3*h+i;
11) a=a+3; b=b+a;
12) x=x*y; y=y+1;
```

II.3 Funktionen

Eine Funktionsdeklaration sieht folgendermaßen aus:

RÜCKGABETYP funktion (FUNKTIONSPARAMETER)

Wenn kein Funktionswert zurückgegeben werden soll, und auch kein Funktionsparameter übergeben wird, sind beide Typen `void`.

```
void funktion (void)
```

Als Beispiel soll eine frühere Funktion, die zwei Bildschirmausgaben bewerkstelligt, in zwei Funktionen zerlegt werden, die dann von der `main`-Funktion aufgerufen werden. (Vom alten Sprachgebrauch Unterprogramme ist möglichst Abstand zu nehmen.)
Da die zwei Funktionen nur eine `cout`-Anweisung bearbeiten sollen, sind Rückgabewert und Funktionsparameter `void`.

```
#include <iostream>

using namespace std;

void ausgabe1(void)
{
  cout << "Dies ist der 1. Text!" << endl;
}

void ausgabe2(void)
{
  cout << "Dies ist der 2. Text!" << endl;
}

int main()
{
  ausgabe1();
  ausgabe2();
}
```

Regel:
Funktionsaufrufe besitzen immer ein Paar runde Klammern hinter dem Funktionsnamen.

Diese Regel gilt auch dann, wenn an die Funktion wie in unserem Beispiel keine Parameter übergeben werden. Der Compiler erkennt an den Klammern, daß es sich um einen Funktionsaufruf handelt.

Übung:
Erstellen Sie jeweils ein Struktogramm für das ursprüngliche Programm sowie für das Hauptprogramm mit den zwei Funktionsaufrufen.

Funktionsnamen

Funktionsnamen müssen mit einem Buchstaben oder Unterstrich beginnen und dürfen weiterhin nur Buchstaben, Ziffern oder Unterstriche enthalten.

Beginnen Sie trotzdem einen Funktionsnamen nicht mit einem Unterstrich.

Regeln (für alle Namen):

- Bei Namen wird auf Groß- und Kleinschreibung geachtet.
- Namen dürfen nicht identisch mit C++-Schlüsselwörtern sein.
- Zur Namensunterscheidung werden i.a. nur die ersten 31 Zeichen herangezogen.

Übung:

Vertauschen Sie die Reihenfolge der Anweisungen: Stellen Sie die `main`-Funktion vor die beiden Funktionen `ausgabe1` und `ausgabe2`. Wie lautet die Ausgabe?

Prototypen

Die Fehlermeldungen des Übungs-Programmes kommen dadurch zustande, daß die Funktionen `ausgabe1` und `ausgabe2` dem Compiler an dieser Stelle (im `main`-Programm) noch gar nicht vorgestellt wurden. Ähnlich ist das Problem bei `cout` ohne `#include <iostream>`.

Die zweite Art der Fehlermeldung kommt daher, daß der Compiler nicht erkennen kann, welchen Rückgabetyp die Funktionen `ausgabe` haben. Der Compiler nimmt nun den Typ `int` an. Statt dessen liefern die Funktionen `ausgabe` gar keinen Rückgabewert, so daß dies für den Compiler eine Redeklaration (Umdeklaration) ist und einen Fehler meldet.

Mit **Prototypen** können wir dem Compiler vor den Funktionsaufrufen sagen, wie die beiden Funktionen aussehen werden.

Die Prototypen sehen in diesem Beispiel wie folgt aus:

```
     #include <iostream>

     using namespace std;

     void ausgabe1(void);          // Prototyp
     void ausgabe2(void);          // Prototyp

     int main()
     {
       ausgabe1();
       ausgabe2();
     }

void ausgabe1(void)
{
  cout << "Dies ist der 1. Text!" << endl;
}

void ausgabe2(void)
{
  cout << "Dies ist der 2. Text!" << endl;
}
```

Module

Bei großen Programmen (oft mit mehreren tausend Codezeilen) treten folgende Probleme auf:

- das Editieren und Kompilieren einer Quelldatei dauert lange,
- große Quelldateien sind unübersichtlich, schwieriger zu verstehen und zu pflegen,
- das Arbeiten im Team wird erschwert.

Die Lösung dieser Probleme liegt darin, ein größeres Programm in **Module** aufzuteilen (Modulare Programmierung). An einem Beispiel wollen wir gleichzeitig getrenntes Kompilieren und Linken üben.

Trennen wir unseres gemeinsames Ausgabeprogramm in ein "Dienstleistungsmodul" x.c

```
#include <iostream>           // Name: X.C

using namespace std;

void ausgabe1(void)
{
  cout << "Dies ist der 1. Text!" << endl;
}

void ausgabe2(void)
{
  cout << "Dies ist der 2. Text!" << endl;
}
```

und in ein Hauptprogramm y.c

```
int main()                //Name: Y.C
{
  ausgabe1();
  ausgabe2();
}
```

Module

Mit den Befehlen

```
gxx –c x.c
gxx –c y.c
```

oder dem Befehl

```
gxx –c x.c y.c
```

werden die Dateien compiliert. Die daraus .o-Dateien sind nun kompiliert, aber noch nicht **gelinkt** (zusammengefügt). Mit

```
gxx –o x.exe x.o y.o
```

kann nun eine x.exe erzeugt werden. Wird nun z.B. y.c geändert, braucht nur diese Datei mit gxx –c y.c kompiliert werden und mit gxx –o x.exe x.o y.o gelinkt werden.

Übung:
Probieren Sie obiges Verfahren und erzeugen Sie getrennte Module.

Wieder stoßen wir auf das Problem der nicht deklarierten Funktionen `ausgabe1` und `ausgabe2`, das wir vorhin mit Prototypen gelöst haben.

Übung:
Deklarieren Sie die Prototypen für die beiden Funktionen. (In welcher Datei ?) .Läuft nun das Programm ?

Header-Dateien

Das Deklarieren der Prototypen vor der `main`-Funktion ist bei großen Dateien umständlich und hat außerdem den Nachteil, daß sie auch nur dieser einen Funktion zur Verfügung stehen.

Um die Prototypen der beiden Funktionen `ausgabe1` und `ausgabe2` unabhängig von ihrer Verwendung zu deklarieren, benutzen wir eine **Header-Datei**.

Wir nennen unsere Header-Datei x.h

```
// Name: X.H

void ausgabe1(void);
void ausgabe2(void);
```

Nun muß diese Header-Datei noch der `main`-Funktion bekannt gemacht werden:

```
#include "x.h"

int main()
{
  ausgabe1();
  ausgabe2();
}
```

Ihre eigene Header-Datei wird mit Anführungsstrichen eingebunden. Bei Anführungszeichen sucht der Compiler die Header-Datei im aktuellen Arbeitsverzeichnis. Bei spitzen Klammern sucht der Compiler die Header-Datei im Standardverzeichnis des Compilers.

Nun müßten das getrennte Kompilieren und das Linken wie oben angegeben funktionieren.

Übung

Aufgabe

Ein Programm soll entwickelt werden, bei dem der Anwender über die Tastatur zwei Zahlen eingeben soll. Die Summe soll dann berechnet werden. Ist die Summe größer gleich zwanzig, so soll folgender Text mit Hilfe eines Unterprogramms herausgegeben werden: "Die Summe ist größer gleich 20!". Ist die Summe kleiner als zwanzig, soll mit einem anderen Unterprogramm ausgegeben werden: "Die Summe ist kleiner 20!".

Die zwei Unterprogramme sollen in einzelne Dateien abgelegt werden. Im Hauptprogramm sollen die Prototypen nicht explizit definiert werden, sondern über eine entsprechende Header-Datei eingebunden werden.

a.) Entwickeln Sie ein Struktogramm für das Programm.
b.) Legen Sie die Dateinamen fest.
c.) Schreiben Sie das Hauptprogramm auf Papier auf.
d.) Schreiben Sie die Unterprogramme sowie die Header-Datei auf Papier auf.
e.) Schreiben Sie die notwendigen Compiler-Befehle auf Papier auf.
f.) Geben Sie Ihren gesamten Code Ihrer Tauschgruppe und korrigieren Sie deren Code.
g.) Programmieren Sie dann Ihren eigenen korrigierten Code.

Benutzen Sie eine bisher unbekannte C++-Anweisung:

```
if (zahl < 10)
     {
          Anweisungen          // falls Bedingung true
     }
   else
     {
          Anweisungen          // falls Bedingung false
     }
```

Bezugsrahmen von Variablen

Variablen haben eine

- **Lebensdauer** und eine
- **Sichtbarkeit,**

die entscheidend davon abhängen, an welcher Stelle sie definiert werden.

Definition:
Den Bereich eines Programms, in dem eine bestimmte Variable existiert, nennt man **Bezugsrahmen** der Variablen.

Bezugsrahmen = engl. **Scope**

Bisher haben wir nur mit einer **Speicherklasse** von Variablen gearbeitet: den **lokalen Variablen.**

Beispiel:

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

void aendern(void)
{
     x=10;
}

int main()
{
     int x=4;
     cout << "x= " << x << endl;
     aendern();
     cout << "x= " << x << endl;
}
```

Bezugsrahmen von Variablen

Regel:
Eine Variable, die innerhalb eines Anweisungsblocks definiert wurde, ist lokal und gilt nur innerhalb dieses Blocks.

Übung:
Ändern wir daher die Funktion `aendern`:

```
void aendern(void)
{
    int x;
    x=10;
    cout << "x= " << x << endl;
}
```

Wie lautet die Ausgabe?

Wichtig !!!

Regel:
Eine Variable, die innerhalb eines Anweisungsblocks definiert wurde, ist lokal und gilt nur innerhalb dieses Blocks.

Bezugsrahmen von Variablen

Regel:
Gibt es mehrere Variablen mit gleichem Namen, spricht man über den Namen immer die lokalste der Variablen an.

Globale Variablen

Neben den lokalen Variablen gibt es die **globalen Variablen**, die von mehreren Funktionen benutzt werden können.

Beispiel:

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

int x;

void aendern(void)
{
     x=10;
}

int main()
{
     x=4;
     cout << "x= " << x << endl;
     aendern();
     cout << "x= " << x << endl;
}
```

Wie lautet die Ausgabe?

Bezugsrahmen von Variablen

Regel:
Variablen, die außerhalb einer Funktion definiert werden, sind global und können in jeder Funktion benutzt werden.

Lokale und globale Variablen können auch zusammen benutzt werden:

Übung:
Welche Werte gibt das folgende Programm aus? Erklären Sie den Programmablauf.

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

int x;

void aendern(void)
{
     x=10;
}

int main()
{
     int x=4;
     cout << "x= " << x << endl;
     aendern();
     cout << "x= " << x << endl;
}
```

Übung

Welche Bildschirmausgaben produziert folgendes Programm? Spielen Sie das Programm in Gedanken durch und diskutieren Sie ihr Ergebnis mit Ihrem Nachbarn und Tauschpartner. Erst dann probieren Sie es per Compiler.

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

int a,b;

void aendern(void)
{
     int a;
     a=0;
     if (a==0)
          {
               int a=20;
               cout << "a= " << a << endl;
          }
     cout << "a= " << a << endl;
}

int main()
{
     a=b=10;
     cout << "a= " << a << " und b= " << b << endl;
     aendern();
     cout << "a= " << a << " und b= " << b << endl;
}
```

Bezugsrahmen von Variablen

Bezugsrahmenoperator :: = engl. Scope-Operator

Beispiel:

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

int x=4;

int main()
{
     int x=10;
     cout << "x= " << x << endl;
     cout << "x= " << ::x << endl;
}
```

Grundsatz:
So lokal wie möglich, so global wie nötig.

Globale Variablen sind eine große Fehlerquelle. Sie passen nicht in das Konzept der modularen Programmierung sowie der Datenkapselung bei der objektorientierten Programmierung.

Statische Variablen

Regel:
Statische Variablen werden beim Verlassen ihres Gültigkeitswertes nicht gelöscht, sondern behalten ihren Wert bei.

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

void count(void)
{
     static int a=1;
     cout << "a= " << a << endl;
     a++;
}

int main()
{
     count();
     count();
     count();
}
```

Notwendig:
Statische Variablen müssen bei ihrer Definition initialisiert werden.

Übung:
Was ändert sich, falls im obigen Beispiel die Variable `a` getrennt erst deklariert wird und dann in einer extra Anweisung initialisiert wird ?

Funktionsparameter

Einer Funktion können Werte übergeben werden, auch ohne globale Variablen zu benutzen. Dies geschieht über die **Funktionsparameter.**

Beispiel:

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

void quadrat(int x)
{
     int quad;
     quad=x*x;
     cout << "Das Quadrat ist " << quad << endl;
}

int main()
{
     int wert;
     cout << "Bitte geben Sie eine Zahl ein:";
     cin >> wert;
     quadrat(wert);
}
```

Funktionsparameter sind lokale Variablen der Funktion, die mit den Übergabeparametern initialisiert sind.

Funktionsparameter verhalten sich wie lokale Variable. Man kann ihre Werte innerhalb der Funktion verändern, und zwar ohne Auswirkungen auf die Original-Variablen, die im Funktionsaufruf stehen.

Funktionsparameter

Mehrere Parameter können auch übergeben werden.

Regel:
Funktionsparameter werden in der Parameterliste durch Kommata voneinander getrennt. Dabei muß jeder Parameter eine eigene Typangabe besitzen.

Übung:
Schreiben Sie ein Haupt-Programm, über das Sie nacheinander 2 ganzzahlige Zahlenwerte eingeben. In einem Unterprogramm `summe` soll die Summe berechnet und ausgegeben werden.
Entwickeln Sie Ihre Lösung mit Hilfe eines Struktogramms.

Wertrückgabe

Beispiel:

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

int quadrat(int x)
{
     int quadrat;
     quadrat=x*x;
     return(quadrat);
}

int main()
{
     int wert,ergebnis;
     cout << "Bitte geben Sie eine Zahl ein:";
     cin >> wert;
     ergebnis=quadrat(wert);
     cout<<"Quadrat von "<<wert<<"ist "<<ergebnis<<endl;
}
```

Regeln:

Die `return`-Anweisung beendet die Funktion und gibt den Wert zurück, der hinter `return` angegeben wurde.

Funktionen, die einen Wert zurückgeben, repräsentieren diesen Wert mit ihrem Aufruf.

Wird der Rückgabewert einer Funktion nicht angegeben, geht der Compiler davon aus, daß ein `int`-Wert zurückgegeben wird.

Prototypen von Funktionen mit Funktionsparametern

Steht im vorigen Beispiel die `quadrat`-Funktion hinter der `main`-Funktion, ist wieder ein Prototyp notwendig.

Der Prototyp für `quadrat` sieht folgendermaßen aus:

```
int quadrat(int);
```

Regel:
Prototypen enthalten den Rückgabetyp und die Variablentypen der Funktionsparameter, nicht jedoch die Variablennamen.

Aufgabe

Schreiben Sie ein Programm, bei dem Länge und Breite eines Raumes abgefragt werden sollen. Die Fläche soll in einem Unterprogramm `flaeche` berechnet werden, wobei das Resultat dem Hauptprogramm übergeben werden soll. Die Hauptfunktion soll dann das Ergebnis auf dem Bildschirm ausgeben.

Übung

Aufgabe

Ein Programm soll entwickelt werden, bei dem der Anwender über die Tastatur zwei Zahlen eingeben soll. Mit Hilfe einer Funktion `summe` soll die Summe berechnet werden und ans Hauptprogramm zurückgegeben werden. Ist die Summe größer gleich zwanzig, so soll mit der Funktion `quadrat` das Quadrat der Summe berechnet werden und ans Hauptprogramm zurückgegeben werden. Ist die Summe kleiner zwanzig, soll mit der Funktion `quadro` das Quadrat des Quadrates der Summe berechnet werden und ans Hauptprogramm zurückgegeben werden. Das zurückgegebene Ergebnis soll im Hauptprogramm ausgegeben werden.

Die Unterprogramme sollen in einzelne Dateien abgelegt werden. Im Hauptprogramm sollen die Prototypen nicht explizit definiert werden, sondern über eine entsprechende Header-Datei eingebunden werden.

Teil A: ca. 45 Minuten
Bearbeitung allein; Unterlagen können benutzt werden; Ordentlichkeit und Übersichtlichkeit werden mitbenotet; wer fertig ist, geht in die Pause.

a.) Entwickeln Sie ein Struktogramm für das Programm. Ist es möglich, daß die Funktion quadro die Funktion quadrat benutzt? Wenn ja, soll dies ins Struktogramm eingebaut werden.
b.) Legen Sie die Dateinamen fest.
c.) Schreiben Sie das Hauptprogramm auf Papier auf.
d.) Schreiben Sie die Unterprogramme sowie die Header-Datei auf Papier auf.
e.) Schreiben Sie die notwendigen Compiler-Befehle auf Papier auf.

Teil B:
f.) Geben Sie Ihren gesamten Code Ihrem Tauschpartner aus einer anderen Gruppe und programmieren Sie **exakt (!!!)** dessen Code und präsentieren dessen Ergebnis.
g.) Das Ergebnis, das Sie präsentieren, wird benotet und Ihrem Tauschpartner gutgeschrieben. (Läuft das Programm nicht, wird der Code bewertet.) Halten Sie sich bei der Code-Eingabe und Namensvergabe nicht exakt an seine Vorgaben, wird dies Ihnen als Minuspunkte angekreidet.

return

Regel:
Eine Funktion mit Rückgabewert muß so entworfen sein, daß sie zu **jeder** Bedingung mit einem `return` beendet wird.

Beispiel:

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

int quadrat(int x)
{
     int quadrat;
     if (x<10)
          {
          quadrat=x*x;
          return(quadrat);
          }
}

int main()
{
     int wert,ergebnis;
     cout << "Bitte geben Sie eine Zahl ein:";
     cin >> wert;
     ergebnis=quadrat(wert);
     cout<<"Quadrat von "<<wert<<"ist "<<ergebnis<<endl;
}
```

Läuft das Programm? Können wir das beheben? Probieren Sie verschiedene `returns` hinter der `if`-Bedingung aus.

Überladen von Funktionen

Im Gegensatz zu C und vielen anderen Programmiersprachen können Sie in C++ Funktionsnamen überladen.

Definition:
Ein Funktionsname ist dann überladen, wenn mehrere Funktionen diesen Namen besitzen.

Z.B. haben Sie eine Funktion, die die Summe zweier Zahlen ausgibt und `summe` heißt. Falls Sie eine Funktion benötigen, die die Summe dreier Zahlen ausgibt, können sie diese natürlich `summe3` nennen. Dies wird aber schnell unübersichtlich. So kann es sinnvoll sein, daß verschiedene Funktionen denselben Namen haben.

Damit der Compiler trotzdem die Funktionen unterscheiden kann, muß gelten:

Funktionen mit gleichem Namen und gleichem Bezugsrahmen müssen eine unterschiedliche Parameterliste besitzen.

Beispiel:
1.) Schreiben Sie ein Programm, bei dem sie 3 Zahlenwerte eingeben, und diese an zwei Funktionen mit dem Namen `summe` übergeben. Bei der einen Funktion werden 2 Summanden, bei der anderen alle 3 Summanden benutzt. Die beiden Ergebnisse sollen zurückgegeben und auf dem Bildschirm ausgegeben werden.

2.) Ist folgende Deklaration richtig ?

```
int summe(int, int);
long summe(int, int);
```

Default-Werte von Funktionsparametern

Funktionsparameter können mit sogenannten **default-**Werten belegt werden. Für den Fall, daß einem Parameter beim Funktionsaufruf kein Wert zugewiesen wird, wird dann einfach der voreingestellte Wert verwendet.

default = engl.

Voriges Beispiel der Summenfunktion:

```
int summe(int a, int b, int c=)
{
     return(a+b+c);
}
```

Der Aufruf `summe(5,7);` gibt den Wert `12` zurück. Warum?

Was gibt `summe(3);` zurück?

Es können auch mehrere Funktionsparameter default-Werte besitzen:

```
int summe(int a, int b=0, int c=0)
{
     return(a+b+c);
}
```

Es gilt aber:
Nur die Funktionsparameter am Ende der Parameterliste dürfen default-Werte besitzen.

Probieren Sie

```
int summe(int a, int b=0, int c)
{
     return(a+b+c);
}
```

Übung: Lesen von Programmcode

Sie erhalten eine Reihe von Code-Beispielen, die teilweise nur Teile eines Programms sind. Falls Sie Fehler entdecken, korrigieren Sie den Code.

Aufgabe 1:

```
void change (float x, float y)
{
     float a=1;
     a=x*1.16
     a=change return(a);
}

int main ()
{
     cin >> zahl;
     zahl=change(zahl);
     cout << zahl;
}

void change (float);
```

Aufgabe 2

```
double test (void);
int main ()
{
     cin >> zahl;
     quad=test(zahl)
     cout quad;
}
double test (int x)
{
     return (x*x)
}
```

Aufgabe 3

```
#iclude <iostream>;
int quadat(x)
{
     int quadrat;
     if (x<10)
          quadrat=(x*x);
          x=return(quadrat)
          }
}
int main
     int, double wert,ergebnis;
     cout >> "Bitte geben Sie eine Zahl ein:";
     cin << wert
     quadrat(wert);
     cout>>"Quadrat von ">>wert>>"ist ">>ergebnis>>endl;
```

Aufgabe 4

```
void flaeche(laenge, breite);
void volumen(laenge, breite, hoehe);

int mian ()
{
     float a,b,c;
     "Geben Sie Länge, Breite und Höhe des Raumes an :";
     cin >> a,b,c;
     a=flaeche(b,c);
     b=volumen(a,b,c);
     cout << " Fläche ist "<<a<<endl<<"Volumen ist"<<b;
}
void flaeche(int x, int y)
{
     a=x*y;
     return(a)
}
void volumen(int x, int y, int z)
{    return (x+y+z);
}
```