Kurzeinführung zu C++

1 Erste Schritte in C++

Betrachten wir das folgende Programm:

```
1  #include <iostream>
2  #include <cmath>
3  using std::cout;  using std::endl;  using std::sqrt;
4
5  int main ()
6  {
7    double d;
8    d = 2.0;
9    cout << sqrt(d) << endl;     // Zahl ausgeben
10   cout << "sqrt(d)" << endl;   // Text ausgeben
11   return 0;
12 }
```

Die ersten beiden Zeilen binden die beiden Header-Dateien `iostream` (für `std::cout` und `std::endl`) und `cmath` (für `std::sqrt`) ein. Die dritte Zeile bewirkt, dass z.B. `std::cout` und `std::endl` durch `cout` und `endl` aufrufbar werden. Der Text hinter `//` (bis zum Zeilenende) ist ein Kommentar für den Benutzer und hat keinerlei Auswirkungen auf den Programmablauf.

Durch `double d` wird eine Variable `d` vom Typ `double` angelegt, d.h. es wird Speicherplatz reserviert. Damit kann `d` eine Fließkommazahl doppelter Genauigkeit (meist etwa 16 Dezimalstellen) aufnehmen. Das Semikolon `;` signalisiert das Ende eines Befehls und darf nicht vergessen werden. Durch die Zuweisung `d = 2.0` erhält `d` den Wert 2.0.

In der nächsten Programmzeile passiert eine ganze Menge: Mit `cout` werden Daten ausgegeben. Als erstes `sqrt(d)` (die Wurzel von `d`), dazu wird die Funktion `sqrt` aus `cmath` mit dem Argument `d` aufgerufen. Durch `endl` wird die aktuelle Zeile beendet und der Text auf den Bildschirm geschrieben.[1] In der nächsten Zeile steht `sqrt(d)` aber in Anführungszeichen, so dass es als Zeichenkette (string), also nur als eine Folge von Buchstaben, interpretiert wird. Daher wird in dieser Zeile der Text `sqrt(d)` ausgegeben.

Die vier zuletzt unter die Lupe genommenen Programmzeilen werden durch die geschweiften Klammern zu einer Einheit zusammengefasst. Sie bilden gemeinsam den Rumpf der Funktion `main`. Jedes C++-Programm benötigt eine Funktion dieses Namens; sie ist das Hauptprogramm und wird als erstes gestartet. Sie kann weitere Funktionen aufrufen, in unserem Fall beispielsweise die Funktion `sqrt`. Neben dem Rumpf, der festlegt, was eine Funktion macht, braucht sie auch einen Kopf, der festlegt, wie sie mit der Außenwelt in Verbindung tritt. Dieser Kopf ist hier `int main ()`. Er besagt, dass der Funktion keine Daten übergeben werden (denn zwischen den Klammern steht nichts) und dass sie einen ganzzahligen Wert (`int`) zurückgibt (erfolgt in diesem Falle durch `return 0;`). Wir könnten alternativ auch

```
int main (void)
```

schreiben. Die Funktion `sqrt` ist in `cmath` dagegen als

```
double sqrt (double);
```

deklariert: ihr wird eine `double`-Zahl übergeben, und sie gibt eine solche zurück.

[1] Normalerweise werden Ausgaben erst in einem Puffer zwischengespeichert und (wenn er voll ist) in einem Rutsch geschrieben, was meist effizienter ist als viele einzelne Ausgaben.

2 Übersetzen und Linken in der Konsole

Wenn Sie das Programm aus dem letzten Abschnitt mit Hilfe des Editors eingegeben und unter dem Namen `wurzel.cpp` abgespeichert haben, ist das zunächst nichts weiter als eine Textdatei, mit der der Computer noch nicht viel anfangen kann. Es muss erst noch übersetzt (kompiliert) werden, um in eine für den Computer lesbare Form (Maschinencode) umgewandelt zu werden; dazu dient der Compiler. Wir benutzen hier den C++-Compiler `g++` aus der GNU Compiler Collection[2], die den Bedingungen der GNU General Public License[3] (genau genommen steht GCC unter den Bedingungen der GPL Version 3[4] oder später mit der GCC Runtime Library Exception[5]) unterliegt, auf vielen Plattformen verfügbar und auch im CIP-Pool installiert ist. Bei anderen Compilern funktioniert es meist ähnlich, jedoch können die Optionen abweichend sein.

In `Code::Blocks` kann das Übersetzen bequem per Knopfdruck erledigt werden. Trotzdem schadet es nicht, sich damit zu beschäftigen, was dazu im Hintergrund abläuft, d.h. wie der Compiler aufgerufen wird und welche Schritte nötig sind, um den Programmcode zu kompilieren und zu linken. Probieren Sie also im Folgenden einfach mal aus, das kleine Programm aus dem vorigen Abschnitt „per Hand“ in der Konsole zu übersetzen. Dazu benötigen Sie allerding ein Linux-Betriebssystem[6], wie es im Rechnerraum installiert ist.

Der Compiler erzeugt aus `wurzel.cpp` zunächst den sogenannten Objektcode. Er wird in der Datei `wurzel.o` abgespeichert, wenn man den Compiler in der Linux-Shell mit

```
g++ -c wurzel.cpp
```

aufruft. In einem zweiten Schritt wird dieser Code zu einem ausführbaren Programm zusammengefügt. Dieser Vorgang wird auch als Linken bezeichnet und mit dem Aufruf

```
g++ -o wurzel wurzel.o -lm
```

ausgeführt. So wird das Programm `wurzel` erzeugt, das Sie nun mit `./wurzel` oder vielleicht auch `wurzel` starten können (die erste Version funktioniert immer, die zweite nur dann, wenn der sogenannte Suchpfad auch das aktuelle Verzeichnis „.“ enthält).

Wegen der Option `-o wurzel` bekommt das Programm den Namen `wurzel`. Beim Linken wird auch der Code für die Ausgabe und zur Berechnung der Wurzel hinzugefügt. Dieser findet sich in sogenannten Bibliotheken (libraries), in denen der entsprechende Objektcode abgelegt ist. Die mathematischen Routinen befinden sich in der Bibliothek `libm`, die durch die Option `-lm` hinzugefügt wird.[7] Weil diese Option sich erst auf den zweiten Schritt der Programmbearbeitung bezieht, steht sie (im Gegensatz zur üblichen Konvention) am Ende der Zeile.

Beide Schritte, Übersetzen und Linken, können Sie auch mit einem Aufruf

```
g++ -o wurzel wurzel.cpp -lm
```

hintereinander durchführen lassen.

Statt einer einzigen Quelldatei mit dem C++-Programm kann man dem Compiler auch mehrere Dateien übergeben, die dann zusammengelinkt werden. Darunter können auch Dateien mit Objektcode sein, bei denen dann der erste Bearbeitungsschritt übersprungen wird. Durch

```
g++ -o prog datei1.o datei2.o -lIGL
```

werden beispielsweise die Objektdateien `datei1.o` und `datei2.o` zusammen mit der IGPM-Grafikbibliothek `libIGL` zu einem ausführbaren Programm `prog` gelinkt.

[2] Webseite der GNU Compiler Collection: `http://gcc.gnu.org/`
[3] Die GNU General Public License (GPL) ist eine Lizenz für freie Software.
[4] Lizenztext der GPL Version 3: `http://www.gnu.org/licenses/gpl-3.0.html`
[5] Die GCC Runtime Exception stellt sicher, dass ein mit GCC compiliertes Programm nicht automatisch den Bedingungen der GPL unterliegt. `http://www.gnu.org/licenses/gcc-exception.html`
[6] oder ein linuxartiges Betriebssystem wie MacOS
[7] Bei neueren Compilern kann man auf `-lm` verzichten, da diese Bibliothek automatisch hinzugefügt wird.

Dem Compiler können noch weitere Optionen übergeben werden. Die Option `-g` erleichtert die Fehlersuche, indem sie einem Debugger (Fehlersuchprogramm) zusätzliche Informationen mitgibt, die allerdings das Programm aufblähen. Die Optionen `-O1` bis `-O3` optimieren das Programm. Das Übersetzen dauert dann länger, aber der produzierte Code wird (meist) schneller. Die Optionen `-Wall` und `-pedantic` geben zusätzliche Warnungen oder Fehlermeldungen über möglicherweise *gewagte* Programmiervorgänge aus – manchmal eine gute Hilfe.

Das Buch von Gough [4] führt verständlich in die Welt der Optionen der C- und C++-Compiler der GNU Compiler Collection ein. Die Liste aller Optionen des Compilers `g++` findet man auf den Info-Seiten, die man mit `info g++` aufrufen kann oder auch in der Online-Dokumentation auf der Webseite der GNU Compiler Collection.

Noch ein Hinweis: Nicht entmutigen lassen, wenn der Compiler beim Übersetzen Ihres Programmcodes Dutzende von Fehlermeldungen ausgibt! Oft handelt es sich dabei um Folgefehler, die verschwinden, wenn der allererste vom Compiler monierte Fehler (zum Beispiel eine vergessene schließende Klammer oder ein vergessenes Semikolon) beseitigt ist. In einem solchen Fall sollten Sie also in den Fehlermeldungen immer den *zuerst* aufgetretenen Fehler lokalisieren und beheben und anschließend Ihren Code erneut übersetzen.

3 Fehlersuche

In diesem Abschnitt wollen wir lernen, wie man mit einem Debugger arbeitet. Er ist ein wichtiges Werkzeug bei der Fehlersuche in Programmen. Wir behandeln hier zunächst ein triviales Beispiel, das aber die Vorgehensweise gut widerspiegelt. Danach stellen wir die wesentlichen Befehle des Debuggers zusammen.

3.1 Beispielsitzung zur Fehlersuche

Wir wollen eine kleine Variante des „Hello World“-Programms schreiben. Nehmen wir einmal an, dieses würde wie folgt aussehen:

```
1  #include<iostream>
2  using namespace std;
3
4  int j;
5
6  int main()
7  {
8    // This is the world's most famous program!
9    cout << "Hello" << endl;
10   cout << 10/j << endl;
11   cout << "GoodBye" << endl;
12 }
```

Nun wollen wir dieses Programm übersetzen. Geben Sie dazu im Shell-Fenster den Befehl

```
g++ -g -o hello hello.cpp
```

ein. Die Compileroption `-g` ermöglicht es, das Programm mit dem Debugger zu behandeln. So erhalten Sie ein ausführbares Programm `hello`. Starten Sie es, indem Sie in der Shell `./hello` aufrufen. Sie erhalten folgende Fehlermeldung:

```
Hello
Floating point exception (core dumped)
```

Wir wollen nun mit Hilfe des Debuggers dem Fehler auf die Spur kommen. Rufen Sie dazu mit dem Befehl

```
gdb hello
```

den Debugger auf. Sie erhalten eine mit (gdb) gekennzeichnete Eingabezeile. Diese funktioniert ähnlich wie die Shell, indem Sie Befehle eintippen welche Sie mit Enter ausführen.

Setzen Sie nun durch Eingabe von `break 9` einen Haltepunkt (breakpoint) in der Zeile

```
cout << "Hello" << endl;
```

und starten Sie das Programm mit Eingabe von `run`. Damit wird das Programm bis zu dem gesetzten Haltepunkt ausgeführt und dort angehalten (bevor diese Zeile ausgeführt wird). `gdb` gibt automatisch die Zeile aus, die als nächstes ausgeführt wird.

Mit Eingabe von `next` springt der Debugger zur nächsten Programmzeile und gibt den Text `Hello` aus. Die nächste Zeile wird angezeigt. Jetzt können wir noch mal Enter drücken, um den letzten Befehl (`next`) erneut auszuführen. Das bewirkt die Ausgabe der obigen Fehlermeldung. Das Programm ist also in der Zeile

```
cout << 10/j << endl;
```

abgestürzt.

Lassen Sie sich nun mit `print j` den Wert für `j` angezeigt: er beträgt Null! Offensichtlich versuchen wir in dieser Zeile, durch Null zu dividieren, was zu der Fehlermeldung und dem Abbruch des Programms führt.

Ersetzen Sie nun Zeile 4 zum Beispiel durch

```
int j=1;
```

und übersetzen und starten Sie das Programm. Nun erscheint auf dem Bildschirm die Ausgabe

```
Hello
10
Good Bye
```

Eine ausführliche Anleitung zum `gdb` finden Sie unter `https://sourceware.org/gdb/current/onlinedocs/gdb/`.

Falls Sie lieber mit einer graphischen Oberfläche statt mit einer textbasierten arbeiten, können Sie eins der vielen graphischen Front-Ends für den `gdb` benutzen. Beispielsweise bieten sich hier `kdbg` oder der in die Entwicklungsumgebung `Code::Blocks` eingebaute Debugger an.

4 C++-Kurzreferenz

Dieses Kapitel gibt eine kurze Übersicht über C++. Ausführliche Referenzen (auch über die aktuelleren Sprachstandards C++11 bis C++20) findet man online unter [?] oder [?].

4.1 Datentypen in C++

Elementare Datentypen		
Fließkommazahlen (floating-point numbers)		
`float`	`float x; x=3.0;`	(meist 4 Bytes)
`double`	`double y; y=-1.5e4;`	(meist 8 Bytes)
`long double`	`long double z; z=y;`	(meist 8–16 Bytes)
ganze Zahlen (integers)		
`short int, (un)signed short int`	`short int i; i=1;`	(meist 1 oder 2 Bytes)
`int, (un)signed int`	`int j; j=-3;`	(meist 2 oder 4 Bytes)
`long int, (un)signed long int`	`long int k; k=123;`	(meist 4 oder 8 Bytes)
Buchstaben (characters)		
`char, (un)signed char`	`char c; c='f';`	(meist 1, selten 2 Bytes)

Zeichenketten (strings)	
`std::string, char[]`	`std::string s="Hi!"; char cstr[]="abc";`
Logische Werte (booleans)	
`bool`	`bool b; b=true; b=false;` (meist wie `int`)
Aufzählung (enumeration)	

`enum`	`enum Antwort {Ja,Nein}; Antwort Ant;`
	`enum {Ja,Nein} Ant; Ant=Ja;`
	`enum Antwort {Ja=1,Nein=0};`

Nichts	
`void`	nur in zusammengesetzten Typen Zeiger: `void *Ptr`, Funktion: `void main ()`

Zusammengesetzte Datentypen

Felder (arrays)	

Datentyp Name[Konstante]	`int Zahlen[6];`
	`Zahlen[0]=3; Zahlen[5]=1;`
	`double Vektor[42];`

Zeiger (pointers)	

*Datentyp *Name*	`int *Zeiger;`
	`Zeiger=new int[10]; Zeiger[9]=1;`
	`delete[] Zeiger;`
	`Zeiger=new int; *Zeiger=1;`
	`delete Zeiger;`

Funktionen (functions)	

Datentyp Name(Paramliste)	`void main ()`
	`double Maximum ( double, double )`
	`void Vertausche ( double &, double & )`

Datentyp Name(Paramliste) {Definition}	`void Plus1 ( int &i ) { i++; }`
Datensätze (structures)	
`struct` *Name*	`struct Paar;`

`struct` *Name {Definition};*	`struct Paar { int i; double x; };`
	`Paar IntDbl;`
	`IntDbl.i=3; IntDbl.x=6.0;`

Klassen (classes)	
`class` *Name*	`class Vektor;`
`class` *Name {Definition};*	(structs sind Spezialfälle von Klassen)

4.2 Operatoren in C++

Operator	Erklärung	Beispiel
Vorzeichen		
`+ -`		`+a -3`
Arithmetik		
`+ - * /`		`1+3 a*b`
`%`	Rest der Division (modulo)	`5%3`
Zuweisung		
`=`		`a=1 b=x`
`+= -= *= /= %=`	Operation und Zuweisung	`a+=1`
Inkrement, Dekrement		
`++ --` (Präfix)	vorher erhöhen/erniedrigen	`++a --b`
`++ --` (Postfix)	nachher erhöhen/erniedrigen	`a++ b--`
Vergleich		
`== !=`	gleich, ungleich	`a==1 b!=x`
`< > <= >=`		`a<1 b>=x`
Logik		
`&& || !`	und, oder, nicht	`a&&b a||b !x`
Bits		
`& | ^ ~`	und, oder, xor, Komplement	`a&b a|b a^b ~x`
`<< >>`	nach links/rechts verschieben	`a<<2 b>>x`
`&= |= ^= <<= >>=`	mit Zuweisung	`a&=b a<<=3`
Ein-/Ausgabe		
`<< >>`	Ausgabe, Eingabe	`cout<<a cin>>b`
Zeiger (Pointer)		
`new`	Speicher belegen	`new int; new int[5];`
`delete delete[]`	Speicher löschen	`delete p; delete[] p;`
Adresse, Zeiger, Member		
`&`	Adresse einer Variablen	`&a`
`*`	Dereferenz (darauf zeigt der Zeiger)	`*p`
`. .*`	Memberauswahl	`MyStruct.x MyStruct.*p`
`-> ->*`	Memberauswahl (mit Dereferenz)	`p->x p1->*p2`
Klammern		
`[]`	Index	`a[3]`
`()`	Funktionsaufruf	`sin(x)`
`()`	Typumwandlung	`(double)3; double(3)`
Sichtbarkeit (Scope)		
`::`	globale Variable	`::x`
`::`	Namespace-/Klassenmember	`MyClass::x; std::cout`
Sonstiges		
`,`	Sequenz (liefert den Wert von `b`)	`a,b`
`?:`	wenn `a`, dann `b`, sonst `c`	`a?b:c`
`sizeof`	Größe	`sizeof(int); sizeof a;`

4.3 Kontrollstrukturen in C++

Hintereinanderausführung

stat1 stat2 ...

```
a=3; cout << a;
```

Klammerung

{ stat1 stat2 ... }

fasst mehrere Befehle zu einer Einheit zusammen

if-Anweisung

if (expr) stat

```
if (i<0) cout << "Zahl ist negativ!"<< endl;
```

if-else-Anweisung

if (expr) stat1 else stat2

```
if (a<0) b=sqrt(-a); else b=sqrt(a);
```

switch-Anweisung

switch (expr) { case const1: stat1 ... case constn: statn }

```
switch (i) { case 0: a=5; break; case 1: a=8; break; case 2: a=1; }
```

switch-Anweisung mit default

switch (expr) { case const1: stat1 ... case constn: statn default: stat }

```
switch (i) { case 0: a=5; break; case 1: a=8; break; default: a=1; }
```

while-Schleife

while (expr) stat

```
while (i>0) { i--; j*=2; }
```

do-Schleife

do stat while (expr);

```
do { i--; j*=2; } while (i>0);
```

for-Schleife

for (stat_init; expr; stat_inc) stat

```
for ( int i=0; i<n; i++ ) cout << x[i];
```

break-Anweisung

```
break;
```

verlässt eine Schleife

continue-Anweisung

```
continue;
```

bricht einen Schleifendurchgang ab

return-Anweisung

return expr;

verlässt eine Unterroutine und gibt den Wert *expr* zurück

5 Literaturverzeichnis

Literatur zu C++

[1] Breymann, Ulrich: *C++: eine Einführung.* Carl-Hanser-Verlag, München, 2016.

[2] Davis, S. R.: *C++ für Dummies.* mitp, 2000.

[3] Erlenkötter, H.: *C++. Objektorientiertes Programmieren von Anfang an.* Rowohlt Taschenbuch, Reinbek, 8. Auflage, 2000. http://www.erlenkoetter.de/html/c.htm.

[4] Gough, B. J.: *An Introduction to GCC - for the GNU Compilers gcc and g++.* A network theory manual. Network Theory Ltd., Bristol, 2005. http://www.network-theory.co.uk/docs/gccintro/.

[5] Josuttis, N.: *Objektorientiertes Programmieren in C++.* Addison-Wesley, Bonn, 2001. http://www.josuttis.com/cppbuch/.

[6] Lippman, S. B. und J. Lajoie: *C++ Primer.* mitp, Bonn, 2002. http://www.awprofessional.com/cpp_primer.

[7] Prata, S.: *C++ Primer Plus.* SAMS Publishing, Indianapolis, IN, 4. Auflage, 2001.

[8] Stroustrup, B.: *Die C++-Programmiersprache.* Addison-Wesley, Bonn, 4. Auflage, 2000. Deutsche Übersetzung der Special Edition, http://www.research.att.com/~bs/3rd.html.

[9] Wolf, J.: *Grundkurs C++: C++-Programmierung verständlich erklärt.* Galileo Computing, 2013.