Grundlegende Anweisungen in C++

K. Vollmayr-Lee, O. Ippisch, Adrian Ngo

18. April 2011

1 Kompilieren

C++-Programme lassen sich mit dem GNU C++-Compiler `g++` übersetzen. Dabei kann mit der Option `-o` der Name der zu erzeugenden ausführbaren Datei angeben (sonst heißt das Executable einfach `a.out`).
Beispiel:

```
g++ testApp.cc                (erzeugt das Programm a.out)
g++ -o testApp testApp.cc     (erzeugt das Programm testApp)
```

Bei vielen Unix-Systemen befindet sich das aktuelle Verzeichnis aus Sicherheitsgründen nicht im Pfad in dem nach ausführbaren Programmen gesucht wird. Um ein übersetztes Programm zu starten ist es deshalb nötig ein `./` voranzustellen (`.` bezeichnet das aktuelle Verzeichnis), also: `./a.out` bzw. `./testApp`

2 C++-Syntax und Kommentare

Jeder C++-Befehl wird durch ein Semikolon (`;`) abgeschlossen. Ein Zeilenumbruch hat in C++ keine Bedeutung, d.h. man kann einen C++-Befehl auch über mehrere Zeilen verteilen.

Kommentare sollen den geschriebenen Code für Außenstehende kurz und knapp erläutern.

Kommentare sind ein unverzichtbarer Bestandteil eins Programms. Sie erklären die Funktionsweise von Programmen (nicht nur Anderen, sondern auch dem Autor, wenn er seinen Code nach längerer Pause wieder verstehen will!) und werden vom Compiler ignoriert.

`/* ... */`	kennzeichnet den Text zwischen `/*` und `*/` als Kommentar (mehrere Zeilen sind möglich)
`// ...`	kennzeichnet den Rest der Zeile hinter `//` als Kommentar

3 Datentypen

Datentyp	Variablendeklaration (Beispiel)	Zuweisung (Beispiel)	Darstellungsbereich
ganze Zahlen	`short i1,i2;`	`i1 = 3;`	$[-32768 \ldots 32767]$
	`int i1,i2;`		$[-2.14 \cdot 10^9 \ldots 2.14 \cdot 10^9]$
	`long i1,i2;`		$[-2.14 \cdot 10^9 \ldots 2.14 \cdot 10^9]$
	`unsigned i1,i2;`		$[0 \ldots 4.29 \cdot 10^9]$
	`unsigned long i1,i2;`		$[0 \ldots 4.29 \cdot 10^9]$
reelle Zahlen	`float f1,f2;`	`f1 = 3.2; f2 = 2.1E-3;`	$\pm[10^{-38} \ldots 10^{38}]$ 7 Stellen genau
	`double f1,f2;`		$\pm[10^{-308} \ldots 10^{308}]$ 16 Stellen genau
	`long double f1,f2;`		$\pm[10^{-4932} \ldots 10^{4932}]$ 19 Stellen genau
einzelne Zeichen	`char c1,c2;`	`c1 = 'R'`	
Zeichenketten	`string s1,s2;`	`s1 = "Farmer"`	
logischer Ausdrücke	`bool b1,b2;`	`b1 = true; b2 = false`	

3.1 Konstanten

Der Wert einer Konstanten kann nach der Definition nicht mehr geändert werden. Eine konstante Konstante wird definiert durch

```
const type VariablenName = Wert;
```

Beispiele:

```
const int Punktezahl = 10;
const double epsilon = 1E-20;
```

4 Ein- und Ausgabe

4.1 Ein- und Ausgabe auf dem Bildschirm:

Um die Ein- und Ausgabeanweisungen benutzen zu können, ist es notwendig zuerst eine Headerdatei einzubinden, die die entsprechenden Funktionsdeklarationen enthält (am Beginn der Programmdatei):

```
#include <iostream>
```

`std::cout` ist der Standard-Output. Die Ausgaben auf `std::cout` landen auf dem Bildschirm. Die Ausgabe von `std::endl` führt zu einem Zeilenumbruch. `std::cin` ist der Standard-Input und liest von der Tastatur.

4.1.1 Ausgabe

```
int value=42;
std::cout << "Auszugebender␣Text␣...";
std::cout << value << std::endl;   // gib Wert von value aus
std::cout << "Der␣Wert␣von␣value␣ist␣" << value << std::endl;
```

4.1.2 Eingabe

```
int variable;
std::cout << "Geben␣Sie␣eine␣ganze␣Zahl␣ein:␣"
std::cin >> variable;
```

4.2 IO-Manipulatoren

Mit IO-Manipulatoren lässt sich die Art der Ausgabe beeinflussen. Für die Manipulatoren mit Argument muss der Header `iomanip` eingebunden werden.

```
#include <iomanip>
```

Die Manipulatoren ohne Argument sind bereits in `iostream` deklariert.

4.2.1 Integer Manipulatoren

`dec`	Ausgabe als Dezimalzahl
`oct`	Ausgabe als Oktalzahl
`hex`	Ausgabe als Hexadezimalzahl

Beispiel:

```
#include <iostream>

int main()
{
    int a = 12;
    std::cout << "12␣is␣octal␣" << std::oct << a
              << "␣which␣is␣hexadecimal␣"
              << std::hex << a << std::endl;

}
```

Ausgabe:

```
12 is octal 14 which is hexadecimal c
```

4.2.2 Fließkomma Manipulatoren

`fixed`	Ausgabe als Festkommazahl
`scientific`	Ausgabe als Fließkommazahl
`setprecision(int p)`	Ausgabe mit `p` Stellen Genauigkeit
`setw(int w)`	Ausgabe mit `w` Stellen Breite

Beispiel:

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

int main()
{
    std::cout << "1/3␣is␣with␣three␣digits␣␣" << std::setw(15);
    std::cout << std::setprecision(3) << 1./3. << std::endl;
    std::cout << "1/3␣is␣with␣twelve␣digits␣" << std::setw(15);
    std::cout << std::setprecision(12) << 1./3. << std::endl;
}
```

Ausgabe:

```
1/3 is with three digits           0.333
1/3 is with twelve digits  0.333333333333
```

4.2.3 Format Manipulatoren

`left`	Linksbündige Ausgabe
`right`	Rechtsbündige Ausgabe

Beispiel:

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

int main()
{
    std::cout << std::left;
    std::cout << "1/3␣is␣with␣three␣digits␣␣" << std::setw(15);
    std::cout << std::setprecision(3) << 1./3. << std::endl;
    std::cout << "1/3␣is␣with␣twelve␣digits␣" << std::setw(15);
    std::cout << std::setprecision(12) << 1./3. << std::endl;
}
```

Ausgabe:

```
1/3 is with three digits  0.333
1/3 is with twelve digits 0.333333333333
```

4.2.4 Boole'sche Manipulatoren

`boolalpha`	Ausgabe als Zeichenkette `true`/`false`
`noboolalpha`	Ausgabe als `0`/`1`

Beispiel:

```
#include <iostream>

int main()
{
    bool a = true;
    std::cout << "without␣boolalpha␣flag␣the␣value␣of␣a␣is␣";
    std::cout << a << std::endl;
    std::cout << "with␣boolalpha␣flag␣the␣value␣of␣a␣is␣";
    std::cout << std::boolalpha << a << std::endl;
}
```

Ausgabe:

```
without boolalpha flag the value of a is 1
with boolalpha flag the value of a is true
```

4.3 Ein-/Ausgabe mit einer Datei:

Für die Ein-/Ausgabe mit Dateien ist die Headerdatei, ist es notwendig die Headerdatei `fstream` einzubinden:

```
#include <fstream>
```

die die Klassen `ofstream` (zur Ausgabe von Daten in eine Datei) und `ifstream` (zum Einlesen von Daten aus einer Datei) zur Verfügung stellt.

4.3.1 Ausgabe

```
double a = 100.0;
std::ofstream outfile("Dateiname.txt");
outfile << a;
```

schreibt den Wert der Variablen `a` in eine Datei mit dem Namen `Dateiname.txt`.

4.3.2 Eingabe

Eine Datei mit dem Namen `Eingabedatei.txt` enthält drei Zahlen, die entweder durch ein Leerzeichen oder einen Zeilenumbruch voneinander getrennt sind. Das folgende Programmfragment liest die Zahlen in die Variablen `a` bis `c` vom Typ `double` und gibt sie auf den Bildschirm aus:

```
double a,b,c;
std::ifstream infile("Eingabedatei.txt");
infile >> a >> b >> c;
std::cout << "a␣=␣" << a << "b␣=␣" << b << std::endl;
std::cout << "c␣=␣" << c << std::endl;
```

5 Arithmetische Berechnungen

5.1 Operationen

$+ \quad - \quad * \quad /$

5.2 Mathematische Funktionen

Um die nachfolgenden Funktionen benutzen zu können, muss die Headerdatei `cmath` am Anfang des Programmes eingebunden werden:

```
#include <cmath>
```

C++ Name	Funktion
`pow(x,y)`	x^y
`sin(x)`	
`cos(x)`	
`tan(x)`	
`asin(x)`	$\sin^{-1}(x)$ im Bereich $[-\pi/2, \pi/2]$
`acos(x)`	$\cos^{-1}(x)$ im Bereich $[0, \pi]$
`atan(x)`	$\tan^{-1}(x)$ im Bereich $[-\pi/2, \pi/2]$
`sinh(x)`	
`cosh(x)`	
`tanh(x)`	
`exp(x)`	e^x
`log(x)`	$\ln(x)$
`sqrt(x)`	$\sqrt{x}$
`fabs(x)`	$\|x\|$
`floor(x)`	größte ganze Zahl, die kleiner oder gleich x ist; z.B.: `floor(5.768)` $= 5$
`ceil(x)`	kleinste ganze Zahl, die nicht kleiner als x ist; z.B.: `ceil(5.768)` $= 6$
`fmod(x,y)`	Fließkomma-Rest der Division x/y. Ergebnis hat dasselbe Vorzeichen wie x.
`x % y`	Rest der ganzzahligen Division x/y. x und y müssen ganzzahlig sein.

6 Programmflusssteuerung

6.1 Vergleichsoperatoren

C++ Name	Funktion	Beispiel
`==`	Gleichheit	`i1 == i2`
`!=`	$\neq$	`i1 != i2`
`>`	$>$	`i1 > i2`
`<`	$<$	`i1 < i2`
`>=`	$\geq$	`i1 >= i2`
`<=`	$\leq$	`i1 <= i2`
`&&`	logisches UND	`(i1 != i2)&& (i1 == i3)`
`||`	logisches ODER	`(i1 == i2) || (i1 == i3)`

Vorsicht! In C++ unterschieden sich der Zuweisungsoperator `=` und der Operator zum Test auf Gleichheit `==`. Die Verwechslung von beiden ist einer der häufigen Fehler in C++.

6.2 Fallunterscheidungen

6.2.1 `if, else if, else`

```
// bedingte Ausfuehrung
if( Bedingung )
{
    Anweisungen
}

// bedingte Ausfuehrung mit Alternative
if( Bedingung )
{
    Anweisungen
}
else
{
    Anweisungen
}
```

```
// bedingte Ausfuehrung mit mehreren Alternativen
if( Bedingung )
{
    Anweisungen
}
else if
{
    Anweisungen
}
else
{
    Anweisungen
}
```

Beispiel:

```
if (i>0)
{
    std::cout << "i␣ist␣positiv" << std::endl;
}
else if (i<0)
{
    std::cout << "i␣ist␣negativ" << std::endl;
}
else
{
    std::cout << "i␣ist␣null" << std::endl;
}
```

6.2.2 switch/case/default

```
// Unterscheidung mehrerer Moeglichkeiten
switch ( CaseVariable )
{
    case Wert1:
        {
          Anweisungen
        }
        break; // wenn break fehlt wird der
               // folgende Block auch ausgefuehrt
    case Wert2a:
    case Wert2b:
        {
          Anweisungen
        }
        break;
    default:
        {
          Anweisungen
        }
}
```

6.3 Wiederholungen

6.3.1 while-Schleife

```
// wiederhole solange Bedingung wahr ist
while ( Bedingung )
{
  Anweisungen
}
```

Beispiel:

```
int i=0;
while ( i<10 )
{
  i=i+1;
  std::cout << "i=" << i << std::endl;
}
```

6.3.2 `for`-Schleife

```
// fuehre Initialisierungsanweisungen aus
// vor jeder Wiederholung fuehre Aktualisierungsanweisungen
// aus, wiederhole solange Bedingung war ist
for( Initialisierung; Bedingung; Aktualisierung )
{
  Anweisungen
}
```

Beispiel:

```
for( int i=0; i<10; ++i )
{
  std::cout << "i=" << i << std::endl;
}
```

6.3.3 `do`-Schleife

```
// fuehre einmal aus, wiederhole wenn Bedingung wahr ist
do
{
  Anweisungen  // Achtung: Anweisungen werden ausgefuehrt
               // bevor die Bedingung ueberprueft wird.
} while ( Bedingung );
```

Beispiel:

```
int i=0;
do
{
  i=i+1;
  std::cout << "i=" << i << std::endl;
} while ( i<11 );
```

7 Funktionen

Eine Funktion ist eine zusammengefasste Abfolge von Anweisungen. Es ist nützlich, eine benutzerdefinierte Funktion zu schreiben, wenn eine immer wieder kehrende Aufgabe erledigt werden soll. Funktionen helfen auch dabei ein Programm zu strukturieren und lesbarer zu machen. Die Ausführung aller C++-Programme startet mit der Funktion `main`. Drei Schritte sind bei der Benutzung einer Funktion wichtig:

1. Eine Funktion muss deklariert sein, bevor man sie benutzen kann:

   ```
   Rueckgabetyp Funktionsname( Argumentliste );

   double feetInchToMeter(int,double);
   void meterToFeetInch(double meter, int &feet,
                        double &inch);
   ```

   Der Rückgabetyp gibt an welchen Typ von Variable die Funktion zurück liefert. `void` bedeutet dass die Funktion keinen Wert zurück gibt.

   In der Argumentliste steht eine Liste von durch Komma getrennten Variablentypen, die die Eingabeparameter der Funktion beschreiben. Wahlweise darf nach dem Typ auch ein Variablennamen stehen. Das macht das Programm lesbarer.

   Steht nach dem Variablentyp ein `&` wird eine Referenz der Variablen übergeben, d.h. wird der Wert der Variablen in der Funktion verändert, dann ändert sich auch der Wert der übergebenen Variablen im aufrufenden Programmteil (dass kann auch dazu verwendet werden, mehrere Ergebnisse zurückzuliefern). Sonst wird eine Kopie der Variablen angelegt, Änderungen des Variablenwertes haben dann nur lokale Auswirkungen innerhalb der Funktion.

2. Verwendung der Funktion:

   ```
   Funktionsname( Parameterliste );

   int feet=6;
   double meter = feetInchToMeter(feet,1.5);
   double inch;
   meterToFeetInch(meter,feet,inch);
   ```

3. Jede Funktion muss an einer Stelle definiert (=implementiert) werden:

   ```
   Rueckgabetyp Funktionsname( Argumentliste )
   {
     Deklarationen und Anweisungen
   }
   ```

   Steht die Definition einer Funktion vor ihrem ersten Aufruf, dann ist Deklaration nicht nötig.

   Beispiel:

```
double feetInchToMeter(int feet,double inch)
{
    return 0.3048*feet+inch*0.0254;
};

void meterToFeetInch(double meter, int &feet,
                     double &inch)
{
    feet = int(meter/0.3048);
    // aendert Wert von meter in aufrufender Funktion nicht
    meter = meter - 0.3048*feet;
    inch = meter/0.0254;
};
```

8 Kommandozeilenparameter

Einem Programm lassen sich beim Aufruf Argumente übergeben. Dies ist praktisch, da sich die Kommandozeile bei wiederholten Aufrufen editieren lässt, so dass man nicht alle Werte neu eingeben muss, wenn sich einzelne ändern (im Gegensatz zum Einlesen von der Tastatur im Programm).

Die Funktion `main` hat entweder kein Argument oder zwei Argumente:

```
//************************************
// Filename: cmdline.cc
// Compiling: g++ -o cmdline cmdline.cc
//************************************

#include<iostream>

int main(int argc, char* argv[])
{
  for (int i=0;i<argc;++i)
      std::cout << argv[i] << std::endl;
  return 0;
}
```

Beispielaufruf:

```
./cmdline methodeX 2.4 5
```

- Das erste Argument `argc` gibt die Anzahl der Kommandozeilenparameter plus eins an, da der Name des aufgerufenen Programms immer zusätzlich als Kommandozeilenparameter übergeben wird.
- Das zweite Argument `argv` ist ein Feld von Zeichenketten, deren Werte sich mit dem eckige Klammer Operator abfragen lassen.
  - `argv[0]` enthält den Programmnamen, also `./cmdline`
  - `argv[1]` enthält den ersten Parameter, also `methodeX`
  - `argv[2]` enthält den zweiten Parameter, also `2.4`
  - `argv[3]` enthält den dritten Parameter, also `5`

8.1 Konvertierung von Zeichenketten

Die als Zeichenketten vorliegenden Kommandozeilenparameter lassen sich in Zahlen konvertieren. Dazu muss der Header `cstdlib` eingebunden werden.

```
#include<cstdlib>
```

Für die Konvertierung in eine Fließkommazahl dient die Funktion `atof`:

```
double a = atof(argv[2]);
```

Für die Konvertierung in eine Ganzzahl gibt es die entsprechende Funktion `atoi`.

```
int b = atoi(argv[3]);
```

9 Nützliche Online-Referenz

http://www.cplusplus.com

enthält ein Suchfeld für C++ Funktionen. Die meisten Beschreibungen sind mit einem kleinen Beispiel versehen.