

C++-Kurzeinführung¹

1 Erste Schritte in C++

Betrachten wir das folgende Programm:

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
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 `sqrt`) ein. Die dritte Zeile bewirkt, dass man in den Namensraum `std` wechselt. Dies hat zur Folge, dass die Befehle `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.² 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)
```

¹Stand: 11. April 2018

²Normalerweise werden Ausgaben erst in einem Puffer zwischengespeichert und (wenn er voll ist) in einem Rutsch geschrieben, was meist effizienter ist als viele einzelne Ausgaben.

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.

2 Übersetzen und Linken

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 Collection³, die den Bedingungen der GNU General Public License⁴ (genau genommen steht GCC unter den Bedingungen der GPL Version 3⁵ oder später mit der GCC Runtime Library Exception⁶) 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.

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

³Webseite der GNU Compiler Collection: <http://gcc.gnu.org/>

⁴Die GNU General Public License (GPL) ist eine Lizenz für freie Software.

⁵Lizenztext der GPL Version 3: <http://www.gnu.org/licenses/gpl-3.0.html>

⁶Die GCC Runtime Exception stellt sicher, dass ein mit GCC kompiliertes Programm nicht automatisch den Bedingungen der GPL unterliegt. <http://www.gnu.org/licenses/gcc-exception.html>

⁷Bei neueren Compilern kann man auf `-lm` verzichten, da diese Bibliothek automatisch hinzugefügt wird.

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

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.

Mit der Option `-std=c++11` teilen Sie dem Compiler mit, dass Sie Sprachfeatures oder die erweiterte Standardbibliothek von C++11 verwenden möchten. Dazu gehören z. B. das `auto`-Keyword, range-based `for` loops oder die Smartpointer-Typen `unique_ptr` und `shared_ptr`.

Das Buch von Gough [3] 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.

2.1 Neuere Compilerversionen im CIP-Pool

Wichtig: Mit Version 5.1 vom GCC gab es inkompatible Änderungen am ABI der C++-Standardbibliothek die zu Fehlermeldungen führen, wenn man versucht, Objektdateien, die mit verschiedenen Compilerversionen generiert wurden zu linken. Da die meisten Kursteilnehmer zuhause einen neueren Compiler haben werden, veröffentlichen wir unsere Testroutinen nur in Objektdateien, die die neue ABI verwenden. Die im CIP-Pool standardmäßig verwendeten Compiler sind dafür aber zu alt (Bei Erstellung dieser Dokumentation war dies GCC Version 4.8.5). Deshalb muss mithilfe von Environment-Modules⁸ eine neuere Compiler-Version geladen werden. Dazu verwenden Sie einfach den Befehl

```
module load gcc-5.2
```

Danach können Sie mit `g++ --version` überprüfen, dass Sie eine neuere Version des C++-Compilers verwenden. Um standardmäßig den neueren Compiler zu verwenden, kann die `module load`-Zeile auch in die Datei `HOME/.bashrc` eingetragen werden, so dass sie automatisch beim Start der Bash-Shell ausgeführt wird.

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.

⁸http://en.wikipedia.org/wiki/Environment_Modules_%28software%29

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 der `ddd` oder der in die Entwicklungsumgebung `Code::Blocks` eingebaute Debugger an.

4 C++-Kurzreferenz

Dieses Kapitel gibt eine kurze Übersicht über C++. Ausführliche Referenzen (sowohl über den aktuellen Sprachstandard C++14, als auch zum vorhergehenden Standard C++11, sowie dem noch immer weit verbreiteten Standard C++98 mit seiner geringfügigen Überarbeitung C++03) findet man online unter [12] oder [9].

Wer schon ein wenig Kenntnisse von C++(98) hat, findet in [6] eine gute Übersicht über nützliche Neuerungen in C++11/14, die zudem sehr unterhaltsam geschrieben ist.

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)
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)	
<i>Datentyp Name</i> [Konstante]	int Zahlen[6]; Zahlen[0]=3; Zahlen[5]=1; double Vektor[42];
Zeiger (pointers)	
<i>Datentyp *Name</i>	int *Zeiger; Zeiger=new int[10]; Zeiger[9]=1; delete[] Zeiger; Zeiger=new int; *Zeiger=1; delete Zeiger;
Datensätze (structures)	
struct <i>Name</i> struct <i>Name</i> { <i>Definition</i> };	struct Paar; struct Paar { int i; double x; }; Paar IntDb1; IntDb1.i=3; IntDb1.x=6.0;
Funktionen (functions)	
<i>Datentyp Name</i> (<i>Datenliste</i>)	void main ()

	double Maximum (double, double)
	void Vertausche (double &, double &)
<i>Datentyp Name(Datenliste) {Definition}</i>	void Plus1 (int &i) { i++; }
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(-else)-Anweisung

```
if (expr) stat1 [else stat2]
```

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

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

switch-Anweisung

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

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

```
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 Funktion und gibt den Wert *expr* zurück

5 Literaturverzeichnis

Literatur zu C++

[1] DAVIS, S. R.: *C++ für Dummies*. mitp, 2000.

- [2] ERLenkÖTTER, H.: *C++. Objektorientiertes Programmieren von Anfang an*. Rowohlt Taschenbuch, Reinbek, 8. Auflage, 2000. <http://www.erlenkoetter.de/html/c.htm>.
- [3] 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/>.
- [4] JOSUTTIS, N.: *Objektorientiertes Programmieren in C++*. Addison-Wesley, Bonn, 2001. <http://www.josuttis.com/cppbuch/>.
- [5] LIPPMAN, S. B. und J. LAJOIE: *C++ Primer*. mitp, Bonn, 2002. http://www.awprofessional.com/cpp_primer.
- [6] MEYERS, S.: *Effective Modern C++*. O'Reilly, 2014.
- [7] PRATA, S.: *C++ Primer Plus*. SAMS Publishing, Indianapolis, IN, 4. Auflage, 2001.
- [8] STROUSTRUP, B.: *The C++ Programming Language*. Addison-Wesley, Boston, 4. Auflage, 2013. <http://www.stroustrup.com/4th.html>.

Online-Referenzen

- [9] *C++-Dokumentation*. <http://www.cplusplus.com>.
- [10] *C++ FAQ Lite*. <http://www.dietmar-kuehl.de/mirror/c++-faq/>.
- [11] *C++ Samples*. <http://www.cppsamples.com/>.
- [12] *C/C++-Referenz*. <http://www.cppreference.com/index.html>.