



André Willms

C++

Eine
kompakte Einführung

C++11/14
Code
inside

dpunkt.verlag



André Willms hat bereits während des Studiums der Allgemeinen Informatik mit dem Schreiben von Büchern zum Thema C++ begonnen. Heute ist er Autor mehrerer erfolgreicher Bücher zu C und C++. Hauptberuflich ist er IT-Trainer mit inzwischen 17 Jahren Berufserfahrung.

André Willms

C++: Eine kompakte Einführung



dpunkt.verlag

André Willms
info@andrewillms.de

Lektorat: Christa Preisendanz
Copy-Editing: Ursula Zimpfer, Herrenberg
Herstellung: Frank Heidt
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN:
Buch 978-3-86490-229-1
PDF 978-3-86491-651-9
ePub 978-3-86491-652-6

1. Auflage 2015
Copyright © 2015 dpunkt.verlag GmbH
Wiebinger Weg 17
69123 Heidelberg

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

Inhaltsverzeichnis

1	Einführung	1
1.1	Über das Buch	1
1.2	Vorstellung des Projekts	2
1.3	Identifizieren der Programmteile	3
1.3.1	Objekte	4
1.3.2	Kontrollstrukturen	4
1.4	Abstraktion	4
1.4.1	Datenabstraktion	5
1.4.2	Algorithmische Abstraktion	5
2	Grundelemente eines C++-Programms	9
2.1	Das erste Programm	9
2.1.1	Implizites return	10
2.2	Die Ausgabe	10
2.2.1	cout	11
2.3	include	13
2.4	Namensbereiche	14
2.5	Kommentare	16
2.6	Escape-Sequenzen	18
2.7	Zusammenfassung	18
2.8	Spielprojekt	19
3	Arithmetik in C++	21
3.1	Variablen	21
3.1.1	Integrierte Datentypen	22
3.1.2	Vorzeichenlose Typen	23
3.2	Definieren einer Variablen	23
3.2.1	Initialisierung	24
3.2.2	Lebensdauer	25

3.2.3	Automatische Typbestimmung	25
3.2.4	Definitionen zusammenfassen	26
3.3	Eingabe	27
3.4	Grundrechenarten	28
3.5	Konstanten	30
3.5.1	Konstante Ausdrücke	30
3.5.2	Unveränderliche Werte	31
3.6	Restwert	31
3.7	Verknüpfen unterschiedlicher Datentypen	32
3.8	Explizite Typumwandlung	33
3.9	Kombinierte Zuweisungsoperatoren	34
3.10	Inkrement und Dekrement	35
3.11	Mathematische Funktionen	35
3.12	Bitweise Operatoren	36
3.13	Zusammenfassung	39
3.14	Spielprojekt	40
4	Verzweigungen	43
4.1	Zusammengesetzte Anweisungen	43
4.2	Bedingungen	44
4.3	if	45
4.4	else	47
4.5	Logische Operatoren	48
4.6	Der ?:-Operator	51
4.7	Die Fallunterscheidung	52
4.8	static_assert	55
4.9	assert	55
4.10	Zusammenfassung	57
4.11	Spielprojekt	57
5	Schleifen	59
5.1	while	59
5.2	do-while	61
5.3	for	62
5.4	break	64
5.5	continue	65
5.6	Zusammenfassung	66
5.7	Spielprojekt	66

6	Funktionen	67
6.1	Funktionsdefinition	67
6.2	return	69
6.3	Standardwerte	70
6.4	Suffixrückgabetyt	71
6.5	Funktionsdeklaration	72
6.6	Module	72
6.7	Funktionen überladen	74
6.7.1	Unterscheidung in der Parameteranzahl	74
6.7.2	inline	75
6.7.3	Unterscheidung im Parametertyp	76
6.8	Lambda-Funktionen	76
6.9	Zusammenfassung	79
6.10	Spielprojekt	79
7	Klassen	81
7.1	Objektorientierte Programmierung	81
7.1.1	Objekte als abgrenzbare Einheiten	81
7.1.2	Nicht objektorientierte Objekte	82
7.2	Klassen als Bauplan	83
7.2.1	Definition	83
7.3	Zugriffsrechte	84
7.4	Konstruktoren	85
7.4.1	Standardkonstruktor	87
7.4.2	Der Destruktor	87
7.5	Methoden	88
7.5.1	Zugriffsmethoden	89
7.5.2	Konstante Methoden	89
7.6	Externe Definition	90
7.7	Mehrfachdefinition	92
7.8	Typalias	93
7.8.1	typedef	94
7.8.2	using	95
7.8.3	Zugriffsrecht	95
7.9	cv-Qualifizierung	96
7.10	Zusammenfassung	97
7.11	Spielprojekt	98

8	Arrays und Verweise	101
8.1	Arrays definieren	101
8.1.1	sizeof	102
8.2	Arbeiten mit Arrays	102
8.2.1	Initialisierung	103
8.2.2	Arrays durchlaufen	103
8.3	Arrays als Funktionsparameter I	104
8.4	Zeiger	105
8.4.1	Hexadezimalsystem	105
8.4.2	Der Adressoperator	106
8.4.3	Definition eines Zeigers	107
8.4.4	Dereferenzierung	108
8.4.5	Zeiger als Funktionsparameter	108
8.4.6	Zeiger auf Zeiger	110
8.4.7	Arrays als Funktionsparameter II	111
8.4.8	Zeigerarithmetik	112
8.5	Referenzen	113
8.6	Objekte als Funktionsparameter	115
8.6.1	Referenzen auf Objekte	115
8.6.2	Zeiger auf Objekte	116
8.6.3	Objekte als Methodenparameter	117
8.7	Zusammenfassung	118
8.8	Spielprojekt	118
9	Strings	119
9.1	char	119
9.1.1	cctype	120
9.2	C-Strings	121
9.2.1	cstring	123
9.2.2	Beispiel	123
9.3	Strings	124
9.3.1	Tastatureingabe von Strings	125
9.3.2	Methoden von string	126
9.4	Zusammenfassung	128
9.5	Spielprojekt	128
10	Dynamische Speicherverwaltung	131
10.1	Zeiger	131
10.1.1	Zeiger und Konstanten	132
10.1.2	Zeiger auf Funktionen	133
10.1.3	Zeiger auf Klassenelemente	134

10.2	Referenzen	137
10.3	new und delete	137
10.3.1	Die Klasse Name	139
10.4	Smart Pointer	140
10.4.1	Unique Pointer	140
10.4.2	Shared Pointer	143
10.4.3	Weak Pointer	145
10.4.4	Smart Pointer und Arrays	148
10.4.5	Auto-Pointer	148
10.5	Zusammenfassung	149
10.6	Spielprojekt	149
11	Klassen – Vertiefung	151
11.1	Reihenfolge der Zugriffsrechte	151
11.2	Der this-Zeiger	153
11.3	Konstruktoren	154
11.3.1	Standardkonstruktor	156
11.3.2	Kopierkonstruktor	157
11.3.3	Die Klasse Name	158
11.3.4	Elementinitialisierungsliste	160
11.3.5	Verschiebekonstruktor	163
11.3.6	Implizite Typumwandlung	166
11.3.7	Konstruktordelegation	166
11.4	Destruktoren	167
11.5	Konstante Objekte und Elemente	168
11.5.1	Implizite Objektparameter	168
11.5.2	mutable	170
11.6	Funktionen als deleted oder default definieren	176
11.7	Zusammenfassung	177
11.8	Spielprojekt	177
12	Klassen – Abschluss	181
12.1	Standardwerte für Attribute	181
12.2	Verschachtelte Klassendefinitionen	181
12.3	Statische Klassenelemente	183
12.3.1	Statische Methoden	183
12.3.2	Statische Attribute	186
12.3.3	Statische Variablen	191
12.4	Konstruktoren und ihre Anwendung	192
12.4.1	Funktionsaufruf aus Konstruktoren heraus	192
12.4.2	Unvollendet konstruierte Objekte	194

12.5	Implizite Klassenelemente	196
12.5.1	Impliziter Standardkonstruktor	198
12.5.2	Impliziter Kopierkonstruktor	198
12.5.3	Impliziter Verschiebekonstruktor	199
12.5.4	Impliziter Kopier-Zuweisungsoperator	199
12.5.5	Impliziter Verschiebe-Zuweisungsoperator	200
12.5.6	Impliziter Destruktor	201
12.6	Aufzählungen	201
12.7	Zusammenfassung	202
12.8	Spielprojekt	203
13	Namensbereiche	205
13.1	Deklarative Bereiche, potenzielle und tatsächliche Bezugsrahmen ..	205
13.2	Namensbereiche definieren	208
13.3	Die Using-Direktive	209
13.4	Ein Alias für Namensbereiche	210
13.5	Unbenannte Namensbereiche	211
13.6	Die Using-Deklaration	211
13.7	Zusammenfassung	212
13.8	Spielprojekt	213
14	Operatoren überladen	215
14.1	Zuweisungsoperatoren	215
14.1.1	Kopier-Zuweisungsoperator	215
14.1.2	Verschiebe-Zuweisungsoperator	217
14.1.3	Kombinierte Zuweisungsoperatoren	218
14.2	Rechenoperatoren	219
14.2.1	Operation als Methode	219
14.2.2	Operation als Funktion	219
14.2.3	Methode oder Funktion	222
14.2.4	Operatoren mit Verschiebe-Semantik	223
14.2.5	Standardverhalten nachbilden	226
14.3	Vergleichsoperatoren	228
14.3.1	Operator-Templates	229
14.4	Die Operatoren << und >>	229
14.4.1	operator<<	230
14.4.2	operator>>	230
14.5	Der Operator []	231
14.6	Der Operator ()	232
14.7	Die Operatoren -> und *	233
14.8	Umwandlungsoperatoren	235

14.9	Die Operatoren ++ und --	236
14.9.1	Präfixoperatoren	238
14.9.2	Postfixoperatoren	238
14.9.3	Weitere Operatoren	239
14.10	Probleme mit Operatoren	240
14.11	Zusammenfassung	241
14.12	Spielprojekt	241
15	Templates	243
15.1	Klassen-Templates	243
15.2	Funktions-Templates	245
15.3	Template-Parameter	246
15.3.1	Standardargumente	247
15.4	Template-Spezialisierung	247
15.5	typename	248
15.6	Zusammenfassung	249
16	STL	251
16.1	Die Komponenten der STL	251
16.2	Container	252
16.2.1	Laufzeitklassen	252
16.2.2	Die Container im Überblick	253
16.2.3	STL-konforme Container	254
16.3	Iteratoren	255
16.3.1	Entwurf eines Iterators	255
16.3.2	Ein Iterator für Name	258
16.3.3	Iteratorkategorien	260
16.3.4	STL-konforme Iteratoren erstellen	263
16.3.5	Iteratoren erzeugen	263
16.3.6	Insert-Iteratoren	264
16.3.7	Stream-Iteratoren	267
16.4	Algorithmen	268
16.4.1	Die Algorithmen im Überblick	268
16.5	Die STL im Einsatz	271
16.5.1	Variable Funktionsaufrufe	271
16.5.2	Aufrufübergreifende Zustände	275
16.5.3	Element suchen	276
16.5.4	Element suchen mit eigener Bedingung	277
16.5.5	Elemente löschen	278
16.5.6	Elemente kopieren	278

16.5.7	Elemente sortieren	279
16.5.8	Eigene Listeninitialisierungskonstruktoren	281
16.6	Zusammenfassung	282
16.7	Spielprojekt	282
17	Vererbung I	287
17.1	Das Klassendiagramm der UML	287
17.2	Vererbung in C++	290
17.3	Die Vererbungssyntax	292
17.4	Geschützte Elemente	296
17.4.1	Zugriff auf Basisklassenelemente	298
17.5	Polymorphie	298
17.6	Verdecken von Methoden	300
17.7	Überschreiben von Methoden	302
17.8	Virtuelle Methoden	304
17.8.1	Virtuelle Methoden und Konstruktoren	305
17.8.2	Downcasts	307
17.8.3	Virtuelle Destruktoren	308
17.9	Rein-virtuelle Methoden	311
17.9.1	Rein-virtuelle Methoden mit Implementierung	311
17.9.2	Rein-virtuelle Destruktoren	312
17.10	Vererbung und Arrays	313
17.11	Vererbung und Standardwerte	314
17.12	Vererbung und überladene Operatoren	315
17.13	Versiegelte Elemente	316
17.13.1	Versiegelte Klasse	316
17.13.2	Versiegelte Methode	316
17.13.3	Warum Elemente versiegeln?	317
17.14	Geerbte Konstruktoren verwenden	318
17.15	Überschreibungshilfe	319
17.16	Zusammenfassung	320
17.17	Spielprojekt	320
18	Vererbung II	325
18.1	Beziehungen	325
18.1.1	ist ein	325
18.1.2	ist implementiert mit	330
18.1.3	hat ein	332
18.2	Was wird vererbt?	333
18.2.1	Schnittstelle mit verbindlicher Implementierung	333
18.2.2	Schnittstelle mit überschreibbarer Implementierung	335

18.2.3	Schnittstelle	336
18.2.4	Implementierung	338
18.3	Das Offen-geschlossen-Prinzip	338
18.4	Operationen oben, Daten unten	342
18.5	Das Umkehrung-der-Abhängigkeit-Prinzip	344
18.6	Das Einzelne-Verantwortung-Prinzip	346
18.7	Zusammenfassung	347
18.8	Spielprojekt	348
19	Mehrfachvererbung	353
19.1	Gemeinsame Basisklassen	354
19.2	Virtuelle Basisklassen	356
19.3	Konstruktoren virtueller Basisklassen	358
19.4	Einsatz von Mehrfachvererbung	359
19.5	Zusammenfassung	360
19.6	Spielprojekt	361
20	Ausnahmen	365
20.1	Warum Ausnahmen?	365
20.2	Einsatz von Ausnahmen	367
20.3	Vordefinierte Ausnahmen	370
20.3.1	Header-Datei »exception«	371
20.3.2	Header-Datei »typeinfo«	371
20.3.3	Header-Datei »memory«	371
20.3.4	Header-Datei »new«	371
20.3.5	Header-Datei »stdexcept«	372
20.4	Ausnahmen im Detail	372
20.4.1	terminate	372
20.4.2	Das Verlassen eines Try-Blocks	373
20.4.3	uncaught_exception	374
20.4.4	Das Werfen einer Ausnahme	374
20.4.5	Das Fangen einer Ausnahme	375
20.5	Ausnahmespezifikationen	378
20.5.1	Ausnahmespezifikationen in der Praxis	379
20.6	Ausnahmen und Konstruktoren	379
20.7	Ausnahmen und Destruktoren	381
20.8	Ausnahmen und dynamische Speicherverwaltung	382
20.9	Ressourcenerwerb ist Initialisierung	384
20.10	Funktions-Try-Blöcke	385
20.11	Ausnahmensicherheit	387
20.12	Zusammenfassung	387

21	Das Spiel	389
21.1	Aktionen	389
21.2	Spielregeln	392
21.3	Räume	394
21.4	Die Erzeugung der Spielwelt	396
21.4.1	Erstellen der Räume	396
21.4.2	Erzeugen der Gegenstände	397
21.4.3	Erstellen der Ausgänge	398
21.4.4	Erzeugen von Türen	399
21.4.5	Erstellen der Interaktionen	402
Literatur		405
Index		407

1 Einführung

»Und was kann ich alles programmieren, wenn ich das Buch durchgelesen habe?«

Eine beliebte Frage auch in Schulungen, wenn die Sprache nicht mit einem speziellen Ziel im Hinterkopf gelernt wird, sondern als erster Einstieg in die Welt des Programmierens gewählt wurde und die genaue Reiseroute noch nicht feststeht.

Um diese Frage zu beantworten, geht dieses Buch einen besonderen Weg. Anstatt die einzelnen Elemente der Sprache isoliert zu behandeln und es dem Leser zu überlassen, wo sich das Gelernte in einem größeren Programm wiederfindet, werden wir uns in diesem Kapitel das Ergebnis eines größeren Projekts anschauen und uns dann schrittweise die notwendigen Kenntnisse aneignen, um das Programm verstehen, erweitern und sogar selbst programmieren zu können.

Abgesehen von diesem großen buchumspannenden Projekt gibt es noch Beispiele, die über mehrere Kapitel entwickelt werden. Zusätzlich wird jedes Thema in genügend kleineren Beispielen erläutert, um die Anwendung der Sprachelemente zu verstehen.

1.1 Über das Buch

Bevor ich Ihnen das große Projekt vorstelle, möchte ich noch kurz abgrenzen, um was es in diesem Buch geht und an wen es gerichtet ist.

Zielgruppe

Dieses Buch richtet sich an Einsteiger in die Programmierung sowie Personen, die bereits erste Erfahrungen mit C++ gemacht haben.

Jedes Thema beginnt mit einer einfachen Einführung und vielen Beispielen, die immer weiter ausgebaut werden, bis auch Details von C++ zur Sprache kommen.

Je nach Vorwissen kann Ihnen eventuell die Einführung zu ausführlich oder die Details zu detailliert erscheinen. Das ist überhaupt kein Problem, übersprin-

gen Sie die Abschnitte einfach. Immer, wenn ein bereits im Buch behandeltes Thema an anderer Stelle Anwendung findet, steht ein Querverweis im Text, sodass Sie die tiefer gehenden Abschnitte auch erst bei Bedarf lesen können.

C++14

Das in diesem Buch verwendete C++ entspricht dem C++14-Standard, der bei Drucklegung dieses Buches kurz vor der offiziellen Verabschiedung steht. Die Erweiterungen von C++11 zu C++14 sind aber nur minimal und beziehen sich in vielen Fällen auf fortgeschrittene Themen, die in diesem Buch nicht behandelt werden. Alle Programmcodes im Buch lassen sich sowohl mit einem C++14-Compiler übersetzen als auch mit einem C++11-Compiler. Im Buch wird an vielen Stellen auf den internationalen Standard ISO/IEC 14882-2011 in der Form [C++ #] verwiesen, wobei # für eine (Kapitel-)Nummer steht.

Objektorientierte Programmierung

Natürlich beschäftigt sich das Buch mit der objektorientierten Programmierung (OOP). Nur: Im Gegensatz zu anderen Sprachen wie zum Beispiel Java erzwingt C++ keine objektorientierte Programmierung. Viele Sprachmittel setzen keine Elemente der objektorientierten Programmierung ein oder sind sogar flexibler, wenn sie außerhalb einer Klasse stehen.

Als ehemaliger Java-Programmierer erscheint Ihnen daher die OOP vielleicht zu spät im Buch, als C-Programmierer kommt sie unter Umständen zu früh.

Um das Ganze mit Zahlen zu hinterlegen: Über 40% des Buches handeln ausschließlich von den Mechanismen der Klassen und der Vererbung und weitere 30% setzen diese Mechanismen ein. Trotzdem gebe ich dem Leser die Zeit, die Sprache zunächst mit ihren weniger abstrakten prozeduralen Mechanismen kennenzulernen, bevor wir die höheren Abstraktionsebenen der objektorientierten Programmierung betreten, die in C++ anspruchsvoller sind als in anderen Sprachen.

1.2 Vorstellung des Projekts

Der Mensch lernt am leichtesten und effektivsten spielerisch. Was liegt bei der Erlernung einer Programmiersprache also näher, als ein Computerspiel zu entwerfen?

Die Herausforderung bestand darin, ein Spielkonzept zu entwickeln, das mit reinem ISO-C++ realisiert werden kann, ohne plattformspezifische Bibliotheken einsetzen zu müssen. Grafische Darstellungen fielen damit schon mal weg. Es musste ein Spiel auf Textebene werden. Weil das Genre der Point&Click-Adventure eines meiner liebsten ist, habe ich mich dazu entschieden, ein Spiel im Stile seiner Vorgänger zu programmieren: ein Text-Adventure.

Den gesamten Programmcode des Spiels, zusätzliche Informationen sowie eine plattformunabhängige Java-Version zum direkten Ausprobieren finden Sie auf meiner Homepage unter <http://cpp.andrewillms.de>.

Ziel des Spiels ist es, aus einem Haus herauszukommen. Dazu muss das Haus erkundet, neue Räume erschlossen und Gegenstände eingesammelt und kombiniert werden. Grundsätzlich gilt: Sie können auch verlieren, entweder direkt, weil Sie beispielsweise einen Stromschlag bekommen haben und deshalb ohnmächtig wurden, bis der Hausbesitzer zurückkommt, oder indirekt, weil Sie sich den Weg verbaut haben, zum Beispiel, weil Sie einen notwendigen Gegenstand zerstört, verloren oder erst gar nicht eingesammelt haben. Die vielen unterschiedlichen und mitunter auch skurrilen Arten des Verlierens machen aber einen Teil des Spielspaßes aus. Sie sollten daher Folgendes beachten:

- Regelmäßig Spielstände speichern.
- Alle Gegenstände untersuchen und gegebenenfalls benutzen.
- Auch verschiedene Aktionen probieren. Es macht oft einen Unterschied, ob Gegenstände benutzt, gezogen, gedrückt, geöffnet oder geschlossen werden.
- Erst einmal nichts tun, was Sie auch in Ihrem eigenen Haus unterlassen würden, wie zum Beispiel alles abfackeln. (Es sei denn, es muss sein.)

Um die Programmierung einfach zu halten, ist die Syntax der Eingabemöglichkeiten rudimentär. »gehe treppe«, »verwende haarknäuel + toilette« gelten innerhalb des Spiels als gutes Deutsch. Präpositionen und Artikel sind nicht erlaubt.

Es reicht immer aus, so viel von einem Wort anzugeben, dass es eindeutig ist. Bei den Befehlen können bis auf »Speichern« alle mit nur einem Buchstaben abgekürzt werden. Damit ist »g v« eine gültige Schreibweise für »gehe vor«.

Es ist für das Lesen des Buches nicht notwendig, das Spiel durchgespielt oder überhaupt gespielt zu haben. Es erleichtert aber das Verständnis, wenn Sie die Programmbeispiele den Aktionen im Spiel zuordnen können.

Als grobe Richtschnur gilt: Sie haben einen guten Einblick in das Spiel, wenn Sie es unfallfrei in den Keller geschafft haben. Haben Sie gar den Vorratsraum erreicht, kennen Sie alle programmtechnischen Spielelemente.

1.3 Identifizieren der Programmteile

An dieser Stelle wollen wir das Spiel einer genaueren Betrachtung unterziehen und typische Elemente eines Computerprogramms ausmachen. Diese werden dann in den weiteren Kapiteln im Detail besprochen.

1.3.1 Objekte

Im Spiel springen zwei Elemente direkt ins Auge, die Räume und die darin befindlichen Gegenstände. Solche klar abgegrenzten und unterscheidbaren Entitäten nennt man in der objektorientierten Programmierung Objekte.

Objekte besitzen üblicherweise zu jedem Zeitpunkt einen eindeutigen Zustand. Das Handtuch besitzt zu Beginn den Zustand »In der Toilette befindlich«. Nachdem der Spieler es genommen hat, wechselt der Zustand zu »Im Inventar befindlich«. Das Feuerzeug ist zunächst unsichtbar. Wenn der Spieler die Jacken untersucht, wechselt der Zustand zu »Sichtbar, im Inventar und 25% gefüllt«. Die zustandsbeschreibenden Elemente eines Objekts nennt man Attribute oder Objektdaten.

Die vorangegangenen Beispiele offenbaren eine weitere Eigenschaft von vielen Objekten; die Fähigkeit, ihren Zustand zu verändern. Die zustandsverändernden Elemente eines Objekts werden als Methoden, Nachrichten und speziell in C++ auch als Elementfunktionen bezeichnet.

1.3.2 Kontrollstrukturen

Es reicht aber nicht aus, dass die Objekte einen Zustand besitzen, den sie ändern können. Sie müssen zu dieser Zustandsänderung auch aufgefordert werden, aber nicht unkontrolliert, sondern nur, wenn bestimmte Bedingungen gelten.

Falls der Spieler in der Toilette steht und das Handtuch im Raum liegt und der Spieler »nimm handtuch« als Befehl angibt, genau dann wechselt das Handtuch seinen Zustand zu »Im Inventar befindlich«. Dieses bedingte Abarbeiten von Anweisungen wird in der Programmierung Verzweigung genannt.

Eine andere Form der Kontrollstruktur ist bei der Eingabe des Anwenders involviert. Dort hat der Anwender die Möglichkeit, einen Befehl und betroffene Objekte einzugeben. Gibt er »ende« ein, dann ist das Spiel beendet. In allen anderen Fällen wird der Text zerlegt, der Befehl und die Objekte identifiziert, der Befehl ausgeführt und der Anwender erneut nach einem Befehl gefragt.

Die Anweisung lässt sich so formulieren: Solange der Anwender nicht »ende« eingegeben hat, wird der Befehl ausgeführt und erneut nach einem Befehl gefragt. Diese Art der Kontrollstruktur wird Wiederholung oder Schleife genannt.

1.4 Abstraktion

Ein weiteres Thema bei der Programmierung des Spiels ist der Weg von der Idee zum Programm. In begrenztem Umfang spiegelt das Spiel die reale Welt wider. Genau genommen ist das Spiel eine vereinfachte und abstrahierte Form der Realität.

1.4.1 Datenabstraktion

Bei der Datenabstraktion werden die Attribute, die den Zustand in der realen Welt beschreiben, auf die für das Programm absolut notwendigen Elemente reduziert. So einfach wie möglich, aber nicht einfacher.

Nehmen wir wieder das Handtuch. In der Realität besitzt es eine Breite, eine Länge, eine Dicke und ein Gewicht. Es besitzt eine Farbe, die je nach Qualität nach jedem Waschen mehr oder weniger ausbleicht. Das Handtuch besteht aus 100% Baumwolle mit 50%iger Polyesterbeimischung (frei nach Lorient), es ist mehr oder weniger mit Weichspüler gesättigt und daher weicher oder kratziger. Ich könnte diese Liste bis zum Ende des Buches fortführen.

Im Spiel kann sich das Handtuch nur auf der Toilette befinden oder im Inventar sein, es kann nass oder trocken sein. Mehr muss es nicht können. Die Reduktion aller möglichen Zustände eines realen Handtuchs auf diese beiden Attribute im Spiel nennt man Datenabstraktion.

1.4.2 Algorithmische Abstraktion

Dass die Zustände der Objekte innerhalb des Programms abgebildet werden können, ist aber nur die halbe Miete. Sinnvoll sind Zustände nur, wenn sie sich auch ändern können.

Jedes Programm besitzt einen Startzustand (definiert über die zu Beginn des Programms vorhandenen Objekte und deren Zustände) und einen Endzustand (definiert über die am Ende des Programms vorhandenen Objekte und deren Zustände). Die Regeln, wann und wie sich die Zustände während des Programmlaufs ändern, werden über den Algorithmus definiert.

Nehmen wir exemplarisch die Berechnung des größten gemeinsamen Teilers zweier positiver ganzer Zahlen. Dieser wird unter anderem beim Kürzen von Brüchen verwendet. Der ggT (die Abkürzung für »größter gemeinsamer Teiler«) von 6 und 12 ist beispielsweise 6, der ggT von 6 und 9 ist 3. Konkrete ggT zu bestimmen, fällt nicht sehr schwer.

Schwieriger wird es, eine allgemeingültige Lösung zur Bestimmung des ggT zu formulieren.

Algorithmus

Ein Algorithmus ist eine Menge von Regeln, durch deren Befolgung in festgelegter Reihenfolge ein bestimmtes Problem gelöst wird.

Eine einfache Beschreibung einer Lösung könnte so aussehen:

Der ggT zweier Zahlen kann naturgemäß nicht größer sein als die kleinere der beiden Zahlen. Deshalb beginnen wir mit ihr als potenziellem ggT.

Ist die so gefundene Zahl nicht ggT der beiden Zahlen, dann vermindere sie um 1 und wiederhole diesen Schritt.

Spätestens bei 1 terminiert diese Schleife, denn die 1 ist Teiler von allem.

Um die Lösung zu visualisieren, verwende ich das Aktivitätsdiagramm der UML. Die Unified Modeling Language, kurz »UML«, ist eine grafische Sprache, mit deren Hilfe Sachverhalte der Softwareentwicklung (Programmfluss, Klassen- und Objektbeziehungen, Zustände etc.) dargestellt werden.

In Abbildung 1–1 wird das Aktivitätsdiagramm der UML eingesetzt, um einen Algorithmus grafisch darzustellen.

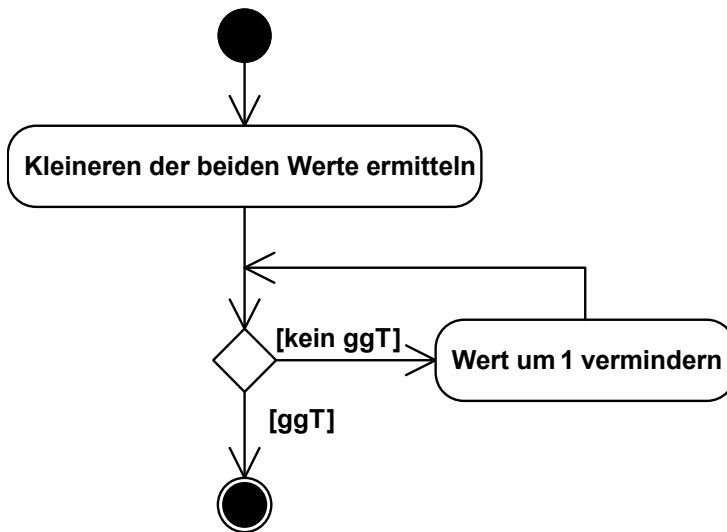


Abb. 1–1 Der größte gemeinsame Teiler als Aktivitätsdiagramm

Der ausgefüllte Kreis definiert den Start der Aktivität, sie endet am ausgefüllten Kreis mit Ring.

Einzelne Schritte oder Anweisungen werden in Form von Rechtecken mit abgerundeten Ecken dargestellt. Die Abarbeitungsreihenfolge ist über die Pfeile definiert, die aus den Symbolen heraustreten beziehungsweise zu ihnen hin führen.

Das auf der Spitze stehende Quadrat definiert eine Verzweigung. Aus einer Verzweigung tritt immer mehr als ein Pfeil aus. An den Pfeilen stehen in eckigen Klammern die sogenannten Wächter. Die bestimmen, unter welcher Bedingung dem jeweiligen Pfeil gefolgt wird.

Jedem Programm liegt zwangsläufig ein Algorithmus zugrunde. Wir werden im alltäglichen Leben andauernd mit Algorithmen konfrontiert. Alle nach Regeln ablaufenden Tätigkeiten sind gewissermaßen Algorithmen. Typische Algorithmen sind zum Beispiel einen Kuchen backen oder einen Fahrschein aus dem

Automaten ziehen. Selbst das Abschließen der Wohnungstür ist ein Algorithmus, wenn auch ein recht primitiver.

Wo immer bestimmte Tätigkeiten ein Problem lösen oder eine Aufgabe bewältigen, haben wir es mit Algorithmen zu tun. Dabei werden Algorithmen in der Weise formuliert, die auch Personen ohne Fachkenntnis das Lösen des Problems ermöglicht. Dies setzt eine Beschreibung mit möglichst klein gehaltenem Vokabular voraus.

Nehmen wir als Beispiel ein Kuchenrezept. Selbst wenn absolut kein Wissen um die Kunst des Kuchenbackens vorliegt, ist man in der Lage, einen Kuchen nach Rezept zu backen, weil sich die Anweisungen einer Sprache bedienen, die jedem Nichtbäcker verständlich ist (z. B. rühren, kneten, in die Backform füllen, Zucker abwiegen).

Und wenn man einen Algorithmus so formuliert, dass ein Computer ihn befolgen kann, dann spricht man von einem Programm.

Programm

Einen Algorithmus, der für den Computer verständlich formuliert wurde, nennt man Programm.

2 Grundelemente eines C++-Programms

In diesem Kapitel legen wir den Grundstein zur Programmierung in C++. Wir schauen uns an, welche Elemente immer in einem C++-Programm vorkommen und wie Texte auf dem Bildschirm ausgegeben werden können.

2.1 Das erste Programm

Schreiben wir nun unser erstes C++-Programm:

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

Um das Programm übersetzen und starten zu können, sollten Sie in der Entwicklungsumgebung Ihrer Wahl ein neues Projekt anlegen, dort eine C++-Datei hinzufügen (an der Endung `.cpp` zu erkennen) und dort das obere Programm einfügen.

Das Programm sollte sich fehlerfrei kompilieren und starten lassen, allerdings wird noch nichts passieren.

Wir haben bisher lediglich das Kernstück eines jeden C++-Programms programmiert, die `main`-Funktion. Jedes C++-Programm muss genau eine `main`-Funktion besitzen, sie ist der Einstiegspunkt in unser Programm. Jeder Start eines C++-Programms beginnt mit der `main`-Funktion.

Dem Funktionsnamen folgt ein Paar runder Klammern. Diese Klammern dienen später dazu, Informationen an die Funktion zu übergeben, bleiben aber fürs Erste leer.

Hinter dem Funktionskopf stehen geschweifte Klammern, mit denen in C++ eine zusammengesetzte Anweisung (compound statement) gebildet wird. Alle Anweisungen innerhalb der geschweiften Klammern werden beim Aufruf der Funktion ausgeführt. Da die Klammern bisher leer sind, passiert beim Start auch noch nichts.

2.1.1 Implizites return

Vor `main` steht immer `int`, das fordert der ISO-Standard. Ohne an dieser Stelle bereits genauer auf die Datentypen von C++ einzugehen, bedeutet dieses `int`, dass `main` immer einen ganzzahligen Wert zurückgibt. Über diesen Wert teilt das Programm der aufrufenden Umgebung mit, ob es fehlerfrei beendet wurde oder nicht. In einigen Fällen wird auch ein Fehlercode zurückgegeben, der den aufgetretenen Fehler genauer spezifiziert.

Der Compiler übersetzt die `main`-Funktion immer so, dass der zurückgegebene Wert die Bedeutung »alles in Ordnung« hat, und das ist üblicherweise der Wert 0. Das vom Compiler erzeugte Programm sieht damit so aus:

```
int main()
{
    return 0;
}
```

Diese automatische Ergänzung mit einer `return`-Anweisung, falls der Programmierer kein eigenes `return` programmiert hat, nimmt der Compiler nur bei der `main`-Funktion vor. In allen anderen Fällen ist der Programmierer dafür verantwortlich, eine geeignete `return`-Anweisung zu programmieren.

2.2 Die Ausgabe

Um unser erstes C++-Programm aus dem Stadium der Sinnlosigkeit herauszuheben, werden wir nun einen der wichtigsten Aspekte besprechen: die Ausgabe. Schauen wir uns dazu zunächst das erweiterte Programm an:

```
#include<iostream>

int main() {
    std::cout << "Hello World";
}
```

Auf dem Bildschirm sollte der Text »Hello World« erscheinen, je nach Entwicklungsumgebung noch direkt gefolgt von der Aufforderung, das Programm mit einem Tastendruck zu beenden.

Dieses kleine Beispiel bietet uns bereits die Möglichkeit, einige grundlegende Dinge von C++ zu besprechen.

Einfache Anweisungen werden in C++ mit einem Semikolon abgeschlossen.

Und noch eine Regel ist wichtig:

Konstante Zeichenfolgen stehen in C++ in Anführungszeichen.

Darüber hinaus muss in C++ penibel auf Groß- und Kleinschreibung geachtet werden. Der Name »Andre« und der Name »andre« sind zwei unterschiedliche Bezeichner.

2.2.1 cout

Der Befehl zur Ausgabe auf die Konsole heißt in C++ `cout`. Warum davor noch ein `std::` steht, besprechen wir gleich.

`cout` ist der sogenannte Standardausgabe-Stream beziehungsweise das Objekt, das der Abstraktion wegen für den Standardausgabestrom steht. Dadurch wird der Programmierer nicht mehr mit den plattformspezifischen Eigenarten der Ausgabe belastet. Er gibt die auszugebenden Daten einfach an das `cout`-Objekt und dieses sorgt dann für eine ordnungsgemäße Ausgabe. Als Standardausgabe wird im Allgemeinen der Bildschirm verwendet.

Der `<<`-Operator schiebt bildlich gesprochen die Daten in den Ausgabestrom. In diesem Fall handelt es sich bei den auszugebenden Daten um eine Stringkonstante.

Stringkonstante

Stringkonstante ist eine in doppelten Anführungszeichen stehende Folge von Zeichen, die implizit mit dem Wert 0 (`'\0'`) abgeschlossen wird.

Gehen wir den Ablauf des Programms einmal schrittweise durch.

Wenn Sie das Programm kompilieren und starten, wird zuerst die Funktion `main` aufgerufen. Die erste Anweisung ist die Anweisung, die die auszugebenden Daten an das `cout`-Objekt schickt, also in den Standardausgabe-Stream schiebt. `cout` gehört zur Standardbibliothek von C++.

Nachdem die auszugebenden Daten zu `cout` geschickt wurden, fährt das Programm hinter dem Semikolon der Anweisung fort. Dort ist aber nur das Ende der Funktion `main`, was dem Ende des gesamten Programms gleichkommt.

Sie werden sich vielleicht gewundert haben, dass im Programmtext die `cout`-Anweisung (und vorher auch schon die `return`-Anweisung) nach rechts eingerückt ist. Dies ist nicht notwendig, dient aber der Übersichtlichkeit. Wenn Sie die gleiche Einrückung wie hier im Buch verwenden möchten, dann sollten Sie die Abstände auf zwei Zeichen einstellen.

Zeilenumbruch

Um bei der Ausgabe eine neue Zeile zu beginnen, reicht es nicht aus, eine zweite Ausgabe zu tätigen:

```
#include<iostream>

int main() {
    std::cout << "Hello World!";
    std::cout << "Jetzt komme ich!";
}
```

Stattdessen muss an der Stelle, an der die neue Zeile beginnen soll, ein Zeilenumbruch in die Zeichenfolge eingefügt werden. Dies geschieht in Form einer Escape-Sequenz. Tabelle 2–1 listet alle in C++ verfügbaren Escape-Sequenzen auf. An dieser Stelle wollen wir die Escape-Sequenz für Newline einsetzen, sie lautet `\n`.

Das Programm mit Zeilenumbrüchen sieht damit so aus:

```
#include<iostream>

int main() {
    std::cout << "Hello World!\n";
    std::cout << "Jetzt komme ich!\n";
}
```

endl

Eine weitere Möglichkeit, einen Zeilenumbruch zu erhalten, ist der Manipulator `endl`, der über den Ausgabestrom ausgegeben wird:

```
#include<iostream>

int main() {
    std::cout << "Hello World!";
    std::cout << std::endl;
    std::cout << "Jetzt komme ich!";
    std::cout << std::endl;
}
```

Manipulator

Als Manipulator bezeichnet man ein Objekt, das über den Ausgabestrom ausgegeben wird und das Verhalten des Stroms manipuliert.

Das `endl` macht aber noch mehr, als einen Zeilenumbruch zu erzeugen. Dazu müssen wir uns anschauen, wie die Ausgabe funktioniert.

Statt direkt auf dem Bildschirm ausgegeben zu werden, landen die Ausgaben zunächst einmal in einem internen Speicherbereich, dem Ausgabepuffer. Erst wenn dieser Puffer voll ist, wird er auf dem Bildschirm ausgegeben. Unter

Umständen werden Ausgaben deshalb nicht sofort angezeigt, weil der Puffer einfach noch nicht voll ist.

Dieses Problem vermeidet `endl`, denn mit der Ausgabe von `endl` wird zusätzlich ein `flush` ausgeführt. Dieser Flush (aus dem Englischen »to flush«, was unter anderem die Bedienung der Toilettenspülung bedeutet) sorgt dafür, dass der Inhalt des Ausgabepuffers auf den Bildschirm »gespült« wird, auch wenn er noch nicht komplett gefüllt war.

Das `endl` ist aber nicht immer notwendig. Vor einer Eingabe oder am Programmende wird der Ausgabepuffer immer geleert, unabhängig von dessen Füllstand.

2.3 include

Mit der Ausgabe hat noch ein weiterer Befehl Einzug in unser Programm gehalten: die `include`-Direktive des Präprozessors:

```
#include<iostream>
```

Der Präprozessor – wie die Silbe »Prä« erahnen last – durchläuft die Datei vor dem eigentlichen Prozess des Kompilierens. Aber was genau bedeutet »kompilieren«? Abbildung 2–1 zeigt den Vorgang.

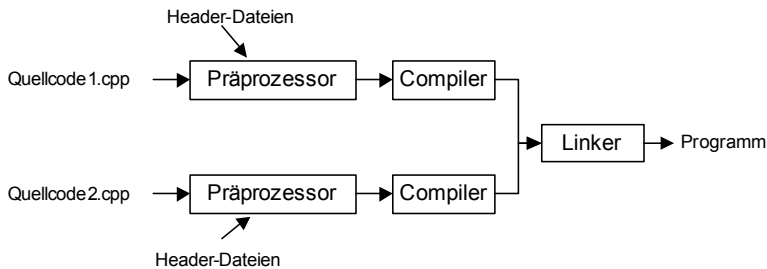


Abb. 2–1 Der Vorgang des Kompilierens

C++ ist eine Hochsprache, die vom Prozessor des Computers nicht verstanden wird, denn dieser kennt nur seine Maschinensprache. Maschinensprache ist eine sehr einfache, aus wenigen Befehlen bestehende Sprache. Entsprechend viele dieser Befehle sind notwendig, um selbst einfachste Dinge zu programmieren. Eine typische Szene könnte sein:

Lade Wert an Adresse \$92E2 in Register1. Addiere Wert an Adresse \$92E6 auf Register1. Speichere Inhalt von Register1 an Adresse \$92EA.

Dasselbe in C++ wäre etwa `x=a+b` – viel kürzer und vor allem für einen Menschen viel verständlicher.

Damit also ein in C++ geschriebenes Programm auf einem Rechner laufen kann, muss es in die Maschinensprache des Prozessors übersetzt werden. Und diesen Vorgang nennt man »kompilieren«.

Kompilieren

Das Übersetzen eines Hochsprachenprogramms in die Maschinensprache des Zielprozessors bezeichnet man als Kompilieren. Der Übersetzer wird Compiler genannt.

In C++ besteht dieser Übersetzungsprozess aus mehreren Schritten. Bevor eine `cpp`-Datei dem Compiler übergeben wird, durchläuft sie der Präprozessor, der nach an ihn gerichteten Befehlen sucht. Der Präprozessor selbst versteht C++ nicht, er arbeitet rein auf Textebene.

Präprozessordirektive

Diese an den Präprozessor gerichteten Befehle beginnen mit einem `#` in der ersten Spalte. Präprozessordirektiven dürfen nicht nach rechts eingerückt werden.

Befehle an den Präprozessor, sogenannte Präprozessordirektiven, beginnen immer mit einem `»#«`. Der wohl häufigste Befehl ist `#include`, was übersetzt so viel wie »Einbinden« bedeutet. Mithilfe dieses Befehls kann eine andere Datei in die Quellcodedatei eingebunden werden. In unserem Fall binden wir die Datei `»iostream«` ein, in der alle für die Textein- und -ausgabe notwendigen Elemente der C++-Standardbibliothek enthalten sind, unter anderem das in unserem Programm verwendete `cout` und `endl`. Würden wir die Include-Direktive aus dem Programm entfernen, käme bei der Kompilation die Fehlermeldung des Compilers, er würde `cout` und `endl` nicht kennen.

Abbildung 2–1 zeigt noch eine weitere Besonderheit von C++: Jede Quellcodedatei des Programms wird isoliert von den anderen kompiliert, der Compiler hat auch keinerlei Erinnerungen an sein Tun.

Für das Beispiel in der Abbildung heißt das: Während er die Datei `»Quellcode1.cpp«` kompiliert, weiß er nicht, dass er noch die Datei `»Quellcode2.cpp«` kompilieren wird. Und während er die Datei `»Quellcode2.cpp«` kompiliert, weiß er nicht, dass er die Datei `»Quellcode1.cpp«` bereits kompiliert hat.

Die einzeln kompilierten Dateien werden im letzten Schritt vom Linker (auf Deutsch so viel wie »Binder«) zu einer einzigen Datei zusammengebunden, die dem lauffähigen Programm entspricht.

2.4 Namensbereiche

Bleibt noch zu klären, warum vor `cout` und `endl` dieses `std::` steht.

Bildlich betrachtet ist `std` vergleichbar mit einer Vorwahl. Stellen Sie sich vor, es gäbe keine Vorwahlen. Dann müsste die Vergabe von Telefonnummern global geregelt werden, schließlich dürfen Teilnehmer in Köln und Timbuktu nicht dieselbe Telefonnummer bekommen. Ländervorwahlen lösen das Problem, denn jedes Land kann hinter seiner Vorwahl die Telefonnummern nach eigenen Regeln

vergeben. Jetzt dürfen auch Teilnehmer in München und Rom dieselbe Nummer besitzen, denn sie unterscheiden sich in der Vorwahl.

Dieses Prinzip nennt sich in C++ Namensbereich. Ein Namensbereich ist nichts anderes als eine programmiertechnische Vorwahl, hinter der Namen beliebig vergeben werden können. Der Namensbereich der C++-Standardbibliothek lautet `std` als Abkürzung von »Standard«.

In C++ kann eine Gruppe von Namen (das können Namen von Konstanten, Funktionen, Klassen etc. sein) zu einem Namensbereich zusammengefasst werden. Ganz konkret gehört die Definition von `cout` zum Namensbereich `std`.

Die Namensbereiche wurden eingeführt, um die Möglichkeit einer Doppelbenennung verhindern zu können. Man ist dadurch in der Lage, sein eigenes `cout` zu definieren, wenn man es einem anderen Namensbereich zuordnet.

using namespace

Bei den Telefonnummern gibt es eine Besonderheit: Möchte ich einen Teilnehmer mit derselben Vorwahl wie meine eigene Telefonnummer anrufen, dann muss ich die Vorwahl nicht mit wählen.

Etwas Ähnliches existiert auch in C++: Wir können dem Compiler mitteilen, dass er in bestimmten Namensbereichen automatisch suchen soll. Auf diese Weise können wir uns die explizite Angabe von `std` sparen, wenn wir ein Element der Standardbibliothek ansprechen möchten.

Der Befehl dazu lautet `using namespace`:

```
#include<iostream>

using namespace std;

int main() {
    cout << "Hello World!";
    cout << endl;
    cout << "Jetzt komme ich!";
    cout << endl;
}
```

Man spricht auch davon, einen Namensbereich global verfügbar zu machen. Die Elemente des Namensbereichs lassen sich dann ansprechen, als ständen sie überhaupt nicht in einem Namensbereich.

In unserem Beispiel brauchen wir dann bei Namen, die im Namensbereich `std` definiert sind, nicht mehr explizit angeben, dass wir die Definition aus `std` verwenden wollen. Elemente aus anderen Namensbereichen müssen weiterhin explizit mit ihrem Namensbereich angegeben werden. Es können aber mehrere `using namespace`-Anweisungen verwendet werden, falls weitere Namensbereiche global verfügbar gemacht werden sollen.

Der Einsatz von `using namespace` spart eine Menge Tipparbeit und gestaltet das Programm übersichtlicher. Und wo wir gerade bei dem Thema »Tipparbeit sparen« sind: Bei der Ausgabe lässt sich der Operator `<<` auch verketteten:

```
#include<iostream>

using namespace std;

int main() {
    cout << "Hello World!" << endl;
    cout << "Jetzt komme ich!" << endl;
}
```

Das erste `endl` ist eigentlich unnötig, weil es an dieser Stelle nur um einen Zeilenumbruch geht und nicht um das Leeren des Ausgabepuffers. Wir könnten es daher auch mit der Escape-Sequenz `\n` ersetzen:

```
#include<iostream>

using namespace std;

int main() {
    cout << "Hello World!\nJetzt komme ich!" << endl;
}
```

Wie eigene Namensbereiche erstellt werden können, besprechen wir in Kapitel 13.

2.5 Kommentare

Nachdem uns nun das erste Programm komplett verständlich ist, wollen wir in diesem Abschnitt die Möglichkeit besprechen, Kommentare in das eigentliche Programm einfügen zu können.

Kommentare dienen dazu, Informationen im Programm unterzubringen, die nicht Bestandteil des tatsächlichen Programms sind und auch vom Compiler nicht verstanden werden könnten.

So können Sie zum Beispiel hinter bestimmten Programmzeilen Bemerkungen schreiben, damit Sie auch später noch wissen, welche Funktion sie haben. Oder Sie können vor jedem größeren Abschnitt ein paar Zeilen über seine Funktion schreiben. Obwohl immer wieder manche Programmierer behaupten, ein Programmtext sei selbsterklärend und nur unfähige Leute bräuchten Kommentare, sollten Sie sich diesbezüglich nicht einschüchtern lassen und nie mit Kommentaren geizen. Sie tun sich und anderen, die Ihr Programm einmal verstehen müssen, einen großen Gefallen.

Mehrzeilige Kommentare

Mehrzeilige Kommentare beginnen mit den Zeichen `/*` und werden mit `*/` beendet. Alles, was zwischen diesen Zeichen steht, wird vom Compiler ignoriert.