Skript zur vhb-Vorlesung

Programmierung in C++
Teil 1

Prof. Dr. Herbert Fischer
Technische Hochschule Deggendorf

Inhaltsverzeichnis

Primär-Literatur:

Herrmann Dietmar
Grundkurs C++ in Beispielen, vieweg Verlag, 6.Auflage, 2010
ISBN: 3-8266-0910-7

Louis Dirk
C++, Hanser Verlag, 1. Auflage, 2014
ISBN: 3-446-44069-2

Kirch-Prinz Ulla, Kirch Peter
C++ Lernen und professionell anwenden, mitp-Verlag, 5.Auflage, 2010
ISBN: 3-89842-171-6

Arnold Willemer
Einstieg in C++, Galileo Computing, 4.Auflage, 2009
ISBN: 3-8362-1385-0

Sekundär-Literatur:

Helmut Balzert
Lehrbuch der Softwaretechnik, Spektrum Akademischer Verlag, 2.Auflage, 2000
ISBN: 3-8274-0480-0

Peter P. Bothner
Ohne C zu C++, Vieweg, 1.Auflage, 2001
ISBN: 3-528-05780-7

Ernst-Erich Doberkat
Das siebte Buch: Objektorientierung mit C++, B.G. Teubner Stuttgart · Leipzig · Wiesbaden, 2000
ISBN: 3-519-02649-X

John R. Hubbard
C++- Programmierung, mitp, 1.Auflage, 2003
ISBN: 3-8266-0910-7

Dietrich May
Grundkurs Softwareentwicklung mit C++, vieweg, 2.Auflage, 2006
ISBN: 3-8348-0125-9

Tools:

Code::Blocks für Windows, Linux, Mac OS (kostenlose Software): **http://codeblocks.org/downloads/26**

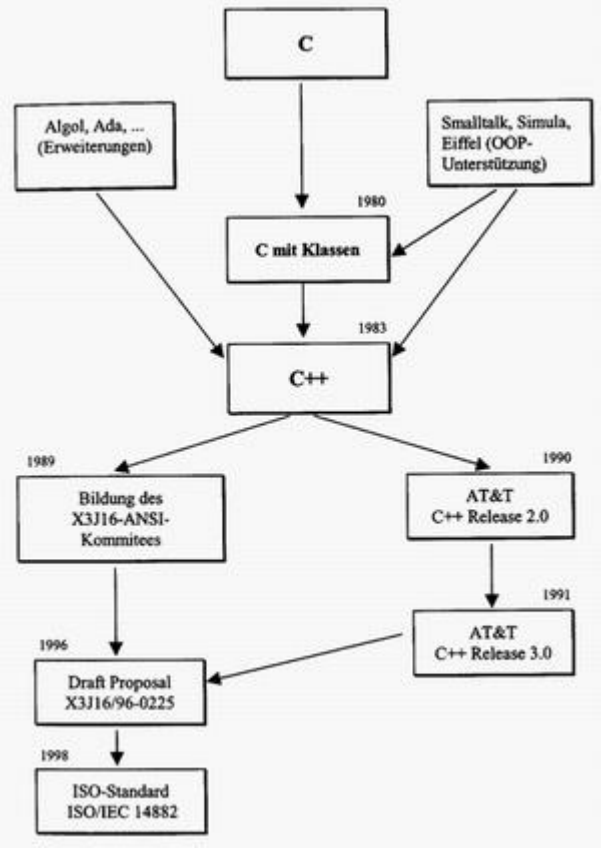
Alternativen:
CodeLite für Windows, Linux, Mac OS: **http://downloads.codelite.org/**
KDevelop für Windows, Linux: **https://www.kdevelop.org/download**
Dev-C++ für Windows: **http://www.bloodshed.net/dev/**
XCode für Mac OS: **https://itunes.apple.com/de/app/xcode/id497799835**

1 Einführung in die objektorientierte Programmierung: C++

In Kapitel 1 erhalten Sie eine kurze Einführung in die C++-Programmierung und werden Ihr erstes C++-Programm erstellen.

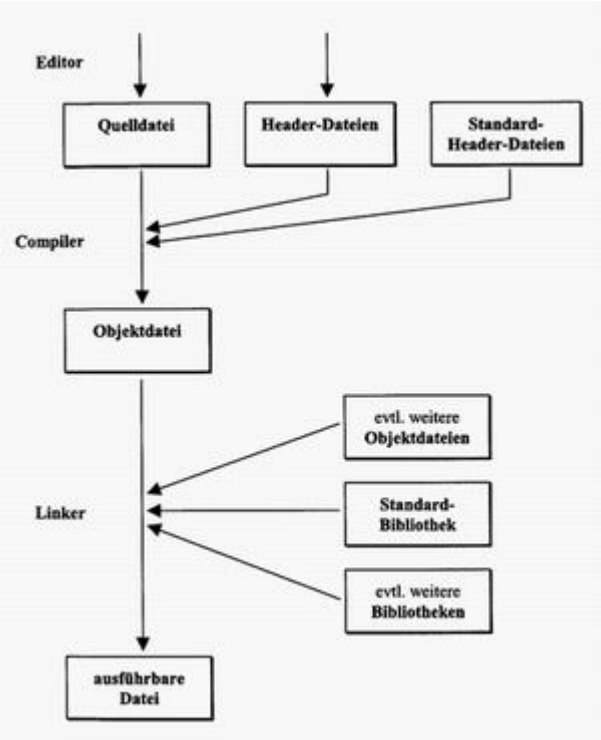
Die Programmiersprache C bildete die Basis für C++, eine Programmiersprache, die häufig auch als »C mit Klassen« bezeichnet wird. In heutigen C++-Programmen ist die Verwandtschaft zu C noch deutlich zu erkennen. C++ wurde nicht geschrieben, um C zu ersetzen, sondern um sie zu verbessern.

1.1 Entwicklung von C++



C++ wurde von Bjarne Stroustrup in den Bell-Laboratorien (Murray Hill, USA) entwickelt, um Simulations-projekte mit minimalem Speicherplatz und Zeitbedarf zu realisieren. Frühe Versionen der Sprache, die zunächst als »C mit Klassen« bezeichnet wurde, gibt es seit 1980. Der Name C++ wurde 1983 von Rick Mascitti geprägt. Er weist darauf hin, dass die Programmiersprache C++ evolutionär aus der Programmiersprache C entstanden ist: ++ ist der Inkrementoperator von C.
C wurde wegen ihrer Effizienz und Portabilität als Grundlage von C++ gewählt. Bei der Weiterentwicklung von C++ wurde stets auf die Kompatibilität zu C geachtet. Somit bleibt die umfangreiche, unter C entwickelte Software auch in C++-Programmen einsatzfähig. Dazu gehören beispielsweise Tools und Bibliotheken für Grafik-Systeme oder Datenbankanwendungen.
Bei der Realisierung objektorientierter Konzepte hatte die Programmiersprache SIMULA67 maßgeblichen Einfluss, insbesondere bei der Bildung von Klassen, der Vererbung und dem Entwurf virtueller Funktionen. Das Überladen von Operatoren und die Möglichkeit, Deklarationen im Programmtext frei platzieren zu können, wurde der Programmiersprache ALGOL68 entlehnt. Die Programmiersprachen Ada und Clu haben die Entwicklung von Templates und die Ausnahmebehandlung beeinflusst.
Schließlich gehen viele Entwicklungen aus den Jahren 1987 bis 1991 auf die direkten Erfahrungen und Probleme von C++-Programmierern zurück. Hierzu gehören beispielsweise die Mehrfachvererbung, das Konzept der rein virtuellen Funktionen und die Nutzung gemeinsamer Speicherbereiche für Objekte.

1.1.1 Der Weg zum ausführbaren C++-Programm



Zur Erstellung und Übersetzung eines C++-Programms sind grundsätzlich die gleichen Schritte wie in C notwendig:

- Das Programm wird mit einem Editor erstellt.
- Das Programm wird kompiliert, d.h. in die Maschinensprache des Rechners übersetzt.
- Der Linker erzeugt schließlich die ausführbare Datei.

1.1.2 Editor

Mit einem Editor werden die Textdateien erstellt, die den C++-Code enthalten. Dabei sind zwei Arten von Dateien zu unterscheiden:

- Quelldateien

Quelldateien, auch Source-Dateien genannt, enthalten die Definitionen von globalen Variablen und Funktionen. Jedes C++-Programm besteht aus mindestens einer Quelldatei.

- Header-Dateien

Header-Dateien, auch Include-Dateien genannt, verwalten zentral die Informationen, die in verschiedenen Quelldateien gebraucht werden.
Dazu gehören:

- Typdefinitionen, z. B. Klassendefinitionen
- Deklarationen von globalen Variablen und Funktionen
- Definition von Makros und Inline-Funktionen

Bei der Benennung der Dateien muss die richtige Endung (engl. Extension) verwendet werden. Diese variieren jedoch von Compiler zu Compiler: Für Quelldateien sind die gebräuchlichsten Endungen .cpp und .cc. Die Namen von Header-Dateien enden entweder wie in C mit .h oder sie haben keine Endung. Aber auch Endungen wie .hpp können vorkommen. Die Header-Dateien der C-Standard-Bibliothek können natürlich weiter benutzt werden.

1.1.3 Compiler

Eine Übersetzungseinheit besteht aus einer Quelldatei und den inkludierten Header-Dateien. Der Compiler erzeugt aus jeder Übersetzungseinheit eine Objektdatei (auch Modul genannt), die den Maschinen-Code enthält. Neben den Compilern, die direkt den Maschinen-Code erzeugen, gibt es auch C++- nach C-Übersetzungsprogramme, sogenannte »C-Front-Compiler«. Diese übersetzen ein C++-Programm in ein C-Programm. Erst anschließend wird die Objektdatei mit einem Standard-C-Compiler erzeugt.

1.1.4 Linker

Der Linker bindet die Objektdateien zu einer ausführbaren Datei. Diese enthält neben den selbsterzeugten Objektdateien auch den Startup-Code und die Module mit den verwendeten Funktionen und Klassen der Standardbibliothek.

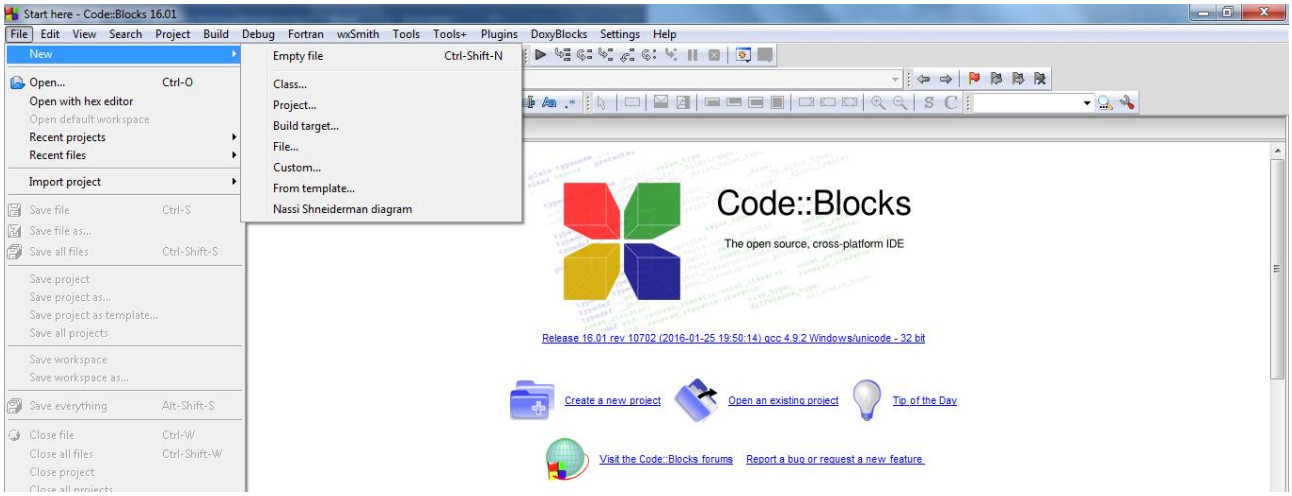
1.2 Einführung in die Programmierumgebung: C++

Nach dieser kurzen Einführung wollen wir nun gleich unser erstes C++-Programm realisieren. Wir werden in diesem Kurs Win32-Konsolenanwendung erstellen. Das sind Programme, die in einem DOS-Fenster laufen. Als Entwicklungsumgebung verwenden wir Code::Blocks. Sie können jedoch analog auch jedes andere C++-Entwicklungstool nutzen (z.B. Dev-C++, CodeLite, XCodes, MS Visual Studio, usw.).
Code::Blocks können Sie unter **http://codeblocks.org/downloads/26** herunterladen.
Ein Videotutorial zu Code::Blocks finden Sie auf der Kurshomepage.

Erstellen einer Konsolenanwendung mit Code::Blocks

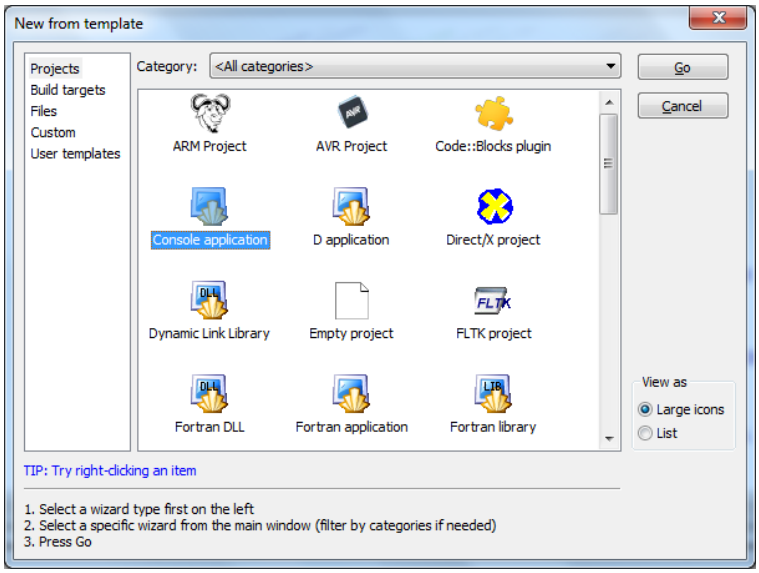
Schritt 1

Starten Sie die Entwicklungsumgebung Code::Blocks. Wählen Sie unter dem Menüpunkt *File* zuerst *New* und dann *Project* oder klicken Sie direkt im Infobereich auf *Create a new project*.



Schritt 2

In dem folgenden Dialogfeld wählen Sie unter der Registerkarte das Icon *Console* application. Klicken Sie dann auf *Go*. Wählen Sie als Sprache C++ aus und vergeben Sie einen von Ihnen frei wählbaren Projekt-Titel und sowie einen Speicherort. Zuletzt prüfen Sie, ob der Compiler *Gnu GCC Compiler*, sowie die beiden Häkchen bei *Debug* und *Release* gesetzt wurden. Klicken Sie zuletzt auf *Finish*.



Schritt 3

Klicken Sie links bei der Navigationsleiste auf *Sources* und machen Sie einen Doppelklick auf *main.cpp*. Dies ist unser Hauptprogramm. Code::Blocks hat schon ein Codegerüst erzeugt, das wir verwenden können:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[ ])
{
        cout << "Hello world!" << endl;
        system("pause");
        return 0;
}
```

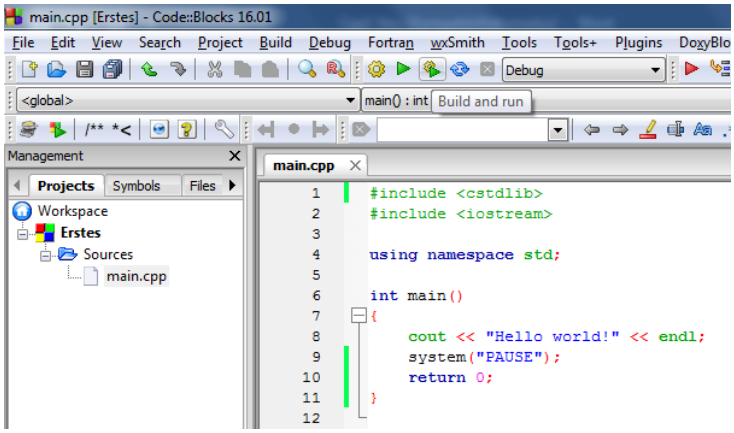
Durch Klicken auf das entsprechende Symbol erzeugt der Compiler ein lauffähiges Programm. D.h. in unserem Fall öffnet sich ein DOS-Fenster, welches die Ausgabe „Hello world!“ erzeugt.

Es gibt folgende Symbole:

Build:	Kompiliert das Programm und erstellt eine ausführbare Anwendung.
Run:	Führt das zuletzt kompilierte Programm aus.
Build & Run:	Kompiliert das Programm und führt es anschließend aus.

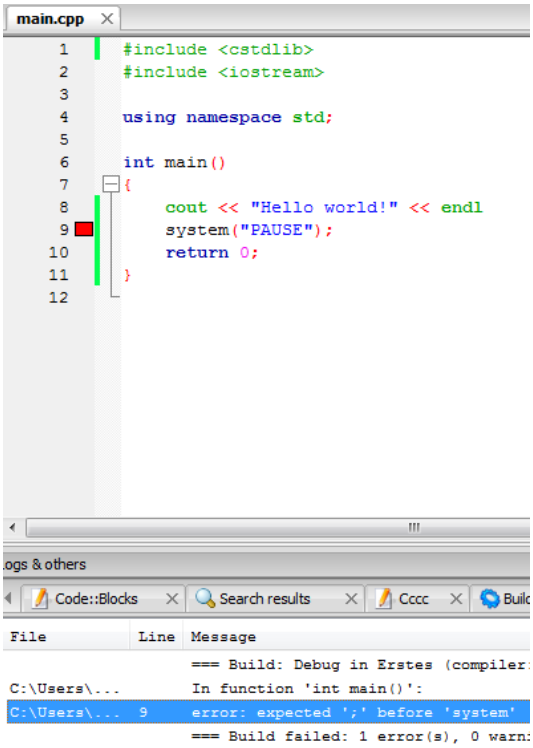
Schritt 4

Klicken Sie auf „Build & Run“, um das Programm zu kompilieren und auszuführen.



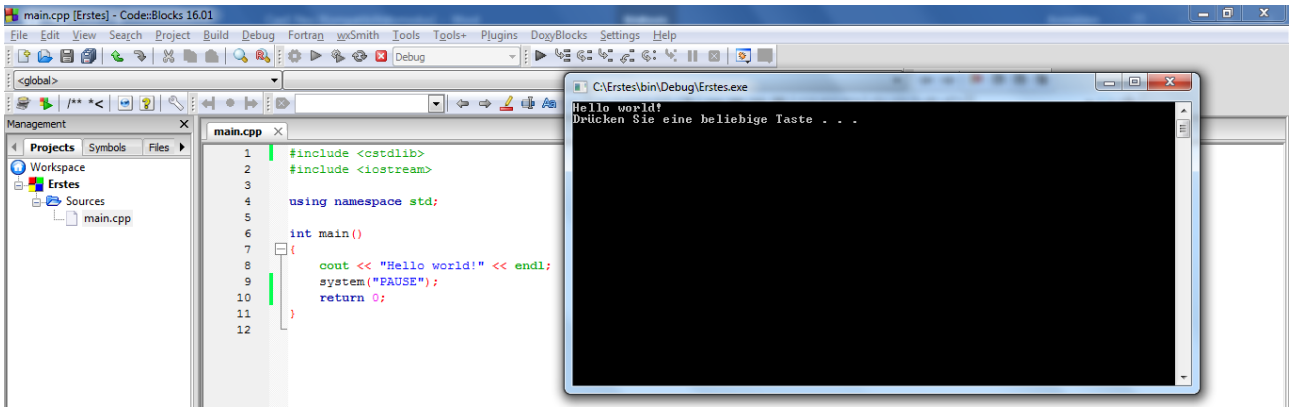
Schritt 5

Wenn der Compiler einen Fehler entdeckt, markiert er die fehlerhafte Zeile links mit einem roten Rechteck und gibt im unteren Fenster eine Meldung aus. Korrigieren Sie den Fehler (hier der fehlende Strichpunkt) und kompilieren Sie nochmal. Speichern Sie die Datei.



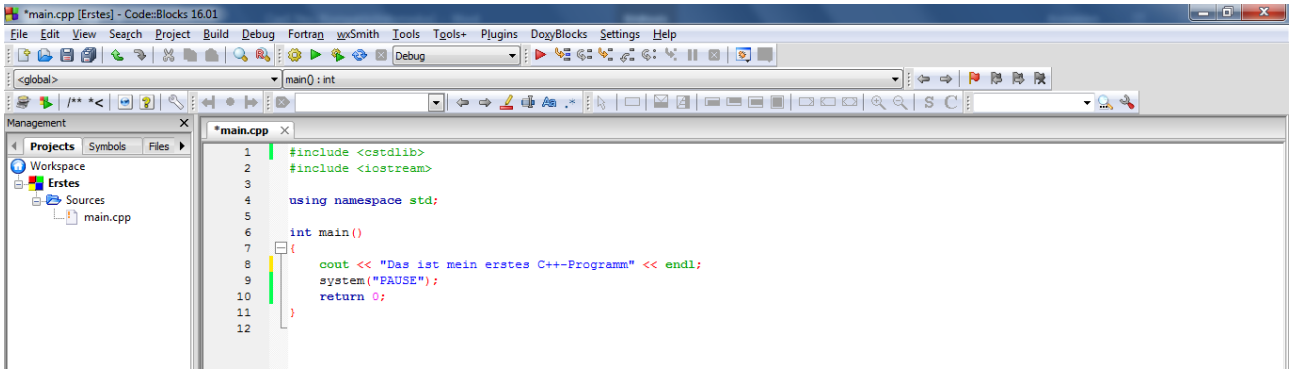
Schritt 6

Nach klicken auf „Run“ wird das Programms in einem DOS-Fenster ausgeführt.



Schritt 7

Speichern Sie (*Save All files* ) und schließen Sie das Programm.



Nun haben Sie Ihr wahrscheinlich erstes C++-Programm mit Erfolg realisiert.

Meinen Glückwunsch!

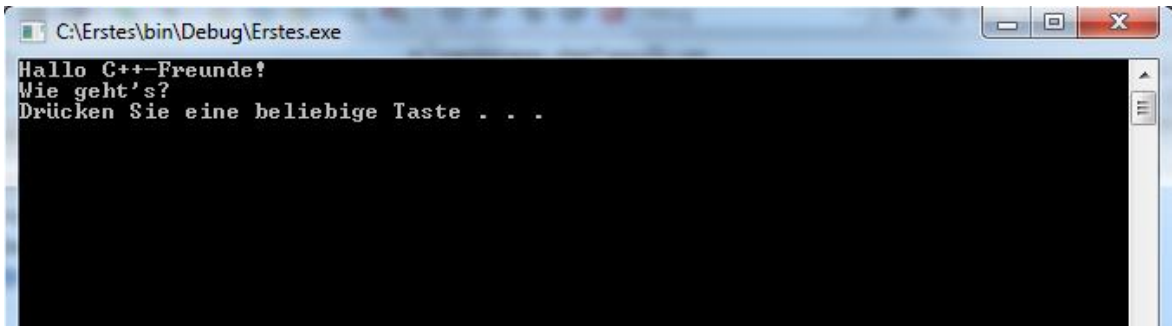
1.2.1 Einfache Ausgabe am Bildschirm

Betrachten wir nochmals unser Programm. Fügen Sie nun folgende Zeilen ein:

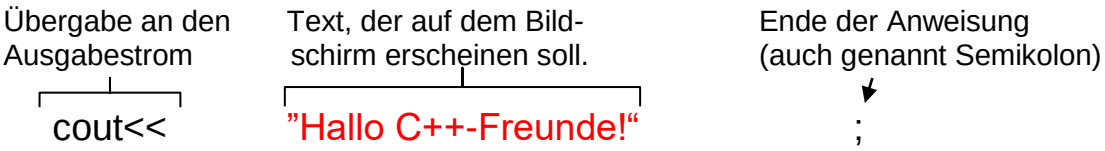
```
cout << "Hallo C++-Freunde!";
cout << endl;
cout << "Wie geht's?" << endl;
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[ ])
{
        cout << "Hallo C++-Freunde!";
        cout << endl;
        cout << "Wie geht's?" << endl;

        system("pause");
        return 0;
}
```

Wenn Sie nun auf „Build & Run“ klicken, erscheint folgende Ausgabe im Konsolenfenster:



Um Text auf dem Bildschirm anzeigen zu können, benötigen wir eine C++-Klasse namens *iostream.* Eine kurze Beschreibung dieser Klasse wäre also angebracht. Sie wissen zwar noch nicht, was Klassen sind, doch sollte Sie das nicht bekümmern. Die Klasse iostream verwendet *streams* (Ströme), um grundlegende Ein-/Ausgabeoperationen durchzuführen – beispielsweise die Ausgabe von Text auf dem Bildschirm oder das Einlesen der Benutzereingabe.



Über den cout-Strom werden Daten an den Standardausgabestrom gesendet. Für Konsolenanwendungen ist dies die Konsole oder der Bildschirm. Die Klasse iostream verwendet spezielle Operatoren, um Informationen in einen Strom zu schreiben. Der *Übergabeoperator* (<<) wird verwendet, um Daten in einen Ausgabestrom zu leiten. Um also Informationen auf der Konsole auszugeben, würden Sie folgendes eingeben: cout << "Tue etwas!";
Damit teilen Sie dem Programm mit, den Text "Tue etwas!" in den Standardausgabestrom einzufügen. Achten Sie dabei darauf, dass der Text in Anführungszeichen steht und die Codezeile mit einem Strichpunkt endet. Wenn die Programmzeile ausgeführt wird, erscheint der Text auf Ihrem Bildschirm.

Anmerkung: cout wird nur in Konsolenanwendungen eingesetzt.

1.2.3 Header-Dateien

Bevor Sie cout einsetzen können, müssen Sie dem Compiler mitteilen, wo die Beschreibung *(Deklaration* genannt) der *iostream*-Klasse steht, in der cout zu finden ist. Die Klasse iostream ist in der Datei IOSTREAM deklariert. Diese Datei wird auch *Header-Datei* genannt.
Um dem Compiler mitzuteilen, dass er in IOSTREAM nach der Klassendeklaration von iostream suchen muss, benutzen Sie die *#include*-Direktive: Durch die #I*nclude*-Direktive werden häufig verwendete Funktionen in den Quellcode eingebunden und so im Programm nutzbar gemacht.

#include <iostream>

Wenn Sie vergessen haben, für eine Klasse oder Funktion die dazugehörige Header-Datei in Ihr Programm mit aufzunehmen, ernten Sie einen Compiler-Fehler. Die Fehlermeldung des Compiler könnte beispielsweise lauten: *Undefined symbol 'cout'*
Wenn Sie diese Meldung erhalten, sollten Sie umgehend überprüfen, ob Sie alle für Ihr Programm benötigten Header mit aufgenommen haben.

using namespace std;

Gibt den Namensraum an, in dem alle Bibliothekselemente in C++ deklariert sind.

Anmerkung:
Die erste Zeile *#include <cstdlib>* wird nur verwendet um System-Anweisungen wie system("pause"); und system("cls“); auszuführen und kann entfallen, wenn man diese Anweisungen nicht nutzt.

1.2.4 Bibliotheken

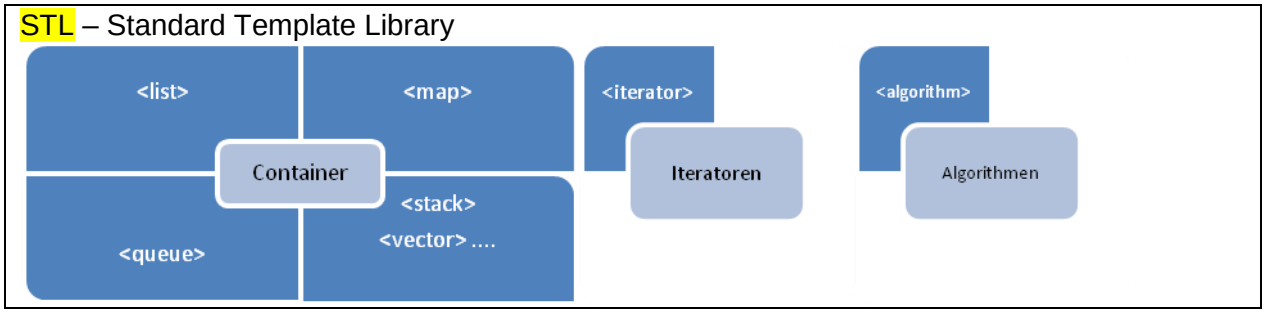
In C/C++ gibt es Bibliotheken mit verschiedenen Klassen und Funktionen. Die Funktionsbibliotheken entstammen teilweise der Programmiersprache C. Der große Vorteil von C und C++ ist die Standardisierung der Bibliotheken. Die C++-Standardbibliothek stellt folgende Komponenten als erweiterbares Rahmenwerk zur Verfügung: Strings, Container, Algorithmen, komplexe Zahlen, Ein-/Ausgabe und vieles mehr. Alle Datentypen, Klassen und Funktionen sind in der Namespace *std* enthalten.

Bereits beim Linken werden die für das Programm benötigten Funktionen aus den Bibliotheksdateien dazu gebunden. Die Verwendung von Bibliotheken erfolgt in zwei Schritten. Zunächst muss die Header-Datei mit Hilfe des Präprozessor-Kommandos *#include* in die Quelltextdatei eingebunden werden. Im zweiten Schritt dagegen werden dem Linker die eigentlichen Bibliotheksdateien bekannt gemacht, z.B. <*string*>.

Beispiele für die wichtigsten Bibliotheken in C++:

<iostream>	Ein-/Ausgabe
<string>	Zeichenketten
<cstdlib>	Hilfsfunktionen
<cmath>	Mathematische Funktionen
<ctime>	Datum und Uhrzeit
<random>	Zufallszahlen

Eine Liste mit weiteren Bibliotheken ist auf folgender Webseite zu finden:
http://www.cplusplus.com/reference/



STL ist die Abkürzung für **S**tandard **T**emplate **L**ibrary. STL ist eine Sammlung von Template-Klassen und stellt einen Container zu Verfügung. Dieser ist in der Lage gleichartige Daten zu organisieren. Die einfachste Form eines Containers ist ein Array (Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.**). Für diese Container stellt die STL einige Funktionalitäten zur Verfügung, wie z.B. Such- und Sortierfunktionen oder Einfüge- und Löschoperationen.

1.2.5 endl;

Die iostream-Klasse enthält spezielle Manipulatoren, mit denen die Behandlung der Ströme gesteuert werden kann. Der einzige Manipulator, mit dem wir uns im Moment befassen wollen, ist *endl;* (end line), der dazu benutzt wird, eine neue Zeile in den Ausgabestrom einzufügen. Wir verwenden *endl,* um eine neue Zeile einzufügen, nachdem wir den Text auf dem Bildschirm ausgegeben haben. Bitte beachten Sie, dass das letzte Zeichen von *endl* ein l und keine 1 ist.
endl kann an den Schluss eine cout-Anweisung angehängt werden, oder mit cout in einer gesonderten Zeile stehen. Man kann auch mehrere *endl* hintereinanderschreiben, um mehrere Leerzeilen auszugeben. Alternativ kann man in den String aber auch \n schreiben, um eine neue Zeile zu erstellen.

```
        cout << ”Hallo C++-Freunde!“ << endl;

oder:   cout << ”Hallo C++-Freunde!“;
        cout << endl;
oder:   cout<< ”Hallo C++-Freunde!\n“;
```

1.2.6 main-Funktion

Die *main*–Funktion (Hauptprogramm) ist der Einstiegspunkt für das Programm. Nach Abarbeitung sämtlicher in ihr enthaltenen Anweisungen, die in den geschweiften Klammern enthalten sind, ist das Programm beendet.

Das Programmgerüst in Code::Blocks lautet:

```
        int main(int argc, char *argv[ ])
```

Es genügt jedoch auch:

```
        int main()
```

1.2.7 Klammern und Whitespace-Zeichen

Auffällig sind auch die geschweiften Klammern im Programm. In C++ beginnt ein Codeblock mit einer öffnenden { und endet mit einer schließenden geschweiften Klammer }. Diese Klammern dienen dazu, den Beginn und das Ende der Codeblöcke von Schleifen, Funktionen, if-Anweisungen etc. zu markieren. In unserem Programm gibt es nur einen Satz Klammern, da es sich um ein sehr einfaches Programm handelt.

In C++ werden *Whitespace*-Zeichen einfach ignoriert. Meistens ist es völlig unerheblich, wo Sie Leerzeichen oder neue Zeilen einfügen. Natürlich können Sie Leerzeichen nicht innerhalb von Schlüsselwörtern oder Variablennamen verwenden, aber ansonsten sind Sie nicht gebunden.

So sind die folgenden Quelltexte beispielsweise vollkommen äquivalent:

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

1.2.8 Kommentare

```
#include <iostream>
#include <cstdlib>

using namespace std;       // Das ist ein Kommentar

int main()
{
        cout << ”Hallo C++-Freunde!“;   // Das ist ein weiterer Kommentar
        system(“pause”);
}
```

Nach den Zeichen // können Sie einzeilige Kommentare in Ihren Quelltext eingeben. Kommentarzeilen dienen dazu, Ihr Programm zu dokumentieren.

Mehrzeilige Kommentare kann man auch folgendermaßen schreiben:

```
/*   Kommentar
        .....
*/
```

1.2.9 system("pause");

Die C-Bibliothek stellt uns die Funktion *system("pause");* zur Verfügung, die auf eine Eingabe über die Tastatur wartet. Mit der Tastatureingabe wird das Konsolenfenster geschlossen. Diese Funktion ist compilerspezifisch. Beim Visual C++-Compiler kann *system("pause");* einfach weggelassen werden. Ebenso kann Return EXIT_SUCCESS; entfallen oder durch return 0; ersetzt werden.

Aufgabe: Schreiben Sie ein Programm, das Ihren Namen und Ihre Anschrift wie folgt auf dem Bildschirm ausgibt:

Moritz Mustermann
Am Stadtplatz 1

94469 Deggendorf

Das Fenster soll nach einem Tastendruck geschlossen werden. Kommentieren Sie jede Zeile.

Lösung:

```
#include <iostream>                  // Einbinden der Headerdatei iostream
#include <cstdlib>                   // Einbinden der Headerdatei cstdlib
using namespace std;                 // Namensraum

int main()                           // Hauptprogramm
{                                    // Programmanfang
   cout<<"Moritz Mustermann"<<endl;        // Textausgabe auf dem Bildschirm mit Zeilenumbruch
   cout<<"Am Stadtplatz 1"<<endl;          // Textausgabe auf dem Bildschirm mit Zeilenumbruch
   cout<<endl;                             // Leerzeile
   cout<<"94469 Deggendorf"<<endl;         // Textausgabe auf dem Bildschirm mit Zeilenumbruch

system("pause");                     // Schließen des Fensters erst nach Tastatureingabe
}                                    // Programmende
```

1.2.10 system("cls");

Mit dem Befehl *system("cls");* kann die komplette Bildschirmausgabe gelöscht werden. Der Cursor steht anschließend oben links am Anfang der Befehlszeile.