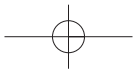
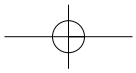
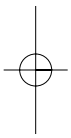
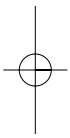
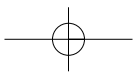


C++
Wochenend Crashkurs









C++ Wochenend Crashkurs

Stephen R. Davis

Übersetzung aus dem Amerikanischen von Dr. Thorsten Graf

Die Deutsche Bibliothek - CIP-Einheitsaufnahme:

Davis, Stephen R.:
C++ Wochenend Crashkurs
Übersetzung aus dem Amerikanischen von Dr. Thorsten Graf
Bonn : MITP-Verlag, 2001
Einheitssacht.: C++ Wochenend Crashkurs
ISBN 3-8266-0692-2

ISBN 3-8266-0692-2

Übersetzung der amerikanischen Originalausgabe:
Stephen R. Davis: C++ Weekend Crash Course

Printed in Germany

Lektorat: Christine Wöltche
Korrektorat: Michael Eckloff
Herstellung: Dieter Schulz
Druck: Media-Print, Paderborn
Satz und Layout: Eva Kraskes, Köln

Über den Autor

Stephen R. Davis
Wenn der 43-jährige Vater und Hausmann nicht gerade Fahrrad fährt oder seinen Sohn zu einem Taekwondo-Wettkampf begleitet, arbeitet und lebt Stephen R. Davis in Greenville, Texas als Programmierer mit Leib und Seele.
Beim MITP-Verlag ist neben dem *C++ Wochenend Crashkurs* auch sein Buch *C++ für Dummies* erschienen.

Vorwort

Mit dem C++ *Wochenend Crashkurs* erlernen Sie an einem einzigen – zugegebenermaßen arbeitsreichen – Wochenende die Programmsprache C++: in 30 Sitzungen à 30 Minuten, also 15 Stunden – von Freitagabend bis Sonntagnachmittag.
Am Ende jeden Teils bekommen Sie Gelegenheit für eine Pause und eine Rekapitulation dessen, was Sie erlernt haben. Viel Glück!

1.1 Was ist C++?

C++ ist heute die populärste Programmiersprache. C++ wird in Anwendungen eingesetzt, die vom Mikroprogramm, das in Ihrer Mikrowelle, in Ihrer Waschmaschine oder in Ihrem Fernseher läuft, bis hin zu Programmen zur Steuerung von Atomraketen oder Marsraketen reichen.

In den späten achtziger Jahren kam C allmählich in die Jahre. Das lag unter anderem daran, dass C keine objektorientierte Programmierung unterstützt. Zu dieser Zeit hat die objektorientierte Welle die Welt im Sturm erobert. Objektorientierten Programmierern wurde das Geld nachgeworfen. Sie brauchten im Gespräch nur »neues Paradigma« zu sagen und hatten sofort eine Menge Bewunderer.

Das Problem war, dass jedes vernünftige Programm in C geschrieben war (es gab einige Programme, die in PASCAL geschrieben waren, wie frühere Versionen von Windows, aber die zählen nicht – wenn Sie die frühen Versionen von Windows kennen, wissen Sie, warum nicht). Die Firmen würden nicht einfach alle existierenden Programme neu schreiben, nur um auf der objektorientierten Welle zu reiten.

Objektorientierte Konzepte wurden in die existierende Programmiersprache C integriert. Das Ergebnis wurde C++ genannt.

C++ ist eine Obermenge von C. Jedes korrekt geschriebene C-Programm kann unter C++ erzeugt werden. Dadurch konnten die Firmen ihre Software stückweise upgraden. Existierender Code konnte weiterhin in C geschrieben sein, wohingegen neuer Code die Features von C++ nutzte.

Zu unserem Glück ist C++ eine standardisierte Sprache. Das American National Standards Institute (ANSI) und die International Standards Organisation (ISO) sind sich einig darüber, was C++ ist. Sie haben eine detaillierte Beschreibung der Programmiersprache C++ erstellt. Diese standardisierte Sprache ist unter dem Namen ANSI oder ISO C++, oder einfach Standard-C++, bekannt.

Standard-C++ unterliegt nicht der Kontrolle einer einzelnen Firma, wie z.B. Microsoft oder Sun. Die Gemeinschaft der Programmierer, die Standard-C++ verwenden, ist nicht abhängig von einem Software Giganten. Außerdem halten sich die Firmen an den Standard, selbst Microsoft's Visual C++ hält sich streng an den C++-Standard.

Die Programme im *C++ Wochenend Crashkurs* können mit jeder Implementierung von Standard-C++ übersetzt werden.

1.2 Das objektorientierte Paradigma

Objektorientierte Programmierung ist nicht nur ein Modetrend. Objektorientierte Programmierung ist eine Methode der Programmierung, die sich sehr von ihren Vorgängern unterscheidet. Objektorientierte Programme können leichter geschrieben und gepflegt werden. Objektorientierte Module können leichter wiederverwendet werden als die Module, die unter einem anderen Paradigma erstellt wurden.

Der *C++ Wochenend Crashkurs* ist mehr als nur eine Einführung in C++. Sie müssen das objektorientierte Paradigma erlernen, um C++ voll nutzen zu können. Der *C++ Wochenend Crashkurs* verwendet Beispiele in C++, um Ihnen die objektorientierte Sicht auf die Welt zu vermitteln. Jeder, der behauptet, in C++ zu programmieren, ohne die objektorientierten Konzepte verstanden zu haben, verwendet C++ als »besseres C«.

1.3 Wer

Der *C++ Wochenend Crashkurs* richtet sich an Anfänger bis hin zu Lesern auf mittlerem Level.

Es werden keine Vorkenntnisse im Bereich Programmierung und Programmierkonzepte beim Leser vorausgesetzt. Die ersten Sitzungen erklären anhand realer Beispiele auf nicht-technische Weise, was Programmierung ist.

Dieses Buch ist auch gut geeignet für den Hobbyprogrammierer. Die vielen Beispiele demonstrieren Programmiertechniken, die in modernen Programmen eingesetzt werden.

Der ernsthafte Programmierer oder Student muss C++ in seinem Köcher der Programmierfähigkeiten haben. Fundiertes Wissen in C++ zu haben, kann den Unterschied machen, ob man einen bestimmten Job bekommt oder nicht.

1.4 Was

Der *C++ Wochenend Crashkurs* ist mehr als nur ein Buch. Er ist ein vollständiges Entwicklungspaket. Eine CD-ROM enthält die berühmte GNU C++-Umgebung.

Sie benötigen ein Textprogramm, wie z.B. Microsoft Word, um Texte bearbeiten zu können. Und sie brauchen eine C++-Entwicklungsumgebung, um Programme in C++ zu erzeugen und auszuführen.

Viele Leser werden bereits eine eigene Entwicklungsumgebung besitzen, wie z.B. Microsoft's Visual C++. Für die Leser, die noch über keine Entwicklungsumgebung verfügen, enthält der *C++ Wochenend Crashkurs* das GNU C++.

GNU C++ ist kein abgespecktes oder laufzeitbeschränktes Programm. Das GNU C++-Paket ist eine vollwertige Entwicklungsumgebung. Der *C++ Wochenend Crashkurs* enthält vollständige Anleitungen zur Installation von GNU C++ und Visual C++.

1.5 Wie

Der *C++ Wochenend Crashkurs* ist für ein Wochenende gedacht. Fangen Sie am Freitagabend an, dann sind Sie am Sonntagnachmittag fertig.

Dieses Ein-Wochen-Format ist

- ideal für Studenten, die mit ihren Mitstudenten gleichziehen möchten,
- ideal für den Programmierer, der seine Fähigkeiten erweitern will, und
- ideal für jeden, der C++ lernen möchte, während die Kinder bei der Oma sind.

Natürlich können Sie das Buch auch etwas langsamer durcharbeiten, wenn Sie das lieber tun. Jeder Teil von 4 bis 6 Sitzungen kann separat gelesen werden.

Der Leser sollte jede der 30 Sitzungen innerhalb von 30 Minuten durcharbeiten können. Zeitmarkierungen helfen, die Zeit im Auge zu behalten.

Am Ende jeder Sitzung befinden sich Fragen, die dem Leser zur Selbsteinschätzung dienen sollen. Eine Menge schwierigerer Fragen, die helfen sollen, das Erlernte zu festigen, befindet sich am Ende jeden Buchteils.

1.6 Überblick

Der *C++ Wochenend Crashkurs* präsentiert seine Sitzungen in Gruppen von 4 bis 6 Kapiteln, die in 6 Buchteile organisiert sind.

1.6.1 Freitagabend – Einführung in die Programmierung

Dieser Teil führt Programmierkonzepte ein und führt Sie durch Ihr erstes Programm.

1.6.2 Samstagmorgen – Einstieg in C++

Dieser Teil behandelt Themen wie Anweisungssyntax, Operatoren und elementare Funktionen.

1.6.3 Samstagnachmittag – Strukturen und Zeiger

Hier beschäftigt sich der Leser mit dem etwas komplizierteren Thema der Zeigervariablen, zusammen mit ihrem Einsatz in verketteten Listen, Arrays und Objekten.

1.6.4 Samstagabend – Einführung in die objektorientierte Programmierung

Das ist ein Punkt mit Schlüsselcharakter – Themen wie C++-Strukturen, die Grundlage der objektorientierten Programmierung sind, werden besprochen.

1.6.5 Sonntagmorgen – Objektorientierte Programmierung

Hier ist die Hauptschlagader. Dieser Teil taucht in die Syntax und die Bedeutung der objektorientierten Programmierung ein.

1.6.6 Sonntagnachmittag – Abschluss

Dieser Teil stellt einige fortgeschrittene Themen dar, wie Fehlerbehandlung und das Überladen von Operatoren.

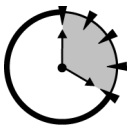
Jeder Teil endet mit einer Diskussion von Debug-Techniken, um die offensichtlichen Fehler in Ihren Programmen zu finden und zu entfernen. Die Komplexität dieser Techniken ist den Fähigkeiten angepasst, die der Leser in der Sitzung erlernt hat.

Der Anhang enthält weiterführende Programmierprobleme zu jeder Sitzung.

1.7 Layout und Features

Niemand sollte versuchen, sich ohne Pause durch das Material durchzuschlagen. Nach jeder Sitzung und am Ende eines jeden Teils finden Sie einige Fragen, um Ihr Wissen zu überprüfen und Ihre neu erworbenen Fähigkeiten auszuprobieren. Machen Sie eine Pause, holen Sie sich einen Snack, trinken Sie einen Kaffee und gehen Sie dann in die nächste Sitzung.

Entlang Ihres Weges finden Sie Markierungen, die Ihnen bei der Orientierung helfen sollen. Sie sagen Ihnen, wo Sie sich gerade befinden und weisen Sie auf interessante Punkte hin, die Sie nicht verpassen sollten. Wenn Sie eine Sitzung durcharbeiten, halten Sie nach den folgenden Zeichen Ausschau:



Dieses und ähnliche Icons zeigen Ihnen, wie weit Sie bereits in der Sitzung gekommen sind.

20 Min.

Es gibt eine Reihe von Icons, die Sie auf spezielle Informationen hinweisen sollen:

Dieses Zeichen weist auf Informationen hin, die Sie im Gedächtnis behalten sollten. Sie werden Ihnen später noch nützlich sein.

Hier erhalten Sie hilfreiche Hinweise darauf, wie Sie eine Sache am besten ausführen oder erfahren eine Technik, die Ihre Programmierung einfacher macht.

Tun Sie das niemals!

Dieses Zeichen weist auf Informationen hin, die Sie auf der CD-ROM finden, die diesem Buch beiliegt.

1.8 Konventionen in diesem Buch

Abgesehen von den Zeichen, die Sie gerade gesehen haben, gibt es nur zwei Konventionen, die in diesem Buch verwendet werden:

- Programmcode, der im normalen Text verwendet wird, erscheint in einem speziellen Font, wie in folgendem Beispiel zu sehen ist:
  Wenn ich die Funktion `main( )` schreibe, kann ich mich auf den Wert konzentrieren, der von der Funktion `sumSequence( )` zurückgegeben wird, ohne darüber nachzudenken, wie diese Funktion intern arbeitet.
- Programmbeispiele, die sich nicht im normalen Text befinden, werden wie folgt dargestellt:

```
float fVariable1 = 10.0;
float fVariable2 = (10 / 3) * 3;
fVariable1 == fVariable2; // sind die beiden gleich?
```

1.9 Was fehlt noch?

Nichts. Öffnen Sie die erste Seite Ihres Arbeitsbuches und halten Sie die Uhr bereit. Es ist Freitagabend, und Sie haben zwei Tage Zeit.

Inhalt

Freitag

Teil 1 – Freitagabend 2

Lektion 1 – Was ist Programmierung? 3
1.1 Ein menschliches Programm 4
1.1.1 Der Algorithmus 4
1.2 Der Prozessor 4
1.3 Das Programm 5
1.4 Computerprozessoren 7
Zusammenfassung 7
Selbsttest 7

Lektion 2 – Ihr erstes Programm in Visual C++ 8
2.1 Installation von Visual C++ 8
2.2 Ihr erstes Programm 9
2.3 Erzeugen Ihres Programms 10
2.4 Ausführen Ihres Programms 14
2.5 Abschluss 15
2.5.1 Programmausgabe 16
2.5.2 Visual C++-Hilfe 16
Zusammenfassung 17
Selbsttest 17

Lektion 3 – Ihr erstes C++-Programm mit GNU C++ 18
3.1 Installation von CNU C++ 18
3.2 Ihr erstes Programm 20
3.2.1 Eingabe des C++-Codes 20
3.3 Erzeugen Ihres Programms 23
3.4 Ausführen Ihres Programms 26

3.5 Abschluss 27
3.5.1 Programmausgabe 27
3.5.2 GNU C++-Hilfe 27
Zusammenfassung 28
Selbsttest 28

Lektion 4 – C++-Instruktionen 29
4.1 Das Programm 29
4.2 Das C++-Programm erklärt 30
4.2.1 Der grundlegende Programmaufbau 30
4.2.2 Kommentare 31
4.2.3 Noch mal der Rahmen 32
4.2.4 Anweisungen 32
4.2.5 Deklarationen 32
4.2.6 Eingabe/Ausgabe 33
4.2.7 Ausdrücke 33
4.2.8 Zuweisung 34
4.2.9 Ausdrücke (Fortsetzung) 34
Zusammenfassung 34
Selbsttest 35
Freitagabend – Zusammenfassung 36

Samstag

Teil 2 – Samstagmorgen 40

Lektion 5 – Variablentypen 41
5.1 Dezimalzahlen 42
5.1.1 Begrenzungen von int in C++ 42
5.1.2 Lösen des Abschneideproblems 45
5.1.3 Grenzen von Gleitkommazahlen 46
5.2 Andere Variablentypen 47
5.2.1 Typen von Konstanten 48
5.2.2 Sonderzeichen 48
5.3 Gemischte Ausdrücke 50
Zusammenfassung 51
Selbsttest 52

Lektion 6 – Mathematische Operationen 53
6.1 Arithmetische Operatoren 54
6.2 Ausdrücke 54
6.3 Vorrang von Operatoren 55
6.4 Unäre Operatoren 56
6.5 Zuweisungsoperatoren 57
Zusammenfassung 58
Selbsttest 58

Lektion 7 – Logische Operationen 59
7.1 Einfache logische Operatoren 59
7.1.1 Kurze Schaltkreise und C++ 61
7.1.2 Logische Variablentypen 62
7.2 Binäre Zahlen 62
7.3 Bitweise logische Operationen 63
7.3.1 Die Einzelbit-Operatoren 63
7.3.2 Die bitweisen Operatoren 64
7.3.3 Ein einfacher Test 65
7.3.4 Warum? 66
Zusammenfassung 67
Selbsttest 68

Lektion 8 – Kommandos zur Flusskontrolle 69
8.1 Das Verzweigungskommando 69
8.2 Schleifenkommandos 71
8.2.1 Die while-Schleife 71
8.2.2 Die for-Schleife 74
8.2.3 Spezielle Schleifenkontrolle 75
8.3 Geschachtelte Kontrollkommandos 78
8.4 Können wir switchen? 79
Zusammenfassung 80
Selbsttest 80

Lektion 9 – Funktionen 81
9.1 Code einer Sammelfunktion 81
9.1.1 Sammelcode 82
9.2 Funktion. 84
9.2.1 Warum Funkionen? 84
9.2.2 Einfache Funktionen 85
9.2.3 Funktionen mit Argumenten 85
9.2.4 Mehrere Funktionen mit gleichem Namen 88
9.3 Funktionsprototypen 89
9.4 Verschiedene Speichertypen 90
Zusammenfassung 91
Selbsttest 91

Lektion 10 – Debuggen 92
10.1 Fehlertypen 92
10.2 Die Technik der Ausgabeanweisungen. 93
10.3 Abfangen von Bug Nr. 1 94
10.3.1 Visual C++ 95
10.3.2 GNU C++ 96
10.4 Abfangen von Bug Nr. 2 97
Zusammenfassung 100
Selbsttest 100
Samstagmorgen – Zusammenfassung 101

Teil 3 – Samstagnachmittag 104

Lektion 11 – Das Array 105
11.1 Was ist ein Array? 105
11.1.2 Ein Array in der Praxis 108
11.1.3 Initialisierung eines Array 110
11.1.4 Warum Arrays benutzen? 110
11.1.5 Arrays von Arrays 111
11.2 Arrays von Zeichen 111
11.3 Manipulation von Zeichenketten 113
11.3.1 Unsere eigene Verbindungsfunktion 114
11.3.2 Funktionen für C++-Zeichenketten 116
11.3.3 Wide Character 117
11.4 Obsolete Ausgabefunktionen 117
Zusammenfassung 118
Selbsttest 118

Lektion 12 – Einführung in Klassen 119
12.1 Gruppieren von Daten 119
12.1.1 Ein Beispiel 120
12.1.2 Das Problem 122
12.2 Die Klasse 122
12.2.2 Beispielprogramm 124
12.2.3 Vorteile 126
Zusammenfassung 126
Selbsttest 126

Lektion 13 – Einstieg C++-Zeiger 127
13.1 Was ist deine Adresse? 128
13.2 Einführung in Zeigervariablen 129
13.3 Typen von Zeigern 132
13.4 Übergabe von Zeigern an Funktionen 133
13.4.1 Wertübergabe 133
13.4.2 Übergabe von Zeigerwerten 134
13.4.3 Referenzübergabe 135
13.5 Heap-Speicher 135
13.5.1 Geltungsbereich 135
13.5.2 Das Geltungsbereichsproblem 137
13.5.3 Die Heap-Lösung 137
Zusammenfassung 138
Selbsttest 139

Lektion 14 – Mehr über Zeiger 140
14.1 Zeiger und Arrays 140
14.1.1 Operationen auf Zeigern 141
14.1.2 Zeichenarrays 145
14.1.3 Operationen auf unterschiedlichen Zeigertypen 148

14.2 Argumente eines Programms 150
14.2.1 Arrays von Zeigern 150
14.2.2 Arrays von Zeichenketten 151
14.2.3 Die Argumente von main( ) 152
Zusammenfassung 154
Selbsttest 155

Lektion 15 – Zeiger auf Objekte 156
15.1 Zeiger auf Objekte 156
15.1.1 Übergabe von Objekten 157
15.1.2 Referenzen 159
15.1.3 Rückgabe an den Heap 159
15.2 Die Datenstruktur Array 160
15.3 Verkettete Listen 160
15.3.1 Anfügen am Kopf der verketteten Liste 161
15.3.2 Andere Operationen auf verketteten Listen 162
15.3.3 Eigenschaften verketteter Listen 164
15.4 Ein Programm mit verkettetem NameData 165
15.5 Andere Container 168
Zusammenfassung 168
Selbsttest 168

Lektion 16 – Debuggen II 169
16.1 Welcher Debugger? 170
16.2 Das Testprogramm 170
16.3 Einzelschritte durch ein Programm 172
16.4 Einzelschritte in eine Funktion hinein 173
16.5 Verwendung von Haltepunkten 175
16.6 Ansehen und Modifizieren von Variablen 176
16.7 Verwendung des Visual C++-Debuggers 179
Zusammenfassung 181
Selbsttest 182
Samstagnachmittag – Zusammenfassung 183

Teil 4 – Samstagabend 186

Lektion 17 – Objektprogrammierung 187
17.1 Abstraktion und Mikrowellen 187
17.1.1 Funktionale Nachos 188
17.1.2 Objektorientierte Nachos 188
17.2 Klassifizierung und Mikrowellen 189
17.2.1 Warum solche Objekte bilden? 189
17.2.2 Selbstenthaltende Klassen 190
Zusammenfassung 191
Selbsttest 191

Lektion 18 – Aktive Klassen 192
18.1 Klassenrückblick 192
18.2 Grenzen von struct 193
18.2.1 Eine funktionale Lösung 194
18.3 Definition einer aktiven Klasse 195
18.3.1 Namengebung für Elementfunktionen 196
18.4 Definition einer Elementfunktion einer Klasse 197
18.5 Schreiben von Elementfunktionen außerhalb einer Klasse 198
18.5.1 Include-Dateien 199
18.6 Aufruf einer Elementfunktion 200
18.6.1 Aufruf einer Elementfunktion über einen Zeiger 201
18.6.2 Zugriff auf andere Elemente von einer Elementfunktion aus 202
18.7 Überladen von Elementfunktionen 204
Zusammenfassung 205
Selbsttest 205

Lektion 19 – Erhalten der Klassenintegrität 206
19.1 Erzeugen und Vernichten von Objekten 206
19.1.1 Der Konstruktor 207
19.1.2 Der Destruktor 212
19.2 Zugriffskontrolle 214
19.2.1 Das Schlüsselwort protected 214
19.2.2 Statische Datenelemente 217
Zusammenfassung 218
Selbsttest 219

Lektion 20 – Klassenkonstuktoren II 220
20.1 Konstruktoren mit Argumenten 220
20.2 Konstruktion von Klassenelementen 223
20.3 Reihenfolge der Konstruktion 228
20.3.1 Lokale Objekte werden in der Reihenfolge konstruiert 229
20.3.2 Statische Objekte werden nur einmal angelegt 229
20.3.3 Alle globalen Variablen werden vor main( ) erzeugt 229
20.3.4 Keine bestimmte Reihenfolge für globale Objekte 229
20.3.5 Elemente werden in der Reihenfolge ihrer Deklaration konstruiert 231
20.3.6 Destruktoren in umgekehrter Reihenfolge wie Konstruktoren 231
20.4 Der Kopierkonstruktor 231
20.4.1 Flache Kopie gegen tiefe Kopie 233
20.4.2 Ein Fallback-Kopierkonstruktor 234
Zusammenfassung 235
Selbsttest 236
Samstagabend – Zusammenfassung 237

Sonntag

Teil 5 – Sonntagmorgen ... 240

Lektion 21 – Vererbung ... 241
21.1 Vorteile der Vererbung ... 241
21.2 Faktorieren von Klassen ... 242
21.3 Implementierung von Vererbung in C++ ... 243
21.4 Unterklassen konstruieren ... 246
21.5 Die Beziehung HAS_A ... 249
Zusammenfassung ... 251
Selbsttest ... 251

Lektion 22 – Polymorphie ... 252
22.1 Elementfunktionen überschreiben ... 252
22.2 Einstieg in Polymorphie ... 254
22.3 Polymorphie und objektorientierte Programmierung ... 254
22.4 Wie funktioniert Polymorphie? ... 257
22.5 Was ist eine virtuelle Funktion nicht? ... 259
22.6 Überlegungen zu virtual ... 261
Zusammenfassung ... 262
Selbsttest ... 262

Lektion 23 – Abstrakte Klassen und Faktorieren ... 263
23.1 Faktorieren ... 263
23.2 Abstrakte Klassen ... 267
23.2.1 Deklaration einer abstrakten Klasse ... 267
23.2.2 Erzeugung einer konkreten Klasse aus einer abstrakten Klasse ... 269
23.2.3 Warum ist eine Unterklasse abstrakt? ... 271
23.2.4 Ein abstraktes Objekt an eine Funktion übergeben ... 272
23.2.5 Warum werden rein virtuelle Funktionen benötigt? ... 273
Zusammenfassung ... 274
Selbsttest ... 275

Lektion 24 – Mehrfachvererbung ... 276
24.1 Wie funktioniert Mehrfachvererbung? ... 276
24.2 Uneindeutigkeiten bei der Vererbung ... 278
24.3 Virtuelle Vererbung ... 279
24.4 Konstruktion von Objekten bei Mehrfachnennung ... 285
24.5 Eine Meinung dagegen ... 285
Zusammenfassung ... 286
Selbsttest ... 287

Lektion 25 – Große Programme 288
25.1 Warum Programme aufteilen? 288
25.2 Trennung von Klassendefinition und Anwendungsprogramm 289
25.2.1 Aufteilen des Programms 289
25.2.2 Die #include-Direktive 290
25.2.3 Anwendungscode aufteilen 292
25.2.4 Projektdatei 293
25.2.5 Erneute Betrachtung des Standard-Programm-Templates 295
25.2.6 Handhabung von Outline-Elementfunktionen 296
Zusammenfassung 297
Selbsttest 297

Lektion 26 – C++-Präprozessor 298
26.1 Der C++-Präprozessor 298
26.2 Die #include-Direktive. 299
26.3 Die Direktive #define 299
26.3.1 Definition von Makros 300
26.3.2 Häufige Fehler bei der Verwendung von Makros 300
26.4 Compiler-Kontrolle 302
26.4.1 Die #if-Direktive 302
26.4.2 Die #ifdef-Direktive 303
Zusammenfassung 305
Selbsttest 306
Sonntagmorgen – Zusammenfassung 307

Teil 6 – Sonntagnachmittag 310

Lektion 27 – Überladen von Operatoren 311
27.1 Warum sollte ich Operatoren überladen? 312
27.2 Was ist die Beziehung zwischen Operatoren und Funktionen? 312
27.3 Wie funktioniert das Überladen von Operatoren? 313
27.3.1 Spezielle Überlegungen 316
27.4 Ein detaillierterer Blick 316
27.5 Operatoren als Elementfunktionen 318
27.6 Eine weitere Irritation durch Überladen 320
27.7 Wann sollte ein Operator ein Element sein? 321
27.8 Cast-Operator 321
Zusammenfassung 324
Selbsttest 324

Lektion 28 – Der Zuweisungsoperator 325
28.1 Warum ist das Überladen des Zuweisungsoperators kritisch? 325
28.1.1 Vergleich mit dem Kopierkonstruktor 326
28.2 Wie den Zuweisungsoperator überladen? 326
28.2.1 Zwei weitere Details des Zuweisungsoperators 329
28.3 Ein Schlupfloch 330
Zusammenfassung 331
Selbsttest 331

Lektion 29 – Stream-I/O 332
29.1 Wie funktioniert Stream-I/O? 332
29.2 Die Unterklassen fstream 333
29.3 Die Unterklassen strstream 337
29.3.1 Vergleich von Techniken der Zeichenkettenverarbeitung 338
29.4 Manipulatoren 341
29.5 Benutzerdefinierte Inserter 343
29.6 Schlaue Inserter 344
29.7 Aber warum die Shift-Operatoren? 346
Zusammenfassung 347
Selbsttest 347

Lektion 30 – Ausnahmen 348
30.1 Konventionelle Fehlerbehandlung 348
30.2 Warum benötigen wir einen neuen Fehlermechanismus? 349
30.3 Wie arbeiten Ausnahmen? 350
30.3.1 Warum ist der Ausnahmemechanismus eine Verbesserung? 352
30.4 Abfangen von Details, die für mich bestimmt sind 352
30.4.1 Was kann ich »werfen«? 354
30.5 Verketten von catch-Blöcken. 357
Zusammenfassung 358
Selbsttest 359
Sonntagnachmittag – Zusammenfassung 360

Anhang A: Antworten auf die Wiederholungsfragen 363
Anhang B: Ergänzende Probleme 381
Anhang C: Was ist auf der CD-Rom 393
Index 395
GNU General Public License 421