



Aachen Programming Club

C ++ Blitzeinführung

© GNU/FDL* 2002 Martin Henne

Version 23.11.2002

Inhalte:

- Grundlagen
- Hello, world!
- Standardausgabe
- Standardeingabe
- grundlegende Datentypen

* Ausnahme zur GNU/FDL: Der Quellcode dieses Dokuments darf nicht in die Formate *.doc, *.pps, *.xls oder andere Formate des Microsoft Office Paketes konvertiert werden. Für die Richtigkeit und/oder Vollständigkeit dieses Dokumentes wird gemäß der GNU/FDL keinerlei Haftung übernommen.

Dieses Dokument wächst ständig und wird immer wieder verbessert und erweitert. Die neueste Version ist unter <http://www.martinhenne.de/referenzkarten/> zu finden. Bei entdeckten Fehlern oder sonstigen Anmerkungen wird um eine EMail an den Autor gebeten.

Die Referenzkarten und die Blitzeinführung sind zur Begleitung einer C++ Schulung für Anfänger und fortgeschrittene C/C++-Programmierer und als Nachschlagewerk, jedoch nicht als Unterlagen zum selbständigen Erlernen der Inhalte gedacht. Ohne den Kontext der Schulung können einzelne Inhalte evtl. schwer verstanden werden. Von Flames im Bezug auf unvollständige Darstellungen bitte ich daher abzusehen ;-)

Grundlagen

Die **Grundlegende Vorgehensweise** beim Programmieren in C++ ist für alle Betriebssysteme gleich und lässt sich grob in zwei Bereiche unterteilen:

1. Das Erstellen des Programm-Quellcodes ("**Programmieren**")
2. Erzeugen eines lauffähigen Programmes aus dem Quellcode ("**Compilieren**")

Dabei lassen sich beide Bereiche in mehrere Schritte unterteilen, die jedoch, je nach der zum Programmieren verwendeten Software, automatisch erfolgen.

Zum Erstellen des Quellcodes genügt ein einfacher Texteditor, der das ASCII-Format unterstützt. Vereinfacht ausgedrückt, ist der Quellcode nichts anderes als ein Textdokument, das Anweisungen der Programmiersprache C++ enthält. Unter Windows kommt dafür als Notlösung sogar das sogenannte 'Notepad' in Frage. Unix-Betriebssysteme verfügen meist standardmäßig über leistungsfähige Editoren, die einen Programmierer in vielerlei Hinsicht unterstützen (z.B. vi oder emacs).

Zur Umwandlung des Quellcodes in ein lauffähiges Programm (Unter Windows bekannt als *.exe-Datei) wird ein sogenannter "Compiler" benutzt. Ein Compiler im weiteren Sinne ist eine Software, die in mehreren Schritten aus ASCII - Programmcode (Quellcode) ein lauffähiges Programm erzeugen kann. Einer der leistungsfähigsten Compiler überhaupt, der GCC ("GNU Compiler Collection") ist für eine große Anzahl von Betriebssystemen und Hardwareplattformen frei erhältlich. Darunter auch Linux, Windows, MacOS, Solaris, PalmOS und viele mehr.

Häufig trifft man auf sogenannte **IDE's (Integrated Development Environment - integrierte Entwicklungsumgebung)**, das sind Programme, die den Editor, den Compiler und weitere Werkzeuge, die das Programmieren ermöglichen bzw. erleichtern, in einer einheitlichen Softwareumgebung zusammenfassen.

Unter Windows ist die kommerzielle Microsoft IDE *Visual C++* stark verbreitet. Da diese jedoch viele Schritte automatisiert, die zur Folge haben, dass sich die so programmierte Software nur noch unter Windows verwenden lässt, ist diese Entwicklungsumgebung für Einsteiger nicht zu empfehlen. Es besteht die Gefahr, unbewusst leistungsfähige und portable Standardkomponenten der Sprache C++ durch proprietäre Erweiterungen zu ersetzen, die unter Linux, Macintosh oder anderen Betriebssystemen nicht lauffähig sind.

Auch anspruchsvolle grafische Anwendungen können unter Verwendung portabler Komponenten (Bibliotheken) für mehrere Betriebssysteme gleichzeitig erstellt werden.

Programmieren setzt grundlegendes Verständnis für Computer und das verwendete Betriebssystem voraus. Selbst wenn es möglich ist, einige Dinge auch ohne PC-Grundwissen zu vermitteln, so macht es keinen Sinn, Programme für ein System schreiben zu wollen, das man nicht versteht.

Umgekehrt wird jedoch die Programmierfähigkeit in der Regel das grundlegende Verständnis für Computer und deren Eigenheiten erheblich erweitern.

Das erste Programm, das üblicherweise erstellt wird, wenn man eine neue Programmiersprache erlernt, dient nur dem Zweck, die Zeichenkette „Hello, world!“ auf dem Bildschirm auszugeben.

Die Idee dabei ist, die minimal notwendigen Komponenten kennenzulernen, die für ein lauffähiges Programm unentbehrlich sind. Ausserdem werden alle erforderlichen Schritte anhand der verwendeten Programmierumgebung durchlaufen.

C++ Quellcodedateien müssen über eine **Suffix** verfügen, die signalisiert, dass diese ASCII-Datei C++ Quellcode enthält. Die meisten Compiler verstehen folgende Datei-Endungen:

`*.cc *.cpp *.cxx *.C`

Unter Windows kann die letzte Variante – das grosse C – nicht verwendet werden, da dieses Betriebssystem bei Dateinamen nicht zwischen Gross- und Kleinschreibung unterscheidet. Ein kleines c als Dateiendung ist die gebräuchliche Suffix für C-Programme ('C' ist eine ältere Programmiersprache, auf der C++ basiert).

Es wird empfohlen, **grundsätzlich** die Endung ***.cpp** zu verwenden. Ausserdem wird ausdrücklich empfohlen, aus Gründen der **Konsistenz** alle Dateinamen in Kleinbuchstaben zu schreiben. Die Dateinamen sollten keine Leer- oder Sonderzeichen bzw. Umlaute enthalten. Wenn man eine solche Festlegung für sich selbst **konsequent** durchhält, schliesst man dadurch eine mögliche Fehlerquelle bei der Arbeit mit mehreren Quellcode-Dateien von Anfang an aus. Die meisten Betriebssysteme können zwar mit solchen Dateinamen in jedem Fall umgehen, häufig stellt jedoch deren Verwendung für den ungeübten Benutzer ein Problem dar. Es handelt sich also dabei weder um eine Einschränkung von C++, noch von Unix oder Windows, sondern um eine Maßnahme zum „Selbstschutz“ :-)

Genug der Vorrede – jetzt wird programmiert.

Die folgenden Zeilen sollen in einem ASCII-Editor eingegeben und in einer Datei mit dem Namen „**helloworld.cpp**“ abgespeichert werden. (!! Bitte nicht „Hello, World.cpp“ oder „hello world.cpp“ usw. - keine Leer- oder Sonderzeichen s.o. !!) :

Die Standardausgabe (cout)

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
}
```

Erläuterung:

Das o.a. Mini-Programm gibt die Zeichenkette "Hello, world!" auf dem Bildschirm aus. Dabei bedeuten die einzelnen Zeilen folgendes:

#include <iostream> ist eine sogenannte Präprozessoranweisung. Diese spezielle Include-Anweisung stellt die Funktionalität der Headerdatei "**iostream**" zur Verfügung.

Diese Funktionalität beinhaltet unter anderem Ein- und Ausgabeströme, darunter auch ein Ausgabestrom zum Bildschirm bzw. der sogenannten Standardausgabe. Der Standardausgabestrom gibt Daten auf der Konsole (auch: Eingabeaufforderung, Shell, usw.) aus. Er heisst '**cout**' (console output) und befindet sich im Namensraum '**std**'. "**iostream**" wird immer eingebunden, wenn **Ausgaben auf dem Bildschirm** oder **Eingaben von der Tastatur** benötigt werden!

Durch "iostream" steht ebenfalls die Zeilenschaltung 'endl' (end of line) zur Verfügung, die bewirkt, dass die darauf folgenden Ausgaben am Beginn der nächsten Zeile erscheinen. Wie alle im C++-Standard festgelegten Funktionen und Elemente liegt auch '**endl**' im Namensraum '**std**'.

Hinweis:

Der Begriff "Namensraum" wird später genauer erläutert. Für den Moment reicht es aus, zu wissen, dass man auf Elemente, die sich in einem Namensraum befinden, nur zugreifen kann, wenn man den Namensraum direkt qualifiziert (wie im Beispiel durch das Schreiben von 'std::' vor das Element) oder den Namensraum vorher ganz oder teilweise öffnet (wird noch behandelt).

Den sogenannten Funktionskopf (auch "Header" oder "Signatur") der Hauptfunktion stellt die Zeile `int main()` dar.

Die `main()`-Funktion (Hauptfunktion) ist die einzige Funktion, die in jedem C++ - Programm vorhanden sein muß. Es ist allerdings eher üblich, dass neben dieser Hauptfunktion eine Vielzahl von Funktionen - meist einige Hunderte - zum Einsatz kommen.

Funktionen werden später erklärt. Im Augenblick genügt es, zu wissen, daß jedes C++-Programm eine sog. **main()-Funktion** braucht und daß die darin befindlichen Anweisungen (hier: `std::cout << "Hello, world!" << std::endl;`) in geschweiften Klammern eingeschlossen werden.

Die einzige Anweisung in unserer **main()-Funktion** gibt die Zeichenkette **"Hello, world!"** auf der Standardausgabe (**std::cout**, meist der Bildschirm) gefolgt von einer Zeilenschaltung (**std::endl**) aus. Dabei wird der sogenannte **Ausgabeoperator (<<)** verwendet.

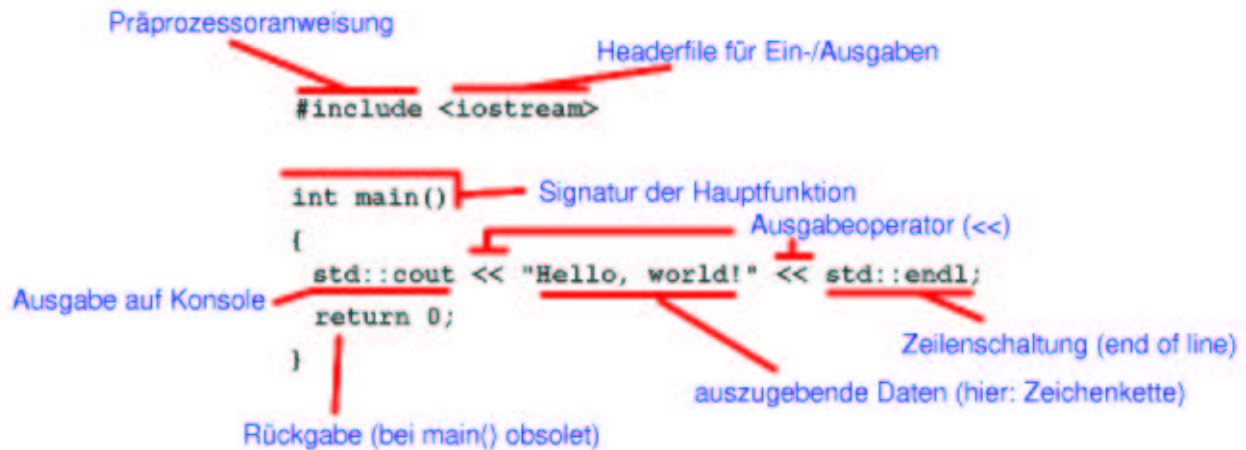
Hinweis:

Jede Anweisung endet mit einem Semikolon!

Ein C++-Quellcode sollte **Kommentare** enthalten. Für das richtige und maßvolle **Kommentieren** gibt es einige Richtlinien, die erst in einem fortgeschrittenen Stadium eine Rolle spielen. Während der ersten Schritte in C++ ist es o.k., wenn jeder AHA-Effekt mit einem Kommentar versehen wird, der beim späteren Lesen des Quellcodes das Gelernte verdeutlicht. Hier noch einmal unser "Hello, world!"-Beispiel mit (zu vielen) Kommentaren (Kommentare sind hier blau):

```
/*  
    hello.cpp  
    Erstes C++-Programm  
*/  
  
// Einbinden von Ausgabefunktionen:  
#include <iostream>  
  
// Hauptfunktion  
int main()  
{  
    // Beginn des Funktionsrumpfes  
    std::cout << "Hello, world!" // Ausgabe von "Hello, world!"  
              << std::endl; // Zeilenschaltung und Semikolon  
}  
// Ende von main()
```

"Hello, world!" im Überblick:



Nocheinmal die wichtigsten Fakten:

1. Das Einbinden des **Headerfiles 'iostream'** mittels der Präprozessoranweisung **'include'** ist u.a. für alle Ausgaben auf der Konsole erforderlich. Erst dadurch stehen **'cout'** und **'endl'** zur Verfügung. Präprozessoranweisungen beginnen mit einer Raute!
2. Der **Rückgabewert** von **main()** ist **immer 'int'**. Im Falle von `main()` kann jedoch laut Standard (ISO 14882) auf die Rückgabe mit der `'return'`-Anweisung verzichtet werden. Da der Microsoft-Compiler (bis einschl. .NET) damit Probleme hat, ist es sicherer, die Rückgabe auch bei `main()` in den Code zu schreiben. Da `main()` mit einer ganzen Zahl zurückkehrt (`int`) folgt der `return`-Anweisung die Zahl 0. Dies signalisiert i.d.R. einen ordnungsgemäßen Programmablauf (dazu später mehr).
3. Jede Anweisung endet mit einem **Semikolon!** Unser Programm enthält zwei Anweisungen, nämlich die Ausgabe der Zeichenkette "Hello, world!" und die Anweisung zur Rückkehr aus der Hauptfunktion (`return`). Daher finden wir am Ende dieser Anweisungen jeweils das Semikolon.
4. Der Rumpf jeder Funktion (also auch der Hauptfunktion `main()`) ist in **geschweifte Klammern** einzuschließen.

Die Standardeingabe (cin)

```
// Standardeingabe
#include <iostream>

int main()
{
    int einWert;
    std::cout << "Bitte Wert eingeben: ";
    std::cin >> einWert; // Eingabe von Tastatur lesen
    std::cout << "Sie gaben " << einWert << " ein." << std::endl;

    return 0;
}
```

Nachdem Kennenlernen des **Ausgabestromes 'cout'** ist es nur Konsequent, sich sofort auch mit dem **Eingabestrom 'cin'** vertraut zu machen.

Während **'cout'** üblicherweise mit der Konsole verknüpft ist (d.h. Der Ausgabestrom 'mündet' in einer Ausgabe auf der Konsole), werden **Eingaben** mittels **'cin'** im Normalfall **von der Tastatur** gelesen. (Anmerkung: es ist möglich, sowohl 'cout' als auch 'cin' umzuleiten)

Da Eingaben zur Weiterverarbeitung gespeichert werden müssen, wird **'cin'** hier mit einer Variable zur Speicherung von ganzen (engl.: integer, **int**) Zahlen eingesetzt.

Der per Tastatur eingegebene Wert wird in der **Variable** mit dem Namen *'einWert'* gespeichert. Dies wird in der Anweisung mit **'cin'** erreicht. Der Inhalt der **Variablen** *'einWert'* wird in der nächsten Anweisung mittels **'cout'** ausgegeben.

Übung:

Experimentieren Sie mit dem bisher kennengelernten Elementen von C++. Verändern Sie bisher geschriebene Programme und achten Sie auf Compilerfehler und verändertes Programmverhalten. Ab hier gilt: **Ausprobieren, Ausprobieren, Ausprobieren... Sein Sie mutig!**

grundlegende Datentypen

```
// Der Datentyp "string" erfordert eine neue Headerdatei: <string>
#include <string>
#include <iostream>

int main()
{
    using std::string;    // von hier ab kann string ohne std:: verwendet werden
    using std::cout;     // Verwendung von cout ohne std:: ab hier möglich
    using std::endl;     // auch endl kommt nun ohne std:: aus

    string hw = "Hello, world!"; // Zeichenkette
    cout << hw << endl;
}
```

```
// elementare Datentypen
#include <iostream>

int main()
{
    using namespace std;
    float    fpi    = 3.1415962; // Fließkommazahl (meist 32-bit)
    double   dval   = 42.19288;  // Fließkommazahl (meist 64-bit)
    int      ival   = 42;        // Ganzzahl (integer, meist 32-bit)
    char     cval   = 'A';      // Buchstabe, Ziffer oder Zeichen
    cout << "float-Wert: " << fpi << endl;
    cout << "double-Wert: " << dval << endl;
    cout << "int-Wert: " << ival << endl;
    cout << "char-Wert: " << cval << endl;
}
```

Übung:

Kombinieren Sie diese Datentypen mit Eingaben über die Tastatur (cin)!

Funktionen

```
// Funktionsaufrufe
#include <iostream>

// Funktion zur Ausgabe einer sehr berühmten Zeichenkette
void say_hello()
{
    std::cout << "Hello, world!" << std::endl;
}

// Funktion zum Summieren zweier Argumente
double sum(double a, double b)
{
    return (a+b);
}

// Hauptfunktion
int main()
{
    double x,y; // Deklaration!
    std::cout << "Bitte 2 Werte eingeben: ";
    std::cin >> x;
    std::cin >> y;
    double result;
    result = sum(x,y); // Funktionsaufruf, Rückgabewert wird verwendet!
    std::cout << "Die Summe ist " << result << std::endl;

    std::cout << "By the way: ";
    say_hello(); // Funktionsaufruf, kein Rückgabewert (void) !
    return 0;
}
```

Hinweis:

Funktionen können nicht an beliebiger Stelle im Quellcode plziert werden, ohne einige Dinge zu beachten. Im Augenblick sollte die Regel gelten, dass Funktionen nur verwendet werden können, nachdem sie deklariert und definiert wurden, hier also unterhalb der Funktionsdefinition! Nähere Informationen sind ein fortgeschrittenes Thema.

Verzweigungen (if, else)

```
// Verzweigungen
#include <iostream>

// Ausgabe einer Zeichenkette
void positive() {
    std::cout << "Positiver Wert\n";
}

void negative() {
    std::cout << "Negativer Wert\n";
}

// Hauptprogramm
int main() {
    double anyval = 0.0;
    std::cout << "Bitte Wert eingeben: ";
    std::cin >> anyval;
    if(anyval > 0) positive(); // Falls der Wert positiv war
    if(anyval < 0) negative(); // Falls der Wert negativ war
    if(anyval == 0) std::cout << "0!\n"; // Falls der Wert gleich 0 war

    if(anyval == 42.0) { // mehrere Anweisungen in Klammern!
        std::cout << "Aha, sie haben die Antwort auf die Frage ";
        std::cout << "nach dem Sinn des Lebens gefunden!\n";
    }
    else { // if() nicht erfüllt -> andernfalls!
        std::cout << "Wobei der Wert " << anyval;
        std::cout << " noch nicht das Gelbe vom Brot ist!\n";
    }

    return 0;
}
```

Übung:

Verändern Sie das o.a. Beispiel und beachten Sie die evtl. auftretenden Fehlermeldungen und das möglicherweise geänderte Programmverhalten.

Schleifen (for, while, do while)

```
// Beispiel einer FOR-Schleife
#include <iostream>

int main()
{
    int i; // für den Schleifenindex
    for(i=0;i<10;i++)
    {
        std::cout << "i hat den Wert: " << i << std::endl;
    }
}
```

```
// Beispiel einer WHILE-Schleife
#include <iostream>

int main()
{
    int i=0; // für den Schleifenindex (mit Wert 0)
    while(i<10)
    {
        std::cout << "i hat den Wert: " << i << std::endl;
        i++;
    }
}
```

```
// FOR-Schleife mit WHILE-Eigenschaften
#include <iostream>

int main()
{
    int i=0; // Initialisierung
    for(;i<10;) // leere Initialisierung und leere Weberschaltung
    {
        std::cout << "i hat den Wert: " << i << std::endl;
        i++; // Weberschaltung
    }
}
```