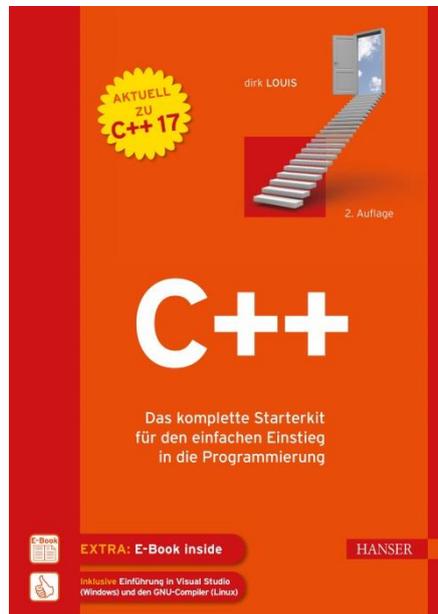


HANSER



Leseprobe

zu

C++

Das komplette Starterkit für den einfachen Einstieg in die Programmierung

von Dirk Louis

ISBN (Buch): 978-3-446-44597-0

ISBN (E-Book): 978-3-446-45388-3

Weitere Informationen und Bestellungen unter

<http://www.hanser-fachbuch.de/>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort	XXIII
Teil I: Grundkurs	1
1 Keine Angst vor C++!	3
1.1 Von C zu C++	4
1.1.1 Rückblick	4
1.1.2 Die strukturierte Programmierung	6
1.1.3 Chips sind billig, Programmierer teuer	8
1.1.4 Fassen wir zusammen	9
1.2 Von der Idee zum fertigen Programm	10
1.3 Näher hingeschaut: der C++-Compiler	12
1.3.1 Der Compiler ist ein strenger Lehrer	12
1.3.2 Definition und Deklaration	13
1.3.3 Das Konzept der Headerdateien	15
1.3.4 Namensräume	16
1.3.5 Der Compiler bei der Arbeit	18
1.3.6 ISO und die Compiler-Wahl	19
1.3.7 Der neue C++17-Standard	19
1.4 Übungen	20
2 Grundkurs: Das erste Programm	21
2.1 Hallo Welt! – das Programmgerüst	21
2.1.1 Typischer Programmaufbau	22
2.1.2 Die Eintrittsfunktion main()	23
2.1.3 Die Anweisungen	24
2.1.4 Headerdateien	26
2.1.5 Kommentare	27
2.2 Programmerstellung	28
2.2.1 Programmerstellung mit Visual Studio	28
2.2.2 Programmerstellung mit GNU-Compiler	35
2.2.3 Programmausführung	36

2.3	Stil	38
2.4	Übungen	39
3	Grundkurs: Daten und Variablen	41
3.1	Konstanten (Literals)	41
3.2	Variablen	44
3.2.1	Variablendefinition	44
3.2.2	Werte in Variablen speichern	47
3.2.3	Variablen bei der Definition initialisieren	48
3.2.4	Werte von Variablen abfragen	49
3.3	Konstante Variablen	50
3.4	Die Datentypen	51
3.4.1	Die Bedeutung des Datentyps	51
3.4.2	Die elementaren Datentypen	55
3.4.3	Weitere Datentypen	57
3.5	Typumwandlung	57
3.5.1	Typumwandlung bei der Ein- und Ausgabe	57
3.5.2	Automatische Typumwandlungen	60
3.5.3	Explizite Typumwandlungen	61
3.6	Übungen	62
4	Grundkurs: Operatoren und Ausdrücke	65
4.1	Rechenoperationen	65
4.1.1	Die arithmetischen Operatoren	65
4.1.2	Die mathematischen Funktionen	68
4.2	Ausdrücke	69
4.3	Die kombinierten Zuweisungen	71
4.4	Inkrement und Dekrement	71
4.5	Strings addieren	73
4.6	Weitere Operatoren	74
4.7	Übungen	74
5	Grundkurs: Kontrollstrukturen	75
5.1	Entscheidungen und Bedingungen	75
5.1.1	Bedingungen	76
5.1.2	Die Vergleichsoperatoren	77
5.1.3	Die logischen Operatoren	78
5.2	Verzweigungen	80
5.2.1	Die einfache if-Anweisung	80
5.2.2	Die if-else-Verzweigung	82
5.2.3	Die switch-Verzweigung	85

5.3	Schleifen	89
5.3.1	Die while-Schleife	89
5.3.2	Die do-while-Schleife	93
5.3.3	Die for-Schleife	95
5.3.4	Schleifen mit mehreren Schleifenvariablen	96
5.3.5	Performance-Tipps	97
5.4	Sprunganweisungen	97
5.4.1	Abbruchbefehle für Schleife	99
5.4.2	Abbruchbefehle für Funktionen	102
5.4.3	Sprünge mit goto	102
5.5	Fallstricke	102
5.5.1	Die leere Anweisung ;	102
5.5.2	Nebeneffekte in booleschen Ausdrücken	103
5.5.3	Dangling else-Problem	104
5.5.4	Endlosschleifen	105
5.6	Übungen	106
6	Grundkurs: Eigene Funktionen	109
6.1	Definition und Aufruf	110
6.1.1	Der Ort der Funktionsdefinition	111
6.1.2	Funktionsprototypen (Deklaration)	112
6.2	Rückgabewerte und Parameter	113
6.2.1	Rückgabewerte	115
6.2.2	Parameter	117
6.3	Lokale und globale Variablen	122
6.3.1	Lokale Variablen	122
6.3.2	Globale Variablen	123
6.3.3	Gültigkeitsbereiche und Verdeckung	124
6.4	Funktionen und der Stack	126
6.5	Überladung	128
6.6	Übungen	130
7	Grundkurs: Eigene Datentypen	131
7.1	Arrays	131
7.1.1	Definition	131
7.1.2	Auf Array-Elemente zugreifen	133
7.1.3	Initialisierung	133
7.1.4	Arrays in Schleifen durchlaufen	134
7.1.5	Arrays an Funktionen übergeben	137
7.1.6	Mehrdimensionale Arrays	137
7.1.7	Vor- und Nachteile der Programmierung mit Arrays	138

7.2	Aufzählungen	138
7.2.1	Definition	141
7.2.2	Variablen	141
7.2.3	Aufzählungstypen und switch-Verzweigungen	142
7.2.4	Die neuen enum class-Aufzählungen	142
7.3	Strukturen	143
7.3.1	Definition	144
7.3.2	Variablendefinition	145
7.3.3	Zugriff auf Elemente	146
7.3.4	Initialisierung	146
7.3.5	Arrays von Strukturen	146
7.4	Klassen	148
7.4.1	Definition	148
7.4.2	Variablen, Objekte und Konstruktoren	148
7.4.3	Zugriffsschutz	149
7.5	Übungen	152
8	Grundkurs: Zeiger und Referenzen	153
8.1	Zeiger	153
8.1.1	Definition	154
8.1.2	Initialisierung	154
8.1.3	Dereferenzierung	156
8.1.4	Zeigerarithmetik	158
8.2	Referenzen	159
8.3	Einsatzgebiete	159
8.3.1	call by reference	160
8.3.2	Dynamische Speicherreservierung	165
8.4	Übungen	171
9	Grundkurs: Noch ein paar Tipps	173
9.1	Wie gehe ich neue Programme an?	173
9.2	Wo finde ich Hilfe?	174
9.2.1	Hilfe zu Compiler-Meldungen	174
9.2.2	Hilfe bei der Lösung von Programmieraufgaben	175
9.2.3	Hilfe bei Programmen, die nicht richtig funktionieren	179
9.2.4	Debuggen	179
9.3	Programme optimieren	181

Teil II – Aufbaukurs: die Standardbibliothek	183
10 Aufbaukurs: Einführung	185
10.1 Bibliotheken verwenden	185
10.2 Hilfe zu den Bibliothekselementen	186
11 Aufbaukurs: Mathematische Funktionen	189
11.1 Die mathematischen Funktionen	189
11.1.1 Mathematische Konstanten	191
11.1.2 Verwendung der trigonometrischen Funktionen	192
11.1.3 Überläufe	192
11.2 Zufallszahlen	193
11.3 Komplexe Zahlen	195
11.4 Übungen	196
12 Aufbaukurs: Strings	197
12.1 String-Literale	197
12.1.1 Escape-Sequenzen	198
12.1.2 Zeilenumbrüche	200
12.2 Strings erzeugen	201
12.3 Strings aneinanderhängen	202
12.4 Strings vergleichen	202
12.5 Sonstige String-Manipulationen	205
12.6 C-Strings	206
12.7 Umwandlungen zwischen Strings und Zahlen	207
12.8 Übungen	208
13 Aufbaukurs: Ein- und Ausgabe	209
13.1 Daten auf die Konsole ausgeben	209
13.2 Formatierte Ausgabe	210
13.2.1 Ausgabebreite	210
13.2.2 Füllzeichen	211
13.2.3 Genauigkeit	211
13.2.4 Formatierte Ausgabe mit printf()	212
13.3 Deutsche Umlaute	213
13.4 Daten über die Konsole (Tastatur) einlesen	216
13.5 Fehlerbehandlung	217
13.6 Streams	219
13.7 Textdateien	220
13.7.1 In Textdateien schreiben	221
13.7.2 Aus Textdateien lesen	223

13.8	Binärdateien	226
13.9	Übungen	228
14	Aufbaukurs: Zeit und Datum	229
14.1	Zeit und Datum	229
14.2	Laufzeitmessungen	235
14.3	Übungen	237
15	Aufbaukurs: Container	239
15.1	Die STL	239
15.2	vector – ein intelligenter Daten-Container	242
15.2.1	Einsatz eines Containers	243
15.2.2	Größenmanagement von Containern	244
15.2.3	Typische Memberfunktionen	245
15.3	Der Gebrauch von Iteratoren	246
15.4	Die Algorithmen	249
15.4.1	generate()	252
15.4.2	stable_sort()	253
15.5	Schlüssel/Wert-Paare	254
15.6	Übungen	256
16	Aufbaukurs: Programme aus mehreren Quelltextdateien	257
16.1	Quelltext verteilen	257
16.1.1	Funktionen über Dateigrenzen hinweg verwenden	258
16.1.2	Klassen über Dateigrenzen hinweg verwenden	258
16.1.3	Variablen über Dateigrenzen hinweg verwenden	259
16.1.4	Typdefinitionen über Dateigrenzen hinweg verwenden	260
16.2	Mehrfacheinkopieren von Headerdateien verhindern	261
16.3	Übungen	263
Teil III – Objektorientierte Programmierung		265
17	OOP-Kurs: Klassen	267
17.1	Objektorientiert denken – objektorientiert programmieren	267
17.1.1	Objektorientiertes Programmieren	267
17.1.2	Wie sind Objekte beschaffen?	268
17.1.3	Wie findet man einen objektorientierten Lösungsansatz?	270
17.1.4	Objekte und Klassen	271
17.2	Klassendefinition	274
17.2.1	Zugriffsrechte	275

17.2.2	Quelltext- und Headerdatei	277
17.2.3	Klassen zu Visual-Studio-Projekten hinzufügen	280
17.3	Membervariablen	283
17.3.1	Anfangswerte	284
17.3.2	Private-Deklaration	288
17.3.3	Eingebettete Objekte	290
17.3.4	Konstante Membervariablen	292
17.3.5	Statische Membervariablen	293
17.4	Memberfunktionen	294
17.4.1	Definition innerhalb der Klassendefinition	294
17.4.2	Definition außerhalb der Klassendefinition	295
17.4.3	Der this-Zeiger	296
17.4.4	Statische Memberfunktionen	297
17.4.5	Konstante Memberfunktionen	298
17.4.6	Get/Set-Memberfunktionen	299
17.5	Die Konstruktoren	302
17.5.1	Definition und Aufruf	302
17.5.2	Ersatz- und Standardkonstruktoren	304
17.6	Der Destruktor	307
17.7	Übungen	308
18	OOP-Kurs: Vererbung	311
18.1	Das Prinzip der Vererbung	311
18.1.1	Der grundlegende Mechanismus	312
18.1.2	Die Syntax	313
18.1.3	Wann ist Vererbung gerechtfertigt?	314
18.1.4	Einige wichtige Fakten	315
18.2	Das Basisklassenunterobjekt	316
18.2.1	Zugriff	317
18.2.2	Instanzbildung	320
18.3	Die Zugriffsspezifizierer für die Vererbung	322
18.4	Verdecken, überschreiben und überladen	323
18.4.1	Verdeckung	324
18.4.2	Überladung	324
18.4.3	Überschreibung	325
18.5	Der Destruktor	325
18.6	Mehrfachvererbung	326
18.7	Übungen	326
19	OOP-Kurs: Polymorphie	329
19.1	Grundprinzip und Implementierung	330
19.2	Späte und frühe Bindung	333

19.2.1	Frühe Bindung	333
19.2.2	Späte Bindung	334
19.3	Generische Programmierung	335
19.3.1	Basisklassen-Arrays	336
19.3.2	Basisklassenparameter	338
19.4	Typidentifizierung zur Laufzeit (RTTI)	339
19.4.1	Umwandlung mit <code>dynamic_cast</code>	339
19.4.2	Der <code>typeid()</code> -Operator	341
19.5	Abstrakte Klassen	341
19.5.1	Rein virtuelle Funktionen	342
19.5.2	Abstrakte Klassen	342
19.6	Übungen	343
20	OOP-Kurs: Ausnahmebehandlung	345
20.1	Fehlerprüfung mit Ausnahmen	346
20.2	Ausnahmen abfangen	348
20.3	Ausnahmen auslösen	351
20.4	Programmfluss und Ausnahmebehandlung	353
20.4.1	Wo wird der Programmfluss nach einer Ausnahme fortgesetzt? ..	354
20.4.2	Die Problematik des gestörten Programmflusses	354
20.5	Übungen	356
Teil IV – Profikurs	357	
21	Profikurs: Allgemeine Techniken	359
21.1	Vorzeichen und Überlauf	359
21.2	Arithmetische Konvertierungen	361
21.3	Lokale <code>static</code> -Variablen	361
21.4	Der <code>?:</code> -Operator	362
21.5	Bit-Operatoren	362
21.5.1	Multiplikation mit 2	363
21.5.2	Division durch 2	364
21.5.3	Klein- und Großschreibung	364
21.5.4	Flags umschalten	365
21.5.5	Gerade Zahlen erkennen	365
21.6	Zeiger auf Funktionen	367
21.7	Rekursion	369
21.8	<code>constexpr</code> -Funktionen	371
21.9	Variablendefinition in <code>if</code> und <code>switch</code>	372

22 Profikurs: Objektorientierte Techniken	375
22.1 Zeiger auf Memberfunktionen	375
22.2 Friends	377
22.3 Überladung von Operatoren	378
22.3.1 Syntax	378
22.3.2 Überladung des Inkrement-Operators ++	379
22.3.3 Überladung arithmetischer Operatoren +, +=	380
22.3.4 Überladung der Streamoperatoren <<>>	381
22.4 Objekte vergleichen	382
22.4.1 Gleichheit	382
22.4.2 Größenvergleiche	384
22.5 Objekte kopieren	386
23 Profikurs: Gültigkeitsbereiche und Lebensdauer	391
24 Profikurs: Templates	395
24.1 Funktionen-Templates	396
24.2 Klassen-Templates	397
25 Profikurs: Reguläre Ausdrücke	401
25.1 Syntax regulärer Ausdrücke	401
25.1.1 Zeichen und Zeichenklassen	402
25.1.2 Quantifizierer	403
25.1.3 Gruppierung	404
25.1.4 Assertionen (Anker)	405
25.2 Musterabgleich mit regulären Ausdrücken	405
25.3 Suchen mit regulären Ausdrücken	406
25.4 Ersetzen mit regulären Ausdrücken	407
26 Profikurs: Lambda-Ausdrücke	409
26.1 Syntax	409
26.2 Einsatz	411
A Anhang A: Lösungen	413
B Anhang B: Die Beispiele zum Buch	433
B.1 Installation der Visual Studio Community Edition	433
B.2 Ausführung der Beispielprogramme	436
B.2.1 Ausführung mit VS Community Edition 2017	437

B.2.2	Ausführung mit beliebigen integrierten Entwicklungsumgebungen	438
B.2.3	Ausführung mit GNU-Konsolen-Compiler	439
C	Anhang C: Zeichensätze	441
C.1	Der ASCII-Zeichensatz	441
C.2	Der ANSI-Zeichensatz	442
D	Anhang D: Syntaxreferenz	445
D.1	Schlüsselwörter	445
D.2	Elementare Typen	446
D.3	Strings	447
D.4	Operatoren	448
D.5	Ablaufsteuerung	449
D.6	Ausnahmebehandlung	451
D.7	Aufzählungen	451
D.7.1	enum	451
D.7.2	enum class (C++11)	452
D.8	Arrays	452
D.9	Zeiger	453
D.10	Strukturen	453
D.11	Klassen	454
D.12	Vererbung	457
D.13	Namensräume	457
E	Anhang E: Die Standardbibliothek	459
E.1	Die C-Standardbibliothek	459
E.2	Die C++-Standardbibliothek	460
Index		463

Vorwort

Sie besitzen einen Computer und wissen nicht, wie man programmiert?

Das ist ja furchtbar! Jetzt erzählen Sie mir nur nicht, dass Sie Ihren Computer nur zum Spielen und Internetsurfen benutzen. Dann wäre ich wirklich enttäuscht.

Ach so, jemand hat Ihnen erzählt, dass Programmieren sehr kompliziert sei und viel mit Mathematik zu tun hätte.

Tja, dann wollte sich dieser jemand wohl ein wenig wichtigmachen oder hat selbst nichts vom Programmieren verstanden, denn Programmieren ist nicht schwieriger als Kochen oder das Erlernen einer Fremdsprache. Und mehr mathematisches Verständnis als es von einem Schüler der sechsten oder siebten Klasse verlangt wird, ist definitiv auch nicht nötig.

Reizen würde Sie das Programmieren schon, aber Sie wissen ja gar nicht so recht, was Sie programmieren könnten.

Keine Angst, sowie Sie mit dem Programmieren anfangen, werden Ihnen zahlreiche Ideen kommen. Und weitere Anregungen finden sich in guten Lehrbüchern zuhauf beziehungsweise ergeben sich beim Austausch mit anderen Programmierern.

Immer noch Zweifel? Sie würden sich das Programmieren am liebsten im Selbststudium beibringen? Sie suchen ein Buch, das Sie nicht überfordert, mit dem Sie aber dennoch richtig professionell programmieren lernen?

Aha, Sie sind der unsicher-anspruchsvolle Typ! Dann dürfte das vorliegende Buch genau richtig für Sie sein. Es ist nicht wie andere Bücher primär thematisch gegliedert, sondern in Stufen – genauer gesagt vier Stufen, die Sie Schritt für Schritt auf ein immer höheres Niveau heben.

Aufbau des Buchs

Die **erste Stufe** ist der Grundkurs. Hier schreiben Sie Ihre ersten Programme und lernen die Grundzüge der Programmierung kennen. Danach besitzen Sie fundiertes Basiswissen und können eigenständig Ihre ersten Programmideen verwirklichen.

Bestimmte Aufgaben kehren bei der Programmierung immer wieder: beispielsweise das Abfragen des Datums, die Berechnung trigonometrischer Funktionen, die Verwaltung größerer Datenmengen oder das Schreiben und Lesen von Dateien. Für diese Aufgaben gibt es in der C++-Standardbibliothek vordefinierte Elemente. Wie Sie diese nutzen, erfahren Sie im Aufbaukurs – der **zweiten Stufe**.

Die **dritte Stufe** stellt Ihnen die objektorientierte Programmierung vor und lehrt Sie, wie Sie den Code immer größerer Programme sinnvoll organisieren.

Die **vierte Stufe** schließlich stellt Ihnen noch einige letzte C++-Techniken vor, die Sie vermutlich eher selten einsetzen werden, die ein professioneller C++-Programmierer aber kennen sollte.

Abgerundet wird das Buch durch einen umfangreichen **Anhang**, der unter anderem eine C++-Syntaxübersicht und eine Kurzreferenz der Standardbibliothek beinhaltet.

Nicht verzagen!

Natürlich gibt es auch Zeiten des Verdrusses und des Frusts. Oh ja, die gibt es! Aber seien wir ehrlich: Wäre der Weg nicht so steinig, wäre die Freude am Ziel auch nicht so groß. Was sind das denn für trostlose Gesellen, die in ihrer gesamten Zeit als Programmierer noch keine Nacht durchwacht haben, weil sie den Fehler, der das Programm immer zum Abstürzen bringt, nicht finden konnten? Und was soll man von einem Programmierer halten, der noch nie aus Versehen ein Semikolon hinter eine if-Bedingung gesetzt hat? (Und dem die Schamesröte ins Gesicht schoss, als er einen vorbeikommenden Kollegen um Hilfe bat und ihn dieser nach einem flüchtigen Blick auf den Quellcode auf den Fehler aufmerksam machte.) Sind das überhaupt echte Programmierer?

Wer programmieren lernen will, der muss auch erkennen, dass bei der Programmierung nicht immer alles glatt geht. Das ist nicht ehrenrührig, man darf sich nur nicht unterkriegen lassen. Sollten Sie also irgendwo auf Schwierigkeiten stoßen – sei es, dass Sie etwas nicht ganz verstanden haben oder ein Programm nicht zum Laufen bekommen –, versuchen Sie sich nicht zu sehr in das Problem zu verbohren. Legen Sie eine kleine Pause ein oder lesen Sie erst einmal ein wenig weiter – oftmals klärt sich das Problem danach von selbst.

Dirk Louis

Saarbrücken, im Frühjahr 2018

2

Grundkurs: Das erste Programm

Nun ist es endlich so weit! Wir werden unser erstes C++-Programm erstellen.

■ 2.1 Hallo Welt! – das Programmgerüst

Es gibt eine Reihe von typischen Programmelementen, die man in so gut wie jedem C++-Programm wiederfindet. Diese Elemente werden wir uns jetzt einmal näher anschauen.

Bevor ich Ihnen die Programmelemente im Einzelnen vorstelle, sollten wir jedoch einen Blick auf den vollständigen Quelltext des Programms werfen. Wenn Sie bereits über etwas Programmiererfahrung verfügen, werden Sie das Programm womöglich sogar wiedererkennen: Es ist eine Adaption des klassischen „Hello World“-Programms aus der C-Bibel von Kernighan und Ritchie.

Listing 2.1 Das erste Programm (aus HalloWelt.cpp)

```
/*  
 * Hallo Welt-Programm  
 *  
 * gibt einen Gruss auf den Bildschirm aus  
 */  
  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout << "Hallo Welt!" << endl;  
  
    return 0;  
}
```

Packen Sie jetzt bitte Ihr Sezierbesteck aus und schärfen Sie Ihren Verstand. Wir beginnen mit der Analyse.

**Achtung!**

C++ unterscheidet streng zwischen Groß- und Kleinschreibung. Beachten Sie dies, wenn Sie in Abschnitt 2.2 den Quelltext in Ihren Editor eingeben.

2.1.1 Typischer Programmaufbau

Die landläufige Vorstellung von einem Programm ist gemeinhin eine Folge von Anweisungen, die vom Rechner nacheinander ausgeführt werden. Blickt man aber in die Quelltextdatei eines beliebigen C++-Programms, offenbart sich ein ganz anderes Bild.

Tatsächlich bestehen C++-Programme aus:

- Kommentaren
- Präprozessordirektiven
- Namensräumen
- Deklarationen und Definitionen
- einer Eintrittsfunktion namens `main()`

Natürlich gibt es auch Anweisungen, doch existieren diese nur als untergeordnete Elemente in den Definitionen der Funktionen!

Listing 2.2 Die typischen Elemente eines C++-Programms

```
/*
 * Hallo Welt-Programm          // mehrzeiliger Kommentar
 *
 * gibt einen Gruss auf den Bildschirm aus
 */

#include <iostream>           // Präprozessor-Direktive
using namespace std;        // Namensraum-Einbindung

int main()                   // Definition der Eintrittsfunktion
{
    cout << "Hallo Welt!" << endl;

    return 0;
}
```

Das Verhältnis aus Anweisungen (in Listing 2.1 fett hervorgehoben) und Elementen, die vornehmlich der Organisation des Quelltextes dienen (Präprozessordirektiven, Definitionen etc.), ist nicht immer so drastisch wie in diesem HalloWelt-Programm. Doch eines können und sollten Sie aus diesem Beispiel bereits ablesen: Programmierung hat auch viel mit Codeorganisation zu tun!

**Merksatz**

Bei der C++-Programmierung – wie im Übrigen bei der Programmierung mit jeder modernen Programmiersprache – genügt es nicht, sich zu überlegen, welche Anweisungen in welcher Reihenfolge zur effizienten Erledigung einer Aufgabe benötigt werden (Algorithmus). Sie müssen sich auch Gedanken darüber machen, wie Sie Ihren Quelltext organisieren.

Fürs Erste werden wir die Codeorganisation so einfach wie möglich halten. Konkret bedeutet dies, dass wir während unserer ersten Gehversuche mit C++ einfach unsere gesamten Anweisungen in die `main()`-Funktion schreiben werden.

Wo aber kommt diese `main()`-Funktion her? Und welche Bedeutung haben die anderen Elemente des Grundgerüsts aus Listing 2.1?

2.1.2 Die Eintrittsfunktion `main()`

Wenn Sie ein C++-Programm aufrufen, wird der Code des Programms in den Arbeitsspeicher geladen und vom Prozessor ausgeführt. Doch mit welchem Code beginnt die Ausführung des Programms?

Per Konvention beginnen C++-Programme immer mit einer Funktion namens `main()`. Wenn Sie einen Quelltext zu einer `.exe`-Datei kompilieren lassen, generiert der Compiler automatisch Startcode, der dafür sorgt, dass die Programmausführung mit der ersten Anweisung in `main()` beginnt.

Ihre Aufgabe ist es daher, in Ihrem Programmquelltext eine passende `main()`-Eintrittsfunktion zu definieren:

```
int main()
{
    // hier können Sie eigenen Code einfügen

    return 0;
}
```

Was diese Definition im Einzelnen zu bedeuten hat, werden Sie erst in Kapitel 6 erfahren, wenn wir uns intensiver mit der Definition von Funktionen beschäftigen. Bis dahin ist nur eines wichtig: Sie dürfen den Definitionscode nicht verändern, da die Funktion sonst nicht mehr vom Compiler als Eintrittsfunktion erkannt wird.

Wenn Sie also das `int` vergessen oder den `return`-Befehl falsch schreiben oder versuchen, die Funktion von `main()` in `start()` umzutaufen, so werden Sie dafür bei der Programmerstellung entsprechende Fehlermeldungen ernten. Und achten Sie auch auf die Groß- und Kleinschreibung. Für C++ sind `main` und `Main` nicht zwei Schreibweisen eines Namens, sondern ganz klar zwei verschiedene Namen!

**Merksatz**

C++ unterscheidet strikt zwischen Groß- und Kleinschreibung!

Ich sollte allerdings noch erwähnen, dass es eine zweite Variante für die Definition der Eintrittsfunktion `main()` gibt:

```
int main(int argc, char *argv[])
{
    // hier können Sie eigenen Code einfügen

    return 0;
}
```

Ja, manche Compiler erlauben sogar noch weitere Varianten. Grundsätzlich sollten Sie sich aber auf die beiden obigen Varianten beschränken, da nur so sichergestellt ist, dass sich Ihr Programm mit jedem ANSI-kompatiblen Compiler übersetzen lässt.



Wenn Sie den Compiler anweisen, aus einem Quelltext, der keine korrekt definierte `main()`-Eintrittsfunktion enthält, ein `.exe`-Programm zu erzeugen, werden Sie am Ende des Erstellungsprozesses vom Linker eine Fehlermeldung erhalten, dass die im Startcode referenzierte `main()`-Funktion nicht gefunden werden konnte.

2.1.3 Die Anweisungen

Innerhalb der geschweiften Klammern unserer `main()`-Funktion können wir nun endlich die Anweisungen aufsetzen, die bei Start des Programms ausgeführt werden sollen. Im Falle unseres ersten Beispielprogramms bescheiden wir uns mit einer einzigen Zeile, die den Text „Hallo Welt!“ ausgeben soll.

```
int main()
{
    cout << "Hallo Welt!" << endl;

    return 0;
}
```

Was bewirkt die obige Anweisung? Zunächst muss man wissen, dass `cout` ein vordefiniertes Objekt ist, welches die Konsole repräsentiert.

Die Konsole ist ein spezielles Programm des Betriebssystems (siehe Kasten), über das der Anwender Befehle ans Betriebssystem schicken kann. Für uns als Programmierer ist sie interessant, weil wir sie zum Datenaustausch zwischen unseren Programmen und den Anwendern nutzen können. Wir ersparen uns also den Aufbau einer eigenen Benutzeroberfläche und können uns ganz auf den reinen C++-Code konzentrieren.



Die Konsole

Die meisten PC-Benutzer, vor allem Windows- oder KDE-Anwender, sind daran gewöhnt, dass die Programme als Fenster auf dem Bildschirm erscheinen. Dies erfordert aber, dass das Programm mit dem Fenstermanager des Betriebssystems kommuniziert und spezielle Optionen und Funktionen des Betriebssystems nutzt. Programme, die ohne fensterbasierte, grafische Benutzeroberfläche (GUI = graphical user interface) auskommen, können hierauf jedoch verzichten und stattdessen die Konsole zum Datenaustausch mit dem Benutzer verwenden.

Die Konsole ist eine spezielle Umgebung, die dem Programm vorgaukelt, es lebe in der guten alten Zeit, als es noch keine Window-Systeme gab und immer nur ein Programm zurzeit ausgeführt werden konnte. Dieses Programm konnte dann uneingeschränkt über alle Ressourcen des Rechners verfügen – beispielsweise die Tastatur, das wichtigste Eingabegerät, oder auch den Bildschirm, das wichtigste Ausgabegerät. Der Bildschirm war in der Regel in den Textmodus geschaltet, wurde also nicht aus Pixelreihen, sondern aus Textzeilen aufgebaut.

Unter Windows heißt die Konsole MS-DOS-Eingabeaufforderung oder auch nur Eingabeaufforderung und kann je nach Betriebssystem über **Start/Programme** oder **Start/Programme/Zubehör** aufgerufen werden.

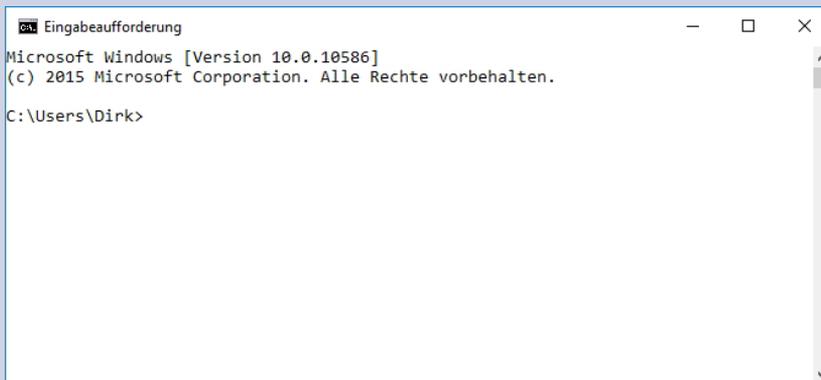


Bild 2.1 Die Konsole von Windows 8 (mit invertiertem Hintergrund)

Damit wir innerhalb eines Programms auf die Konsole zugreifen können, muss es im Programmcode aber ein Element geben, welches die Konsole repräsentiert. Dieses Element ist wie gesagt `cout`.

Allerdings handelt es sich bei `cout` nicht um ein Element, das fest in die Sprache integriert es. Vielmehr verbirgt sich hinter `cout` ein Objekt, das im Code der C++-Standardbibliothek definiert ist. Gleiches gilt im Übrigen auch für den Operator `<<`, der Ausgaben an `cout` schickt, sowie `endl`, das in der Ausgabe einen Zeilenumbruch (**end-of-line** = Zeilenende) erzeugt.

Unsere Anweisung schickt also zuerst mit Hilfe des `<<`-Operators den Text `Hallo Welt!` und dann noch einen Zeilenumbruch (`endl`) zur Konsole (`cout`):

```
cout << "Hallo Welt!" << endl;
```

Zwei Punkte an diesem Code bedürfen noch einer besonderen Erwähnung:



Merksatz

Texte, die ein Programm verarbeitet, werden auch als *Strings* bezeichnet und stehen in Anführungszeichen "", damit der Compiler sie vom Programmcode unterscheiden kann.



Merksatz

Anweisungen dürfen in C++ nur innerhalb von Funktionen stehen und müssen mit einem Semikolon abgeschlossen werden.

2.1.4 Headerdateien

Erinnern Sie sich, was in Kapitel 1 über die Verwendung von Elementen gesagt wurde, die nicht in die Sprache integriert sind: Sie müssen im Programmcode einmal definiert und in jeder Quelltextdatei, in der sie verwendet werden, deklariert werden.

Wie sieht es also mit der Definition aus? Die Definition von `cout`, `<<` und `endl` findet sich im Code der C++-Standardbibliothek. Dieser Code wird bei der Programmerstellung vom Linker automatisch mit Ihrem Code zur ausführbaren `.exe`-Datei verbunden. Wir müssen uns um diesen Teil nicht weiter kümmern. (Außer der Compiler wäre nicht korrekt konfiguriert und findet die Bibliotheksdateien nicht. Diese stehen übrigens meist in einem Verzeichnis *lib* und der Pfad zu diesem Verzeichnis kann über die Compiler-Optionen eingestellt werden.)

Bleibt noch die Deklaration. Da wir `cout`, `<<` und `endl` in unserer Quelltextdatei `HalloWelt.cpp` verwenden, müssen wir die Elemente dem Compiler auch in dieser Datei bekannt machen. Wir könnten dazu so vorgehen, dass wir in der Fachliteratur, der Bibliotheksdokumentation oder – soweit vorhanden – gar direkt im Quelltext der Bibliothek nachschlagen, wie die betreffenden Bibliothekselemente definiert sind, und uns daraus die Deklarationen ableiten, die wir dann über `main()` in den Quelltext einfügen.

Sie werden mir allerdings sicher zustimmen, dass diese Verfahrensweise recht mühselig, kompliziert und fehleranfällig wäre. Die C++-Standardbibliothek stellt daher für jeden Themenbereich, den die Bibliothek abdeckt, eine passende Headerdatei zur Verfügung, in der die benötigten Deklarationen schon gesammelt sind. Die Headerdatei für alle Bibliothekselemente, die mit der Ein- und Ausgabe zu tun haben, heißt `iostream` und kann mit der Präprozessor-Direktive `#include` einkopiert werden

```
#include <iostream>

int main()
{
    ...
}
```

Auch diese Technik ist Ihnen – in der Theorie – bereits in Kapitel 1.3.3 vorgestellt worden. Die wichtigsten Fakten möchte ich aber trotzdem hier noch einmal zusammenfassen.

Die `#include`-Direktive sucht nach der angegebenen Datei. Da der Dateiname in eckigen Klammern steht, wird die Datei im Include-Pfad des Compilers gesucht. (Dieser ist nach der Installation üblicherweise automatisch so eingestellt, dass er auf das Verzeichnis mit den Headerdateien der C++-Standardbibliothek verweist. Sie müssen sich in der Regel also nicht weiter um diese Einstellung kümmern.)

Der Inhalt der Datei wird dann an der Stelle der Direktive in die Quelltextdatei einkopiert.

Faktisch fügt die obige `#include`-Direktive also die Deklarationen aller IO-Elemente der C++-Standardbibliothek ein, sodass wir diese Elemente (darunter eben auch `cout`, `<<` und `endl`) verwenden können. Die Einbindung des Namensraums `std` dient dann nur noch der Bequemlichkeit, damit wir die Bibliothekselemente allein mit ihrem Namen ansprechen können (also `cout` statt `std::cout`, siehe Kapitel 1.3.4).



Das engl. Akronym IO steht für Input/Output, zu deutsch also Ein- und Ausgabe.

2.1.5 Kommentare

Wir haben nun fast alle Bestandteile des Quelltextes analysiert. Übrig geblieben sind allein die ersten einleitenden Zeilen:

```
/******
 * Hallo Welt-Programm
 *
 * gibt einen Gruss auf den Bildschirm aus
 */

#include <iostream>
...
```

Bei diesen Zeilen handelt es sich um einen Kommentar. Kommentare dienen dazu, erklärenden Text direkt in den Quellcode einzufügen – quasi als Erklärung oder Gedankenstütze für den Programmierer.

C++ kennt zwei Formen des Kommentars:

- Will man eine einzelne Zeile oder den Rest einer Zeile als Kommentar kennzeichnen, verwendet man die Zeichenfolge `//`. Alles, was hinter der Zeichenfolge `//` bis zum Ende der Quelltextzeile steht, wird vom Compiler als Kommentar angesehen und ignoriert.

```
int main() // Kommentar
```

- Mehrzeilige Kommentare beginnt man dagegen mit `/*` und schließt sie mit `*/` ab. Oder Sie müssen jede Zeile mit `//` beginnen.

```
/* Kommentar  
über mehrere  
Zeilen */
```



Kommentare werden vom Compiler ignoriert, d. h., er löscht sie, bevor er den Quelltext in Maschinencode umwandelt. Sparsam veranlagte Leser brauchen sich also keine Sorgen darüber zu machen, dass eine ausführliche Kommentierung die Größe der ausführbaren Programmdatei aufplustern könnte.

Sinnvolles Kommentieren

So einfache Programme, wie wir sie am Anfang dieses Buchs erstellen, bedürfen im Grunde keiner Kommentierung. Kommentare sind nicht dazu gedacht, einem Programmieranfänger C++ zu erklären. Kommentare sollen gestandenen C++-Programmierern helfen, sich in einen Quelltext einzudenken und diesen zu erklären. Kommentare sollten daher eher kurz und informativ sein. Kommentieren Sie beispielsweise die Verwendung wichtiger Variablen (siehe nachfolgendes Kapitel) sowie die Aufgabe größerer Anweisungsabschnitte. Einfache Anweisungen oder leicht zu verstehende Konstruktionen sollten nicht kommentiert werden.

■ 2.2 Programmerstellung

Um aus dem Programmquelltext *HalloWelt.cpp* ein ausführbares Programm zu erzeugen, müssen wir den Quelltext mit Hilfe des C++-Compilers in Maschinencode übersetzen.

Wie Sie dabei vorgehen, hängt davon ab, welche Entwicklungsumgebung Sie verwenden. Zwei Entwicklungsumgebungen möchte ich Ihnen im Folgenden vorstellen: die Visual-Studio-Community-Edition für Windows-Desktop und den GNU-Compiler für Linux.

2.2.1 Programmerstellung mit Visual Studio

Wenn Sie mit Visual Studio arbeiten, steht Ihnen eine komplette, leistungsfähige Entwicklungsumgebung zur Verfügung. Viele Leser werden die Arbeit mit der grafischen Benutzeroberfläche von Visual Studio als angenehmer empfinden als die Arbeit mit einem Compiler, der von der Konsole aus bedient wird.

Allerdings fällt bei der Arbeit mit einer Entwicklungsumgebung etwas mehr Verwaltungsarbeit an. Zum Beispiel verwaltet Visual Studio alle Dateien und Daten, die zu einem Pro-

gramm gehören, in Form eines Projekts. Der erste Schritt bei der Programmentwicklung mit Visual Studio besteht daher darin, ein passendes Projekt anzulegen.

Projekt anlegen

1. Rufen Sie Visual Studio auf. Sie können das Programm z. B. über die Programmgruppe auswählen (unter Windows 7 zu finden im Programme-Ordner des Start-Menüs, bei Windows 10 als Kachel auf der Startseite oder in der App-Ansicht) oder suchen Sie mit der Systemsuche (Suchfeld im Start-Menü für Windows 7, Suchfeld in der Taskleiste für Windows 10) nach **Visual Studio**.

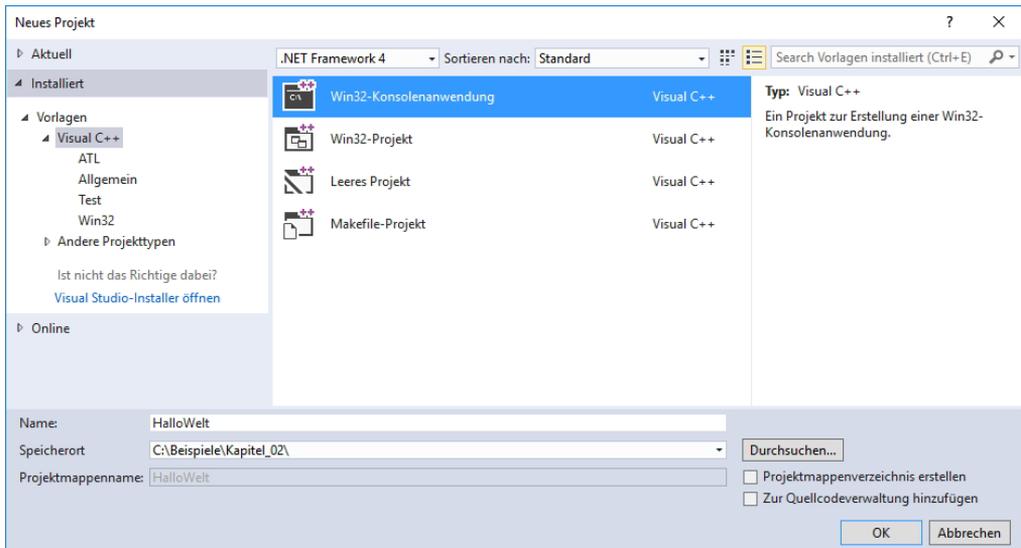


Bild 2.2 Anlegen eines neuen C++-Projekts mit Visual Studio

2. Legen Sie ein neues Projekt an. Rufen Sie dazu den Befehl **Datei/Neu/Projekt** auf.

- Achten Sie darauf, dass links im Dialogfeld **Neues Projekt** die Vorlagenkategorie **Visual C++** ausgewählt ist. Falls nicht, klicken Sie einfach im linken Teilfenster auf den gleichnamigen Eintrag. Wählen Sie dann im mittleren Fenster die Vorlage **Win32-Konsolenanwendung** aus.
- Geben Sie einen **Namen** für das Projekt ein, beispielsweise **HalloWelt**, und wählen Sie im Feld **Speicherort** ein übergeordnetes Verzeichnis für das Projekt aus. (Visual Studio wird unter diesem Verzeichnis ein Unterverzeichnis für das Projekt anlegen, das den gleichen Namen wie das Projekt trägt.)
- Achten Sie darauf, dass die Option **Projektmappenverzeichnis erstellen** deaktiviert ist.
- Drücken Sie zuletzt auf **OK**.

3. Klicken Sie auf der ersten Seite des aufspringenden Assistenten auf **Weiter**.

4. Deaktivieren Sie auf der zweiten Seite die Option **Vorkompilierter Header** und aktivieren Sie dafür die Option **Leeres Projekt**. Klicken Sie auf **Fertig stellen**.

Wenn Sie die Option **Leeres Projekt** nicht aktivieren, legt Visual Studio für Sie eine `.cpp`-Quelltextdatei mit einem einfachen Programmgerüst an. Wir verzichten allerdings auf dieses Programmgerüst, da es a) nur wenig Arbeitserleichterung bringt und b) kein standardisiertes C++ verwendet.

Vorkompilierte Header dienen dazu, die Programmerstellung zu beschleunigen. Sie werden bei der ersten Kompilierung erstellt und können nachfolgende Kompilierungen beschleunigen. Wenn Sie an größeren Projekten arbeiten, ist dies eine recht nützliche Option. Für unsere kleinen Beispielprogramme können wir allerdings auf den „Header“, der viel Speicherplatz belegt, verzichten.

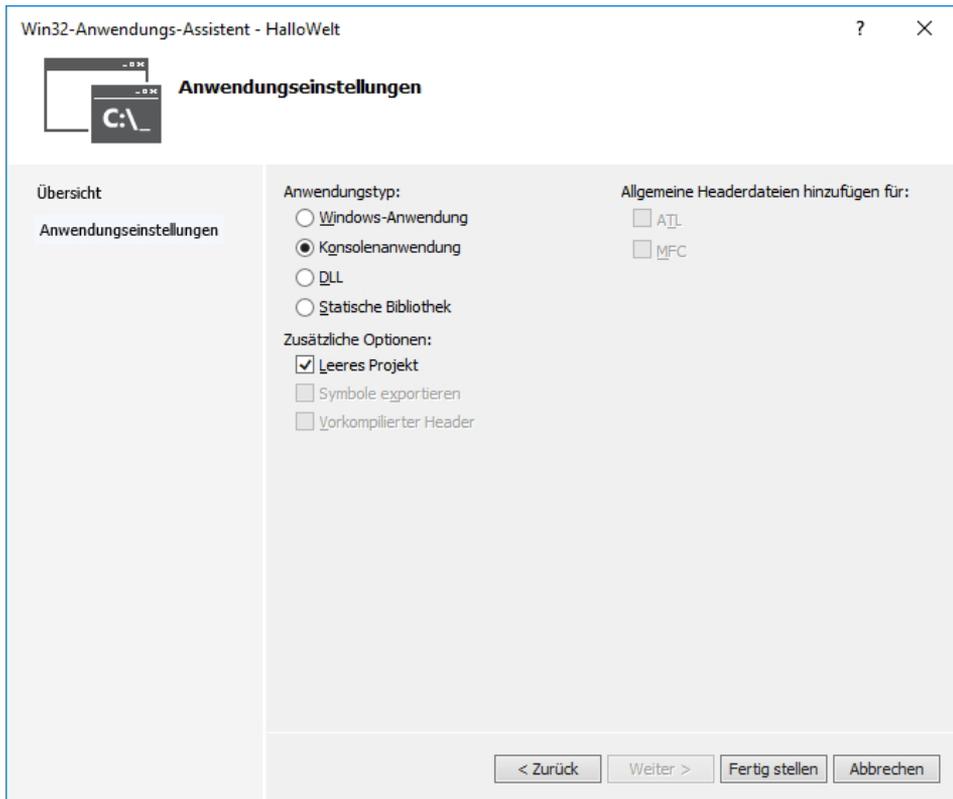


Bild 2.3 Beginnen Sie mit einem leeren Projekt.



Projektmappen

Visual Studio bettet das neue Projekt automatisch in eine Projektmappe ein. Wenn Sie möchten, können Sie über den Befehl **Datei/Neu/Projekt** weitere Projekte in die aktuelle Projektmappe aufnehmen. Sie müssen dann nur im Dialogfenster **neues Projekt** im Listenfeld **Projektmappe** die Option **Hinzufügen** auswählen. Für den Einstieg ist es aber sinnvoller, für jedes neue Programm ein neues Projekt in einer eigenen Projektmappe anzulegen.

Im Projektmappen-Explorer (Aufruf über den gleichnamigen Befehl im Menü **Ansicht**), der standardmäßig links im Visual-Studio-Fenster angezeigt wird, werden alle Projekte der aktuellen Projektmappe zusammen mit den zu den Projekten gehörenden Dateien aufgeführt.

Quelltextdatei hinzufügen

Da wir unsere Arbeit mit einem leeren Projekt begonnen haben, besteht der nächste Schritt darin, dem Projekt eine Quelltextdatei hinzuzufügen.

5. Fügen Sie dem Projekt eine Quelltextdatei hinzu. Klicken Sie dazu mit der rechten Maustaste im Projektmappen-Explorer auf den Projektknoten (in unserem Beispielprojekt ist dies der Knoten mit dem fett dargestellten Namen *HalloWelt*) und rufen Sie im Kontextmenü den Befehl **Hinzufügen/Neues Element** auf.

Falls Sie das Fenster des Projektmappen-Explorers nicht sehen, können Sie es über den Menübefehl **Ansicht/Projektmappen-Explorer** einblenden lassen.

6. Wählen Sie im erscheinenden Dialogfeld die Vorlage *C++-Datei (.cpp)* aus, geben Sie einen Namen für die Datei an und klicken Sie auf **Hinzufügen**.



Die Beispielprogramme der ersten Teile bestehen meist nur aus einer Quelltextdatei, die dann der Einfachheit halber und zur leichteren Identifizierung den Namen des Projekts trägt.

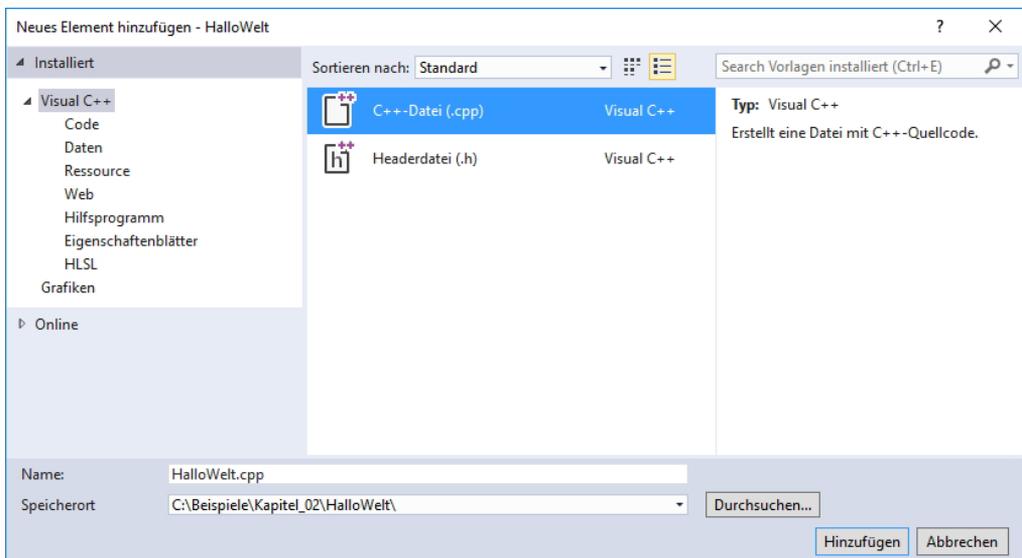


Bild 2.4 Dem Projekt eine Quelltextdatei hinzufügen

Visual Studio legt die neue Quelltextdatei an und lädt sie automatisch in den Editor. Im Projektmappen-Explorer wird die Datei unter dem Knoten **Quelldateien** aufgelistet.

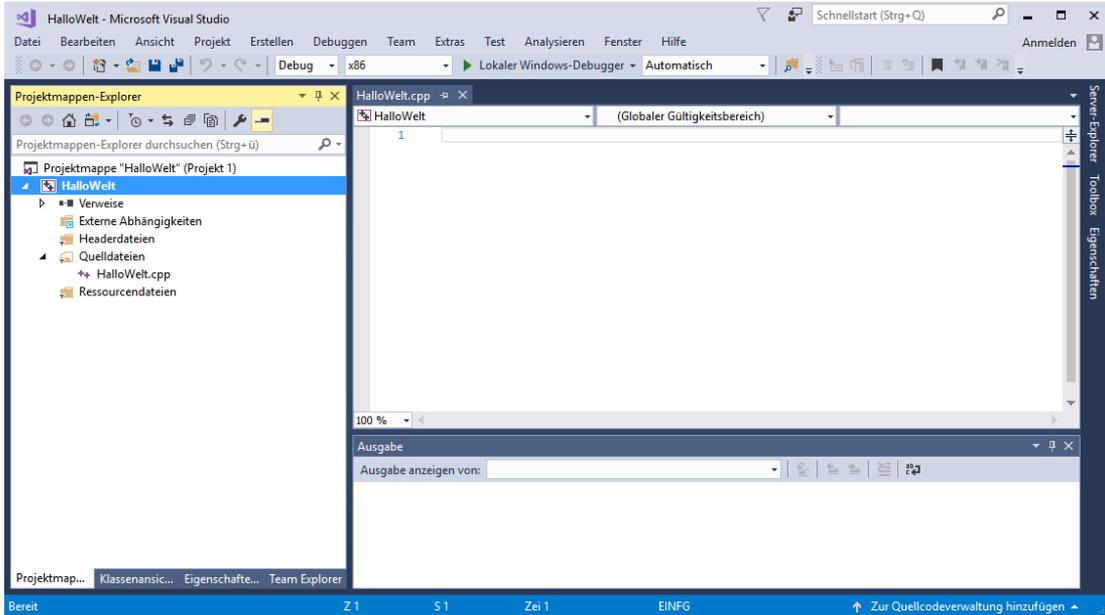


Bild 2.5 Das neue Projekt mit geöffneter, aber noch leerer Quelltextdatei



Wenn Sie im Projektmappen-Explorer auf den Knoten einer Datei doppelklicken, wird die Datei zur Bearbeitung in den Editor geladen.

Quelltext aufsetzen

7. Tippen Sie jetzt den Quelltext aus Listing 2.1 in die Quelltextdatei ein.
8. Speichern Sie zur Sicherheit das Projekt (Befehl **Datei/Alles speichern**).

Projekt kompilieren

Jetzt kommt der große Augenblick. Wir lassen den Quelltext des Programms kompilieren.

9. Um das Programm zu kompilieren und zu erstellen, rufen Sie den Menübefehl **Erstellen/Projektmappe Erstellen** auf.

Im Ausgabefenster, das in den unteren Rand des IDE-Rahmenfensters integriert ist, wird der Fortgang des Erstellungsprozesses angezeigt. Zum Schluss sollte im Ausgabefenster eine Erfolgsmeldung der Form *Erstellen: 1 erfolgreich* und in der Statusleiste die Meldung *Erstellen erfolgreich* zu lesen sein.

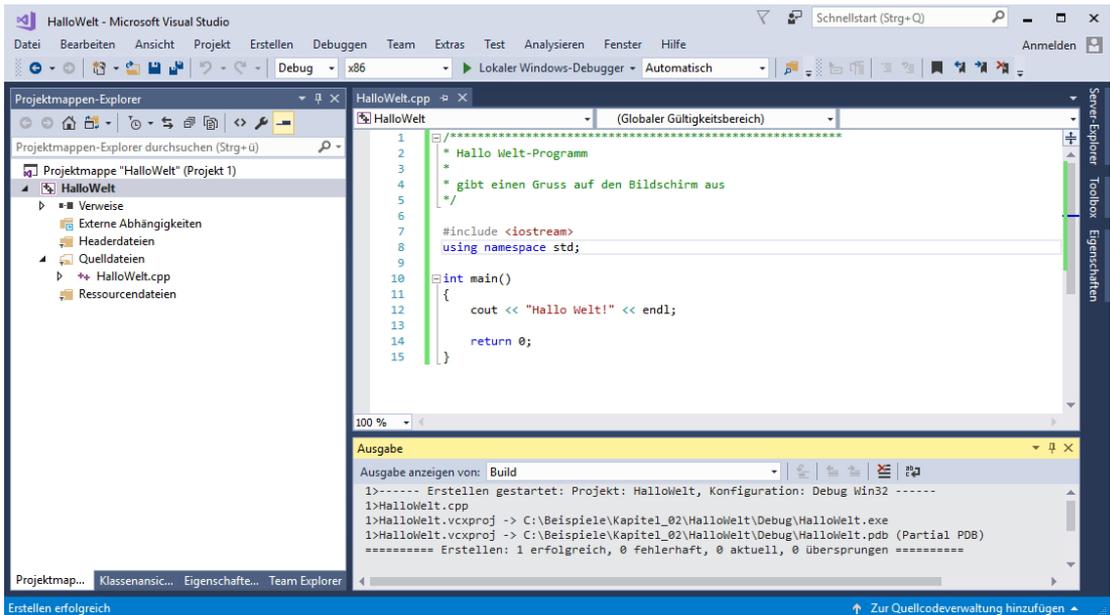


Bild 2.6 Das erfolgreich kompilierte Programm

Sollten bei der Kompilierung Fehler auftreten, gehen Sie so vor, dass Sie die Fehlerliste einblenden lassen (Menübefehl **Ansicht/Fehlerliste**) und dort auf den Reiter **Fehlerliste** klicken. Lesen Sie den Text zu dem ersten Fehler und doppelklicken Sie dann auf die Fehlermeldung, damit Visual Studio die betreffende Stelle im Quelltext markiert. Versuchen Sie, den Fehler zu korrigieren, und erstellen Sie dann das Projekt erneut. Wiederholen Sie diesen Vorgang, bis das Programm erfolgreich erstellt wird.



Manche Fehler erzeugen Folgefehler. Eine fehlende `#include`-Anweisung kann z. B. schnell ein Dutzend oder mehr Fehlermeldungen wegen nicht deklarierter Elemente provozieren.



Kompilieren und Erstellen

Unter *Kompilierung* im engeren Sinne versteht man allein die Übersetzung des Quelltextes in eine Objektdatei. *Erstellen* (engl. build) bedeutet dagegen, dass der Quelltext übersetzt und anschließend vom Linker auch noch mit dem Code weiterer benötigter Objektmodule (Objektdateien anderer Quelltextdateien, Code der Bibliotheken) zu einem ausführbaren Programm zusammengebunden wird. Im allgemeinen Sprachgebrauch wird Kompilieren allerdings oft auch synonym zu Erstellen verwendet.

Programm von Visual Studio aus starten

10. Sie können das Programm direkt von Visual Studio aus starten. Rufen Sie dazu einfach den Befehl **Debuggen/Starten ohne debugging** auf oder drücken und merken Sie sich gleich das Tastaturkürzel **(Strg)+(F5)**.



```
C:\WINDOWS\system32\cmd.exe
Hallo Welt!
Drücken Sie eine beliebige Taste . . .
```

Bild 2.7 Ausführung eines Konsolenprogramms aus der Visual-Studio-IDE heraus

Visual Studio öffnet automatisch ein Konsolenfenster für das Programm und das Programm schickt seine Ausgaben zu diesem Fenster. Als Ergebnis sehen Sie im Konsolenfenster den Gruß Hallo Welt!.



Die Meldung „Drücken Sie eine beliebige Taste ...“ wird von der Visual-Studio-Entwicklungsumgebung hinzugefügt. Sie ist nicht Teil der Anwendung, sondern ein Service der Visual-Studio-IDE, der verhindert, dass das Konsolenfenster gleich wieder verschwindet. Zum Schließen des Fensters drücken Sie eine beliebige Taste.



Achtung!

Wenn Sie zum Ausführen den Befehl **Debuggen/debugging Starten** wählen, schließt sich das Konsolenfenster automatisch nach Beendigung der Programmausführung.

2.2.2 Programmerstellung mit GNU-Compiler

Der GNU-Compiler `g++` bzw. `gcc`¹ ist auf vielen Linux-Systemen standardmäßig installiert.



Im Anhang finden Sie Hinweise, wie Sie testen können, ob der GNU-Compiler auf Ihrem System installiert ist und von wo Sie bei Bedarf eine aktuelle Version herunterladen können.

Quelltexte bearbeiten

1. Legen Sie z. B. mit dem Editor *vi* eine neue Datei namens *HalloWelt.cpp* an, tippen Sie den Quelltext ein und speichern Sie die Datei.

Statt des *vi* können Sie auch jeden beliebigen anderen reinen Texteditor verwenden, beispielsweise den *emacs*, *KEdit* oder *KWrite* unter KDE.

Kompilieren

2. Öffnen Sie ein Konsolenfenster. Wie Ihr Konsolenfenster aussieht und mit welchem Befehl es aufgerufen wird, hängt von Ihrer Linux-Version und dem verwendeten Window-Manager ab. Unter KDE können Sie Konsolenfenster in der Regel über die KDE-Taskleiste aufrufen.

3. Wechseln Sie in der Konsole mit Hilfe des *cd*-Befehls in das Verzeichnis, in dem der Programmquelltext steht.

4. Rufen Sie von der Konsole aus den GNU-Compiler auf. Übergeben Sie dem Compiler in der Kommandozeile den Namen der zu kompilierenden Datei sowie den Schalter *-o* mit dem gewünschten Namen für die ausführbare Datei.

```
g++ HalloWelt.cpp -o HalloWelt
```

oder auch

```
gcc HalloWelt.cpp -o HalloWelt
```

¹ Je nach verwendeter Compilerversion könnte der Compiler auch anders heißen (beispielsweise *egcs*).



Bild 2.8 Kompilation und Erstellung einer ausführbaren Datei

Programm ausführen

5. Tippen Sie in der Konsole den Namen des Programms ein und schicken Sie ab.

Unter Umständen müssen Sie angeben, dass das Programm im aktuellen Verzeichnis zu finden ist. Stellen Sie dazu dem Programmnamen den Punkt als Stellvertreter für das aktuelle Verzeichnis voran: ./HalloWelt.



Bild 2.9 Ausführung von Konsole

2.2.3 Programmausführung

Unabhängig davon, auf welchem Weg Sie Ihr Programm erstellt haben, können Sie die vom C++-Compiler erzeugte ausführbare Programmdatei danach jederzeit aufrufen und ausführen.



Wo befindet sich die ausführbare Programmdatei?

Wenn Sie das Programm mit Visual Studio erstellen, legt die IDE die `.exe`-Datei unter dem Projektverzeichnis in einem Verzeichnis `/Debug` ab.

Wenn Sie das Programm von der Konsole aus mit einem Konsolen-Compiler erstellen, wird die Programmdatei üblicherweise im aktuellen Verzeichnis angelegt.

Sie können Ihren Compiler aber auch so konfigurieren, dass er die Programmdatei in ein anderes Ausgabeverzeichnis schreibt.

Es gibt verschiedene Wege, ein Programm auszuführen, und alle führen letzten Endes über den Aufruf der ausführbaren Programmdatei (unter Windows ist dies die `.exe`-Datei). Dieser kann beispielsweise durch Doppelklick in einem Dateimanager, über Doppelklick auf eine zuvor angelegte Desktop-Verknüpfung oder aber auch aus einem Konsolenfenster heraus erfolgen.

Start aus Konsolenfenster

Für Konsolenanwendungen ist der Start aus einem zuvor explizit geöffneten Konsolenfenster nahezu ideal, da die Anwendung dann genau dieses Fenster für seine Ausgaben benutzt.

1. Öffnen Sie ein Konsolenfenster.

Unter Windows heißt die Konsole Eingabeaufforderung oder auch MS-DOS-Eingabeaufforderung und kann je nach Betriebssystem über die Suche, das **(WIN)+(X)**-Menü, **Start/Alle Programme/Zubehör** oder **Start/Programme** aufgerufen werden.

2. Wechseln Sie in der Konsole in das Verzeichnis mit der ausführbaren Programmdatei.

Zum Wechseln des Verzeichnisses gibt es die Befehle `cd Verzeichnis` und `..` (übergeordnetes Verzeichnis). Den Inhalt des aktuellen Verzeichnisses können Sie zur Kontrolle mit `dir` (Windows) oder `ls` (Linux) auflisten lassen. Das Laufwerk können Sie durch Eingabe des Laufwerksnamens wechseln (beispielsweise `c:`). Ein kleines Tutorium zur Bedienung der Windows-Konsole finden Sie im Übrigen unter www.carpelibrum.de.

3. Starten Sie das Programm, indem Sie den Programmnamen eintippen und abschicken (`Enter`).

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\Dirk> cd C:\Beispiele\Kapitel_02\HalloWelt\Debug
C:\Beispiele\Kapitel_02\HalloWelt\Debug>HalloWelt
Hallo Welt!
C:\Beispiele\Kapitel_02\HalloWelt\Debug>
```

Bild 2.10 Start eines Programms von der Windows-8-Konsole

Start aus Dateimanager oder über Verknüpfung

Sie können Konsolenanwendungen aber auch per Doppelklick aus einem Dateimanager (z.B. Windows Explorer) heraus aufrufen. Allerdings werden Sie dann unter Umständen nicht allzu viel von Ihrem Programm sehen. Dies liegt daran, dass das Programm seine Ausgabe in ein Konsolenfenster schreiben möchte. Wird das Programm nicht aus einem Konsolenfenster heraus aufgerufen, wird die Konsole für die Ausgabe automatisch vom Betriebssystem bereitgestellt. Sie wird aber auch automatisch geschlossen, wenn das Programm beendet ist. Da unser kleines Programm direkt nach der Ausgabe des „Hallo Welt“-Grußes schon beendet ist, sieht man das Konsolenfenster nur kurz aufflackern.

Damit das Programm auch sinnvoll aus einem Dateimanager oder über eine Verknüpfung aufgerufen werden kann, müssen Sie am Ende des Programms, jedoch vor der abschließenden `return`-Anweisung, die Zeile `cin.get();` einfügen:

```
/******  
 * Hallo Welt-Programm  
 *  
 * gibt einen Gruss auf den Bildschirm aus  
 */  
  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    cout << "Hallo Welt!" << endl;  
  
    cin.get();  
  
    return 0;  
}
```

Das Programm wartet dann, bis der Anwender die (Enter)-Taste drückt – und mit ihm wartet das vom Betriebssystem bereitgestellte Konsolenfenster.

■ 2.3 Stil

Zum Abschluss noch ein Wort über guten und schlechten Stil.

C++ besitzt eine ziemlich kryptische Syntax, die daraus resultiert, dass es viele bedeutungstragende Syntaxelemente gibt, die durch einzelne Zeichen dargestellt werden (`{`, `(`, `++`, `%`, `.` etc.), und dass sich die einzelnen Syntaxelemente zu den merkwürdigsten Konstrukten verbinden lassen.

Zwar kann man den sich daraus ergebenden Quelltexten eine gewisse asketische Eleganz kaum absprechen, doch trägt dies weder zur Lesbarkeit noch zur Wartbarkeit der Programme bei – was umso schwerer wiegt, als ein Tippfehler in nur einem Zeichen in C++ schnell zu einem katastrophalen Fehler führen kann.

Tun Sie also Ihr Bestes, um den Quelltext übersichtlich und gut lesbar zu gestalten:

- Schreiben Sie möglichst nur eine Anweisung in eine Zeile.
- Rücken Sie Anweisungsblöcke ein.
- Trennen Sie die einzelnen Elemente durch Leerzeichen.
- Kommentieren Sie Ihren Code.



Zwischen Namen und Schlüsselwörtern müssen Leerzeichen stehen. Sie können also nicht `intmain()` schreiben. Zwischen Namen und Operatoren müssen keine Leerzeichen stehen.

Mit einem Wort: Achten Sie also darauf, dass Ihre Quelltexte wie in Listing 2.1 aussehen und nicht etwa wie folgt:

```
#include <iostream>
using namespace std;
int main(){cout<<"Hallo Welt!"<<endl;cin.get();return 0;}
```

■ 2.4 Übungen

1. Formulieren Sie das Hallo-Welt-Programm ohne die `using`-Anweisung zur Einbindung des `std`-Namensraums. (**Tipp:** Lesen Sie vielleicht noch einmal die Erläuterungen zu den Namensräumen in den Kapiteln 2.1.4 und 1.3.4.)
2. Schreiben Sie das Hallo-Welt-Programm so um, dass es Sie mit Ihrem Namen begrüßt.
3. Räumen Sie Ihre Projektverzeichnisse auf. Compiler legen meist mehr oder weniger umfangreiche Zwischendateien an, die nach Abschluss der Programmentwicklung nicht mehr benötigt werden und möglicherweise nicht automatisch gelöscht werden. Hierzu gehören grundsätzlich die `.obj`-Dateien und speziell für die Visual-Studio-Community-Edition auch die besonders umfangreichen `.idb`-, `.ilk`- und `.pdb`-Dateien (Letztere finden Sie im Verzeichnis der `.exe`-Datei). Zum Löschen dieser Dateien können Sie den Menübefehl **Erstellen/Projektmappe bereinigen** aufrufen. Werfen Sie danach auch noch einen Blick in das Projektverzeichnis. Gibt es dort eine `.sdf`-Datei und ein `ipch`-Verzeichnis? Dann löschen Sie diese.

Index

Symbole

+ (Addition) 66, 380
+ (Addition, string) 73, 200, 202
& (Adresse) 154
~ (bitweises Komplement) 362
| (bitweises ODER) 362
& (bitweises UND) 362
^ (bitweises XOR) 362
– (Dekrement) 71
/ (Division) 67
#endif 261
" Exceptions 345
== (Gleichheit) 77
== (Gleichheit, string) 203
> (Größer) 77
>= (Größer oder gleich) 77
>= (Größer oder string) 203
>= (Größer, string) 203
>> (Eingabe, cin) 216, 381
>> (Eingabe, istream) 216
>> (Eingabe, stringstream) 206
>> (Rechtsverschiebung) 362
#ifndef 261
#include 15
[] (Indizierung, string) 205
[] (Indizierung, vector) 245
++ (Inkrement) 71, 379
++ (Inkrement, Iteratoren) 248
" Instanzen 272
< (Kleiner) 77
<= (Kleiner oder gleich) 77
// (Kommentar, einzeilig) 27
/* (Kommentar, mehrzeilig) 28
! (logisches NICHT) 79
|| (logisches ODER) 79

&& (logisches UND) 78
<= (Kleiner oder gleich, string) 203
< (Kleiner, string) 203
<< (Ausgabe, cout) 209, 381
<< (Ausgabe, ostream) 209
<< (Ausgabe, stringstream) 206
<< (Linksverschiebung) 362
% (Modulo) 67
* (Multiplikation) 67
#pragma once 281
- (Subtraktion) 67
!= (Ungleichheit) 77
!= (Ungleichheit, string) 203
- (Vorzeichen) 66
+ (Vorzeichen) 66

A

Ablaufsteuerung 75
Ableitung 311
Adressen
– in Zeigern speichern 153
– von Variablen 154
Algorithmen (STL) 249
– generate() 252
Algorithmus 11
Anweisungen 24
Arbeitsspeicher 51
argc 120
Argumente
– aus Befehlszeile 121
argv 120
Arrays 131, 242
– als Parameter 137
– Anzahl Elemente 132, 134
– array-Container (C++11) 240

- Definition 132
- Indizierung 133
- Initialisierung 133
- mehrdimensionale 137
- sortieren 253
- Speicherverwaltung 133
- Typ 132
- Variable 132
- von Basisklassentypen 336
- von Strukturen 146
- asctime() 231
- Assembler 5
- Aufzählungen 138
 - Definition 141
 - enum 138
 - enum class (C++11) 142
 - in switch-Verzweigungen 142
- Ausdrücke 69
 - einfache 69
 - Klammern 70
 - komplexe 70
- Ausgabe 209
 - cout 25, 209
 - Formatierung 210
 - Genauigkeit von Gleitkommazahlen 211
 - in Binärdateien 226
 - in Textdateien 221
- Ausnahmen
 - cin 219
- auto (C++11) 56, 247

B

- Backslash-Zeichen 198
- bad() (basic_ios) 218
- badbit (ios_base) 217
- basic_ios (Klasse)
 - bad() 218
 - clear() 218
 - eof() 218
 - fail() 218
 - good() 218
 - rdstate() 218
- Basisklassen
 - abstrakte 341
 - -arrays 336
 - -konstruktoren 320 f.
 - -parameter 338

- Basisklassenunterobjekte 305, 316
 - Initialisierung 320
 - Zugriff 317
- Bedingte Kompilierung 261, 372
- Bedingungen 75
- Beenden
 - Funktionen 116
- Befehlszeilenargumente 121
- begin() (Container) 246
- Beispiele 433, 436
- Benutzereingaben
 - einlesen (cin) 216
- Bezeichner
 - Groß- und Kleinschreibung 46
 - Regeln für die Namensgebung 46
 - vollqualifizierte 16
- Bibliotheken
 - C++-Standardbibliothek 15
 - Pfade 185
 - verwenden 185
- Bildschirmausgaben 25
- Binärikodierung 363
- Binärsystem 52
- Binärzahlen 42
- Bindung 333
 - frühe 333
 - späte 334
- Bit 52
- Bit-Manipulation 362
 - Division 364
 - Flags 365
 - gerade Zahlen 365
 - Klein- und Großschreibung 364
 - Multiplikation 363
- Bogenmaß 192
- bool 55, 77, 446
- Boolescher Typ (bool) 77
- break 85, 99

C

- C 7
- C++ 8
 - Compiler 19
 - Headerdateien 15
 - Programmgerüst 21
 - Standardbibliothek 15
 - Syntaxreferenz 445

- C++11
 - array-Container 240
 - auto 56, 247, 409
 - constexpr 371
 - enum class-Aufzählungen 142
 - for-Schleife 136
 - Initialisierung 48, 61, 134
 - Lambda-Ausdrücke 409
 - long long 55
 - override 332
 - reguläre Ausdrücke 401
 - String-Literale 200
 - Zeitnahme 230, 236
 - Zufallszahlen 194
 - C++14
 - constexpr 372
 - C++17
 - bedingte Kompilierung mit constexpr 372
 - fallthrough 88
 - Variablendefinition in if und switch 372
 - call by reference 160, 163
 - call by value 118
 - capacity() (vector) 244
 - case 85
 - catch 348
 - char 55, 446
 - cin 58, 216
 - >> 216
 - Ausnahmen 219
 - clear() 218
 - Fehlerbehandlung 217
 - get() 244
 - peek() 244
 - class 275
 - clear() (basic_ios) 218
 - clear() (cin) 218
 - clear() (string) 205
 - clear() (stringstream) 206
 - clock() 235
 - compare() (string) 203
 - Compiler
 - Compiler-Schalter 261
 - Fehlermeldungen 174
 - GNU g++ 35
 - Headerdateien 15
 - Übersetzungseinheit 261
 - Visual Studio 28
 - const 50
 - Parameter 164
 - Referenzen 163
 - Variablen 50
 - Zeiger 163
 - const_cast 62
 - constexpr (C++11) 371
 - Container 239
 - [] 245
 - array (C++11) 240, 461
 - deque 251
 - erzeugen 243
 - Größenmanagement 244
 - Initialisierungslisten (C++11) 244
 - Kapazität 244
 - list 247, 255
 - Memberfunktionen 245
 - vector 243
 - continue 99
 - cout 25, 58, 209
 - << 209
 - Fehlerbehandlung 217
 - fill() 211
 - precision() 211
 - width() 210
 - c_str() (string) 205
- ## D
- dangling else 104
 - Dateien
 - Binärdateien 220, 226
 - kopieren 228
 - Textdateien 220
 - Datenstrukturen
 - Arrays 242
 - Hashtabellen 254,
 - Listen 242
 - Mengen 242
 - Stapel 242
 - Warteschlangen 242
 - Datentypen 44, 51
 - Arrays 131
 - Aufzählungen 138, 142
 - auto (C++11) 56, 247, 409
 - Bedeutung 51
 - bool 55, 446
 - char 55, 446
 - Darstellung im Arbeitsspeicher 51
 - double 55, 446
 - elementare 45, 55

- float 55, 446
- int 55, 359, 446
- Klassen 148
- long 55, 446
- long long (C++11) 55, 359, 446
- Operandenkonvertierung 361
- short 55, 359, 446
- Strukturen 143
- typedef 247
- Typumwandlung 57
- Überlaufverhalten 360
- Übersicht 57
- unsigned 359
- Datumsanzeige 229
- Debuggen 179
 - Visual Studio 179
 - Vorgehensweise 179
- default 85
- Definition
 - Arrays 132
 - Aufzählungen 141
 - auto (C++11) 56, 247
 - Referenzen 159
 - Strukturen 144, 148
 - typedef 247
 - Variablen 44
 - Zeiger 154
- Deklaration
 - Funktionen 112, 258
 - Variablen 260
- Dekrement 71
- delete 166, 307
- delete[] 170
- deprecated 353
- Dereferenzierung 156
- Destruktoren 307
 - Vererbung 325
 - virtuelle 325
- difftime() 236
- Divide-and-Conquer-Technik 175
- Division 67
- do 93
- double 55, 446
- do-while 93
- dynamic_cast 62, 339
- Dynamische Speicherreservierung
 - 165
 - Heap 165
 - new 166

- new[] 170
- Tipps 170

E

- Eingabe 216
 - aus Binärdateien 226
 - aus Textdateien 223
 - cin 216
 - Vorschau 244
- Eingabeaufforderung 25, 37
- Eingabetaste 38
- Eintrittsfunktion main() 23, 120
- Ein- und Ausgabe
 - cin 58, 216
 - cout 58, 209, 216
 - Fehlerbehandlung 217
 - printf() 212
 - else 82
 - empty() (Container) 246
 - end() (Container) 246
 - endl 198
 - Endlosschleifen, gewollte 100
 - Entscheidungen 75
 - enum 141
 - eof() (basic_ios) 218
 - eofbit (ios_base) 217
 - erase() (Container) 246
 - erase() (string) 205
 - Ersatzkonstruktor 304
 - Escape-Sequenzen 198
 - exception (Klasse) 350, 352
 - exceptions() (cin) 347
 - Exponentialschreibweise 43

F

- fail() (basic_ios) 218
- failbit (ios_base) 217
- fail() (cin) 347
- failure (ios_base) 219
- Fakultät 370
- fallthrough (C++17) 88
- false 55
- Fehlerbehandlung
 - cin 217
 - cout 217
- Fehlermeldungen 174
- fill() (ostream) 211

find_first_of() (string) 205
 find_last_of() (string) 205
 find() (string) 205
 float 55, 446
 for 95, 136
 friend 377
 Function overhead 127, 295
 Funktionen 109

- Arrays als Parameter 137
- Aufruf 110
- aufrufen 117
- beenden 116
- call by reference 160
- call by value 118
- constexpr 371
- Definition 110
- Deklaration 112, 258
- Function overhead 127
- generische 396
- main() 23, 120
- mathematische 68, 189
- Modularisierung 258
- Ort der Definition 111
- Parameter 117
- Rekursion 369
- return 115
- Rückgabewert 115
- Signatur 128, 367
- trigonometrische 192
- über Dateigrenzen hinweg verwenden 258
- Überladung 128
- Überschreibung 325, 332
- Vorgabeargumente 119
- Zeiger auf 367

G

Ganzzahlen

- Division 67
- Literale 42
- Modulo 67
- Speicherverwaltung 52

 generate() (Algorithmen) 252
 Geschichte, der Programmiersprachen 4
 get() (cin) 244
 get() (ifstream) 225
 getline() (ifstream) 225
 Gleitkommazahlen

- Division 67

- Exponentialschreibweise 43
- Genauigkeit 211
- Literale 43
- Modulo 68
- Speicherverwaltung 53

 gmtime() 230
 good() (basic_ios) 218
 goodbit (ios_base) 217
 goto 102
 Greenwich Mean Time 230
 Groß- und Kleinschreibung 46
 Gültigkeitsbereiche 124

- Block 391
- Datei 123, 391
- Funktion 122, 391
- Klasse 391
- Namensraum 392
- Sichtbarkeit 125
- Verdeckung 125, 392

H

Hashtabellen 254,
 Headerdateien 15

- #include 15
- der Standardbibliothek 15
- einkopieren 15
- Mehrfachaufrufe verhindern 261

 Heap 165
 Hexadezimalzahlen 42
 Hilfe 174

- bei Laufzeitfehlern 179
- bei Problemen 175
- Compiler-Meldungen 174
- Fehlermeldungen 174
- zu Bibliothekselementen 176

 Hollerith, Hermann 4

I

if 80, 372
 if-Anweisung 80, 84
 if-else-Anweisung 82, 372
 Initialisierung

- Arrays 133
- Basisklassenunterobjekte 320
- C++11 48, 61, 134
- Membervariablen (C++11) 332
- Referenzen 159

- Strukturen 146
- Variablen 48
- Zeiger 154
- Inkrement 71
- inline 282, 295
- insert() (Container) 246
- insert() (string) 205
- Instanziierung 303,
- Instanzvariablen 284
- int 55, 359, 446
- Integer 42
- Integral Promotion 361
- ios_base (Klasse) 217
 - badbit 217
 - eofbit 217
 - failbit 217
 - failure 219
 - goodbit 217
- ISO-Standard 19
- istream (Klasse) 216
 - >> 216
- Iteratoren 246

K

- KDE 25
- Klammern
 - in Ausdrücken 70
 - runde 70
- Klassen 148, 267
 - abgeleitete 311
 - abstrakte 341
 - abstrakte Basisklassen 341
 - Basisklassen 311
 - Basisklassenunterobjekte 316
 - Destruktoren 325
 - Friends 377
 - Gültigkeitsbereich 391
 - Klassenvariablen 259
 - Komposition 315
 - Konstruktoren 148
 - Kopierkonstruktor 386, 388
 - Memberfunktionen 148
 - Membervariablen 148
 - Modularisierung 258
 - Polymorphie 329
 - Templates 397
 - über Dateigrenzen hinweg verwenden 258
 - Verdeckung 324

- Vererbung 315
- Zuweisungsoperator 386, 388
- Klassenvariablen 259, 293
- Kommentare 27
- Kompilierung
 - Linker 18
 - vorkompilierte Header 30
- Komplexe Zahlen 195
- Komposition 314
- Konkatenation 73, 202
- Konsole 25
- Konsolenanwendungen
 - Ausgabe (cout) 209
 - Eingabe (cin) 216
- Konstanten 41, 292
 - constexpr-Funktionen 371
 - const-Variablen 50
 - Literale 41
 - Strings 197
- Konstruktoren 148, 273, 302
 - Basisklassenkonstruktoren 320
 - Kopierkonstruktor 386, 388
 - Vererbung 320
- Kontrollstrukturen 99
 - for-Schleife 136
 - if-else-Verzweigung 372
 - switch-Verzweigung 372
- Konvertierungen
 - automatische 361
 - Integral Promotion 361
 - Zahlen in Strings 206
 - Zeit in String 231
- Kopieren
 - flache Kopien 386
 - Objekte 386
 - tiefe Kopien 387
- Kopierkonstruktor 386, 388

L

- Lambda-Ausdrücke (C++11) 409
 - auto-Variable 409
 - Parameter 410
 - Rückgabewert 410 f.
 - Syntax 409
 - Zugriff auf umgebende Variable 410
- Laufzeitfehler 179
- Laufzeitmessung 235
- Laufzeitmessung (C++11) 236

- Laufzeitytypidentifizierung 339
 - dynamic_cast 339
 - typeid 341
- Lebensdauer 393
 - Objekte 393
 - Variablen 393
- Leere Anweisungen 102
- length() (string) 205
- Linker 18
- list (Container-Klasse) 247, 255
- Listen 242
- Literale 41
 - Ganzzahlen 42
 - Gleitkommazahlen 43
 - Strings 41, 197
 - Strings (C++11) 200
 - Verwendung 43
- localtime() 230
- Lochkartensysteme 4
- logic_error (Klasse) 352
- long 55, 446
- long long (C++11) 55, 359, 446
- L-Wert 50

M

- main() 23, 120
 - argc 120
 - argv 120
- Mathematische Funktionen 68, 189
- Mehrfachvererbung 326
- Memberfunktionen 148, 294
 - abstrakte 342
 - Basisklassenparameter 338
 - Bindung 333
 - generische 338
 - override (C++11) 332
 - rein virtuelle 342
 - Überladung 324
 - Überschreibung 325
 - virtuelle 334
- Membervariablen 144, 148, 283
 - Klassenvariablen 259
- Mengen 242
- Menüs, für Konsolenanwendungen 86
 - Funktionen 258
 - Klassen 258
 - Typdefinitionen 260

- Modulo 67
- MS-DOS-Eingabeaufforderung 25, 37
- mutable 299

N

- Namen
 - Groß- und Kleinschreibung 46
 - Regeln für die Namensgebung 46
 - vollqualifizierte 16
- Namensgebung 46
- Namenskonflikte 16
 - Verdeckung 324
- Namensräume 16
 - Namenskonflikte 16
 - Standardbibliothek 17
 - std 17
 - vollqualifizierte Namen 16
- namespace 16
- Namespaces 16
- Nebeneffekte 104
- Neue Zeile-Zeichen 198
- new 166
- new[] 170
- noexcept (C++11) 352
- NULL 156
- nullptr 156
- numeric_limits 360

O

- Objekte
 - kopieren 386
 - Lebensdauer 393
 - vergleichen 382
- Objektorientierte Programmierung 267
 - Polymorphie 329
 - Vererbung 311
- Objektorientiertes Denken 268
- Operandenkonvertierung 361
- Operatoren 16 f., 65, 126
 - + (Addition) 66
 - + (Addition, string) 73, 200, 202
 - & (Adresse) 154
 - ^ (bitweises Komplement) 362
 - ~ (bitweises Komplement) 362
 - | (bitweises ODER) 362
 - & (bitweises UND) 362

- (Dekrement) 71
- / (Division) 67
- == (Gleichheit, string) 203
- >= (Größer oder gleich, string) 203
- > (Größer, string) 203
- >> (Eingabe, cin) 216
- >> (Rechtsverschiebung) 362
- >> (Überladung) 381
- > (Überladung) 384
- >= (Überladung) 384
- [] (Indizierung) 133
- [] (Indizierung, vector) 245
- ++ (Inkrement, Iteratoren) 71, 248
- <= (Kleiner oder gleich, string) 203
- < (Kleiner, string) 203
- << (cout) 209
- << (Linksverschiebung) 362
- << (Überladung) 381
- < (Überladung) 384
- <= (Überladung) 384
- % (Modulo) 67
- * (Multiplikation) 67
- != (Überladung) 382
- + (Überladung) 380
- ++ (Überladung) 379
- += (Überladung) 380
- = (Überladung) 388
- == (Überladung) 382
- - (Subtraktion) 67
- != (Ungleichheit, string) 203
- - (Vorzeichen) 66
- + (Vorzeichen) 66
- arithmetische 65
- Bit-Manipulation 362
- const_cast 62
- Dekrement 71
- delete 166
- delete[] 170
- Dereferenzierung 156
- Division 67
- dynamic_cast 62, 339
- Gültigkeitsbereichsoperator 16 f., 126
- Inkrement 71
- kombinierte Zuweisungen 71
- Modulo 67
- new 166
- new[] 170
- Priorität 70
- reinterpret_cast 62

- sizeof 135
- static_cast 62
- typeid 341
- Typumwandlung 61
- Überladung 378
- Optimierung 181
 - Compiler 182
 - Laufzeitmessung 235
- ostream (Klasse) 209
 - << 209
 - fill() 211
 - precision() 211
 - width() 210
- override (C++11) 332

P

- pair (Klasse) 256
- Parameter 117
 - call by reference 160, 163
 - call by value 118
 - const 164
 - definieren 117
 - Referenzen 160
 - von Basisklassentypen 338
 - Vorgabeargumente 119
 - Zeiger 160
- Passwortabfragen 81
- Passwörter 82
- peek() (cin) 244
- Polymorphie 329
 - abstrakte Basisklassen 341
 - Basisklassenarrays 336
 - Basisklassenparameter 338
 - Bindung 333
 - Definition 329
 - Grundprinzip 330
 - Typidentifizierung 339
 - Überschreibung 331
- pop_front() (deque) 427
- Präprozessor
 - Compiler-Schalter 261
 - Headerdateien 15
- precision() (ostream) 211
- Primfaktorzerlegung 365
- Primzahlen 366
- printf() 212
- private 276, 288, 323
- Programmausführung 36

- Programme
 - Algorithmus 11
 - auf Drücken der Eingabetaste warten 38
 - Beispielprogramme 436
 - Daten einlesen 216
 - debuggen 179
 - Ergebnisse ausgeben 209
 - Erstellung 28
 - Laufzeitmessung 235
 - main() 120
 - optimieren 181
 - Programmausführung 36
 - Programmgerüst 21
 - Programmerstellung 10, 18, 28
 - Ablauf 10, 18
 - mit dem GNU-Compiler 35
 - mit Visual Studio 28
 - Programmierung
 - Assembler 5
 - Geschichte 4
 - Lochkartensysteme 4
 - Modularisierung 7
 - objektorientierte 8
 - strukturierte 6
 - protected 276, 318, 323
 - public 276, 323
 - push_back() (vector) 245
 - push_front() (deque) 427
- R**
- Radiant 192
 - RAM 44
 - rand() 193
 - rdstate() (basic_ios) 218
 - Rechenoperationen 65
 - Referenzen 159
 - const 163
 - Definition 159
 - Initialisierung 159
 - Parameter 160
 - Reguläre Ausdrücke (C++11) 401
 - Assertionen 405
 - Ersetzen 407
 - Gruppierung 404
 - Musterabgleich 405
 - Quantifizierer 403
 - Suchen 406
 - Syntax 401
 - Zeichenklassen 402
 - reinterpret_cast 62
 - Rekursion 369
 - replace() (string) 205
 - return 102, 115
 - Funktionen verlassen 116
 - Rückgabewert 115
 - rfind() (string) 205
 - RSA 366
 - Rückgabewerte 115
 - R-Wert 50
- S**
- Schleifen 89
 - Arrays durchlaufen 134
 - for 136
 - Schlüssel/Wert-Paare 254
 - Semikolon 26, 93, 103
 - short 55, 359, 446
 - Sichtbarkeit 125
 - Signaltonzeichen 198
 - Signatur 128, 367
 - size() (Container) 246
 - size() (vector) 244
 - sizeof 135
 - Sonderzeichen 198
 - Umlaute 213
 - Sortieren
 - Arrays 253
 - Speichermodelle
 - Heap 165
 - Stack 126
 - statischer Speicher 166
 - Speicherverwaltung
 - Arrays 133
 - Ganzzahlen 52
 - Gleitkommazahlen 53
 - Zeichen 54
 - Sprunganweisungen 97
 - srand() 194
 - Stack 126
 - lokale Variablen 126
 - Standardbibliothek 15, 19, 459
 - ANSI C 459f.
 - cin 216
 - complex 195
 - cout 209

- Dateien 220
 - Datentypen 62
 - Ein- und Ausgabe 209
 - Headerdateien 15
 - ifstream 223
 - Iteratoren 246
 - komplexe Zahlen 195
 - mathemat. Funktionen 189
 - numeric_limits 360
 - ofstream 221
 - rand() 193
 - reguläre Ausdrücke (C++11) 401
 - std-Namensraum 17
 - STL 239
 - Streams 219
 - string 201
 - Strings 197
 - Zeitnahme (C++11) 230
 - Zufallszahlen 193
 - Zufallszahlen (C++11) 194
 - Zugriff auf Elemente 17
 - Standardkonstruktoren 304
 - Stapel 242
 - Starten, von Visual Studio 435
 - static
 - Membervariablen 259
 - static_cast 62
 - Statischer Speicher 166
 - Stil 38
 - STL (Standard Template Library) 239, 460
 - Algorithmen 249
 - Aufbau 239
 - Datenstrukturen 242
 - Iteratoren 246
 - Streams 219
 - strftime() 232
 - string (Klasse) 201
 - != 203
 - [] 205
 - + 202
 - == 203
 - > 203
 - >= 203
 - < 203
 - <= 203
 - clear() 205
 - compare() 203
 - c_str() 205
 - erase() 205
 - find() 205
 - find_first_of() 205
 - find_last_of() 205
 - insert() 205
 - length() 205
 - replace() 205
 - rfind() 205
 - substr() 205
 - Strings 26, 41, 197f.
 - + (Konkatenation) 73, 200
 - C-Strings 206
 - erzeugen 201
 - Escape-Sequenzen 198
 - Klasse string 201
 - konkatenieren 202
 - Literale 41, 197
 - Literale (C++11) 200
 - reguläre Ausdrücke (C++11) 401
 - Tabulatoren 198
 - umbrechen 200
 - Umlaute 213
 - Umwandlung 206
 - vergleichen 202
 - vergleichen (nach dt. Alphabet) 204
 - Zeilenumbruchzeichen 198
 - stringstream (Klasse) 206
 - >> 206
 - << 206
 - clear() 206
 - Stroustrup, Bjarne 8
 - struct 144, 148
 - Strukturen 143
 - Arrays von Strukturen 146
 - Definition 144, 148
 - initialisieren 146
 - kontra Klassen 151
 - Variablen 145
 - Zugriff auf Elemente 146
 - substr() (string) 205
 - switch 85, 372
 - switch-Verzweigung 85, 372
 - Syntaxreferenz 445
- T**
- Tabulatorzeichen 198
 - Tauschproblem 161
 - template 396
 - Templates 186, 395

- Funktionen-Templates 396
- Klassen-Templates 397
- STL 239
- Text 41
- this (Instanzzeiger) 296
- throw 351ff.
- time() 229
- tm (Struktur) 230f.
- Trigonometrische Funktionen 192
- true 55
- try 348
- Typdefinitionen
 - Modularisierung 260
 - über Dateigrenzen hinweg verwenden 260
- typedef 247
- typeid 341
- Typidentifizierung 339
 - dynamic_cast 339
 - typeid 341
- Typumwandlung 57
 - cin 58
 - cout 58
 - dynamic_cast 339
 - Integral Promotion 361
 - Operatoren 61
 - Zahlen 60

U

- Überladung 128
 - != 382
 - + 380
 - ++ 379
 - += 380
 - = 388
 - == 382
 - > 384
 - >= 384
 - >> 381
 - < 384
 - <= 384
 - << 381
 - arithmetische Operatoren 380
 - Funktionen 128
 - Memberfunktionen 324
 - Namensauflösung 129
 - Operatoren 378
 - Streamoperatoren 381

- Vergleichsoperatoren 382, 384
- Zuweisungsoperator 388
- Überlaufverhalten 360
- Überschreibung 325, 331f.
- Übersetzungseinheit 18, 261
- Umlaute 213
- unsigned 359
- using 17

V

- Variablen 44
 - const 50
 - Darstellung im Arbeitsspeicher 51
 - Datentypen 44, 51, 55
 - definieren 44
 - Deklaration 260
 - globale 123
 - Gültigkeitsbereiche 124
 - initialisieren 48
 - Lebensdauer 393
 - lokale 122
 - Typumwandlung 57
 - über Dateigrenzen hinweg verwenden 259
 - Werte abfragen 49
 - Werte zuweisen 47
- vector (Container-Klasse) 243
- Verdeckung 125, 324, 392
- Vererbung
 - abgeleitete Klassen 311
 - abstrakte Basisklassen 341
 - Basisklassen 311
 - Basisklassenunterobjekte 316
 - Destruktor 325
 - Grundprinzip 311
 - Konstruktoren 320
 - kontra Komposition 314
 - Mehrfachvererbung 326
 - private Elemente 429
 - Sinn 314
 - Syntax 313
 - Überladung 324
 - Überschreibung 325, 332
 - Verdeckung 324
 - Zugriffsspezifizierer 318, 322
- Vergleiche
 - Objekte 382
 - Strings 202
 - Strings (nach dt. Alphabet) 204

- Verschlüsselungsverfahren 366
- Verzweigungen 80
 - if-else-Verzweigung 372
 - switch-Verzweigung 372
- Virtual 334f., 342
 - Destruktor 325
 - rein virtuelle Memberfunktionen 342
 - Überschreibung 325
- Visual Studio 28, 433
 - Befehlszeilenargumente 121
 - Installation 433
 - Lizenzierung 433
 - Programme ausführen 34
 - Programme erstellen 32
 - Projekte anlegen 29
 - Projektmappen 30
 - Quelldateien 31
 - starten 435
 - Umgebungseinstellungen 435
- void 116
- Vorgabeargumente 119
- Vorkompilierte Header 30
- Vorzeichen 66

W

- Warteschlangen 242
- Warteschleifen 237, 426
- what() (exception) 351
- while 89
- Whitespace 225
- width() (ostream) 210
- Windows-Konsole (Eingabeaufforderung) 25, 37
- write() 226

Z

- Zahlen
 - Binärzahlen 42
 - Division 67
 - Ganzzahlen 42, 52
 - Gleitkommazahlen 43, 53, 211
 - Hexadezimalzahlen 42
 - komplexe 195
 - mathematischen Funktionen 68, 189
 - Modulo 67
 - Rechenoperationen 65
 - trigonometrische Funktionen 192
 - Typumwandlung 60
 - Überlaufverhalten 360
 - Umwandlung in Strings 206
 - Vorzeichen 66
 - Zufallszahlen 193
- Zähler 81
- Zeichen 54
- Zeichensätze
 - ANSI 442
 - ASCII 441
 - Unicode 198
- Zeiger 153
 - Arithmetik 158
 - auf Funktionen 367
 - auf Memberfunktionen 375
 - const 163
 - Definition 154
 - Dereferenzierung 156
 - dynamische Speicherreservierung 165
 - Initialisierung 154
 - nullptr (NULL) 156
 - Parameter 160
- Zeilenumbrüche 200
- Zeit und Datum
 - abfragen 229
 - aufschlüsseln 230
 - C++11 230
 - clock() 235
 - Greenwich Mean Time 230
 - in String formatieren 232
 - in String konvertieren 231
 - tm-Struktur 231
 - Warteschleifen 237, 426
 - Zeitspanne 236
- Zeit- und Datumsanzeige 229
- Zeitmessung 235
- Zeitspanne 236
- Zufallszahlen 193
- Zufallszahlen (C++11) 194
- Zugriffsspezifizierer 275
 - bei Vererbung 322
 - friend 377
 - private 323
 - protected 318, 323
 - public 323
 - Vererbung 318
- Zuse, Konrad 5
- Zustand, eines Objekts 284
- Zuweisungsoperator 386, 388