

Auf einen Blick

TEIL I

Grundlagen	29
------------------	----

TEIL II

Objektorientierte Programmierung und mehr	267
---	-----

TEIL III

Fortgeschrittene Themen	505
-------------------------------	-----

TEIL IV

Die Standardbibliothek	621
------------------------------	-----

TEIL V

Über den Standard hinaus	985
--------------------------------	-----

Inhalt

Vorwort 23

Vorwort zur 1. Auflage 25

TEIL I Grundlagen

1 Das C++-Handbuch 29

1.1 Neu und modern 30

1.2 »Dan«-Kapitel 30

1.3 Darstellung in diesem Buch 31

1.4 Verwendete Formatierungen 31

1.5 Sorry for my Denglish 32

2 Programmieren in C++ 35

2.1 Übersetzen 36

2.2 Übersetzungsphasen 36

2.3 Aktuelle Compiler 38

2.3.1 Gnu C++ 38

2.3.2 Clang++ der LLVM 38

2.3.3 Microsoft Visual Studio 38

2.3.4 Compiler im Container 39

2.4 Entwicklungsumgebungen 40

2.5 Die Kommandozeile unter Ubuntu 42

2.5.1 Ein Programm erstellen 42

2.5.2 Automatisieren mit Makefile 44

2.6 Die IDE »Microsoft Visual Studio Community« unter Windows 45

2.7 Das Beispielprogramm beschleunigen 48

3 C++ für Umsteiger 49

4	Die Grundbausteine von C++	57
4.1	Kommentare	60
4.2	Die »include«-Direktive	60
4.3	Die Standardbibliothek	60
4.4	Die Funktion »main()«	61
4.5	Typen	61
4.6	Variablen	62
4.7	Initialisierung	62
4.8	Ausgabe auf der Konsole	63
4.9	Anweisungen	63
4.10	Ohne Eile erklärt	65
4.10.1	Leerräume, Bezeichner und Token	66
4.10.2	Kommentare	68
4.10.3	Funktionen und Argumente	68
4.10.4	Seiteneffekt-Operatoren	69
4.10.5	Die »main«-Funktion	71
4.10.6	Anweisungen	72
4.10.7	Ausdrücke	75
4.10.8	Zuweisungen	76
4.10.9	Typen	77
4.10.10	Variablen – Deklaration, Definition und Initialisierung	83
4.10.11	Initialisieren mit »auto«	85
4.10.12	Details zur »include«-Direktive und »include« direkt	86
4.10.13	Eingabe und Ausgabe	88
4.10.14	Der Namensraum »std«	89
4.11	Operatoren	91
4.11.1	Operatoren und Operanden	91
4.11.2	Überblick über Operatoren	92
4.11.3	Arithmetische Operatoren	93
4.11.4	Bitweise Arithmetik	94
4.11.5	Zusammengesetzte Zuweisung	97
4.11.6	Post- und Präinkrement sowie Post- und Prädekrement	98
4.11.7	Relationale Operatoren	98
4.11.8	Logische Operatoren	99
4.11.9	Pointer- und Dereferenzierungsoperatoren	101
4.11.10	Besondere Operatoren	102
4.11.11	Funktionsähnliche Operatoren	103
4.11.12	Operatorreihenfolge	104

- 4.12 Eingebaute Datentypen 105
 - 4.12.1 Übersicht 106
 - 4.12.2 Eingebaute Datentypen initialisieren 108
 - 4.12.3 Ganzzahlen 109
 - 4.12.4 Fließkommazahlen 121
 - 4.12.5 Wahrheitswerte 134
 - 4.12.6 Zeichentypen 136
 - 4.12.7 Komplexe Zahlen 138

5 Guter Code, 1. Dan: Lesbar programmieren 143

- 5.1 Kommentare 144
- 5.2 Dokumentation 144
- 5.3 Einrückungen und Zeilenlänge 145
- 5.4 Zeilen pro Funktion und Datei 146
- 5.5 Klammern und Leerzeichen 147
- 5.6 Namen 148

6 Höhere Datentypen 151

- 6.1 Der Zeichenkettentyp »string« 152
 - 6.1.1 Initialisierung 153
 - 6.1.2 Funktionen und Methoden 154
 - 6.1.3 Andere Stringtypen 155
 - 6.1.4 Nur zur Ansicht: string_view 156
- 6.2 Streams 158
 - 6.2.1 Eingabe- und Ausgabeoperatoren 158
 - 6.2.2 »getline« 160
 - 6.2.3 Dateien für die Ein- und Ausgabe 160
 - 6.2.4 Manipulatoren 162
 - 6.2.5 Der Manipulator »endl« 164
- 6.3 Behälter und Zeiger 164
 - 6.3.1 Container 164
 - 6.3.2 Parametrisierte Typen 165
- 6.4 Die einfachen Sequenzcontainer 166
 - 6.4.1 »array« 166
 - 6.4.2 »vector« 169

6.5	Algorithmen	171
6.6	Zeiger und C-Arrays	172
6.6.1	Zeigertypen	172
6.6.2	C-Arrays	172

7 Funktionen

173

7.1	Deklaration und Definition einer Funktion	174
7.2	Funktionsstyp	175
7.3	Funktionen verwenden	175
7.4	Eine Funktion definieren	177
7.5	Mehr zu Parametern	178
7.5.1	Call-by-Value	178
7.5.2	Call-by-Reference	179
7.5.3	Konstante Referenzen	180
7.5.4	Aufruf als Wert, Referenz oder konstante Referenz?	181
7.6	Funktionskörper	182
7.7	Parameter umwandeln	184
7.8	Funktionen überladen	186
7.9	Default-Parameter	188
7.10	Beliebig viele Argumente	190
7.11	Alternative Schreibweise zur Funktionsdeklaration	190
7.12	Spezialitäten	191
7.12.1	»noexcept«	191
7.12.2	Inline-Funktionen	192
7.12.3	»constexpr«	192
7.12.4	Gelöschte Funktionen	193
7.12.5	Spezialitäten bei Klassenmethoden	193

8 Anweisungen im Detail

195

8.1	Der Anweisungsblock	198
8.2	Die leere Anweisung	200
8.3	Deklarationsanweisung	201
8.3.1	Strukturiertes Binden	202

- 8.4 Die Ausdrucksanweisung 203
- 8.5 Die »if«-Anweisung 204
 - 8.5.1 »if« mit Initialisierer 207
 - 8.5.2 Compilezeit »if« 207
- 8.6 Die »while«-Schleife 208
- 8.7 Die »do-while«-Schleife 210
- 8.8 Die »for«-Schleife 211
- 8.9 Die bereichsbasierte »for«-Schleife 213
- 8.10 Die »switch«-Verzweigung 215
- 8.11 Die »break«-Anweisung 219
- 8.12 Die »continue«-Anweisung 220
- 8.13 Die »return«-Anweisung 221
- 8.14 Die »goto«-Anweisung 222
- 8.15 Der »try-catch«-Block und »throw« 224
- 8.16 Zusammenfassung 225

9 Ausdrücke im Detail 227

- 9.1 Berechnungen und Seiteneffekte 228
- 9.2 Arten von Ausdrücken 229
- 9.3 Literale 230
- 9.4 Bezeichner 231
- 9.5 Klammern 231
- 9.6 Funktionsaufruf und Indexzugriff 232
- 9.7 Zuweisung 232
- 9.8 Typumwandlung 234

10 Fehlerbehandlung 237

- 10.1 Fehlerbehandlung mit Fehlercodes 239
- 10.2 Was ist eine Ausnahme? 242
 - 10.2.1 Ausnahmen auslösen und behandeln 243
 - 10.2.2 Aufrufstapel abwickeln 244

- 10.3 Kleinere Fehlerbehandlungen 245
- 10.4 Weiterwerfen – »rethrow« 245
- 10.5 Die Reihenfolge im »catch« 246
 - 10.5.1 Kein »finally« 247
 - 10.5.2 Exceptions der Standardbibliothek 247
- 10.6 Typen für Exceptions 248
- 10.7 Wenn eine Exception aus »main« herausfällt 249

11 Guter Code, 2. Dan: Modularisierung 251

- 11.1 Programm, Bibliothek, Objektdatei 251
- 11.2 Bausteine 252
- 11.3 Trennen der Funktionalitäten 253
- 11.4 Ein modulares Beispielprojekt 255
 - 11.4.1 Namensräume 257
 - 11.4.2 Implementierung 258
 - 11.4.3 Die Bibliothek nutzen 264
- 11.5 Spezialthema: Unity-Builds 265

TEIL II Objektorientierte Programmierung und mehr

12 Von der Struktur zur Klasse 269

- 12.1 Initialisierung 271
- 12.2 Rückgabe eigener Typen 272
- 12.3 Methoden statt Funktionen 273
- 12.4 Das bessere »drucke« 276
- 12.5 Eine Ausgabe wie jede andere 278
- 12.6 Methoden inline definieren 279
- 12.7 Implementierung und Definition trennen 280
- 12.8 Initialisierung per Konstruktor 281
 - 12.8.1 Member-Defaultwerte in der Deklaration 284
 - 12.8.2 Konstruktor-Delegation 284
 - 12.8.3 Defaultwerte für die Konstruktorparameter 285
 - 12.8.4 »init«-Methode nicht im Konstruktor aufrufen 287
 - 12.8.5 Exceptions im Konstruktor 288

- 12.9 Struktur oder Klasse? 288
 - 12.9.1 Kapselung 289
 - 12.9.2 »public« und »private«, Struktur und Klasse 290
 - 12.9.3 Daten mit »struct«, Verhalten mit »class« 290
 - 12.9.4 Initialisierung von Typen mit privaten Daten 291
- 12.10 Zwischenergebnis 292
- 12.11 Verwendung eigener Datentypen 293
 - 12.11.1 Klassen als Werte verwenden 295
 - 12.11.2 Konstruktoren nutzen 298
 - 12.11.3 Typumwandlungen 299
 - 12.11.4 Kapseln und entkapseln 301
 - 12.11.5 Typen lokal einen Namen geben 305
- 12.12 Typinferenz mit »auto« 308
- 12.13 Eigene Klassen in Standardcontainern 311

13 Namensräume und Qualifizierer 315

- 13.1 Der Namensraum »std« 315
- 13.2 Anonymer Namensraum 319
- 13.3 »static« macht lokal 321
- 13.4 »static« teilt gern 322
- 13.5 »static« macht dauerhaft 325
 - 13.5.1 »inline namespace« 327
- 13.6 Zusammenfassung 328
- 13.7 »const« 329
 - 13.7.1 Const-Parameter 330
 - 13.7.2 Const-Methoden 331
 - 13.7.3 Const-Variablen 332
 - 13.7.4 Const-Rückgaben 333
 - 13.7.5 »const« zusammen mit »static« 338
 - 13.7.6 Noch konstanter mit »constexpr« 338
 - 13.7.7 Un-Const mit »mutable« 342
 - 13.7.8 Const-Korrektheit 342
 - 13.7.9 Zusammenfassung 344
- 13.8 Flüchtig mit »volatile« 344

14	Guter Code, 3. Dan: Testen	347
14.1	Arten des Tests	347
14.1.1	Refactoring	349
14.1.2	Unittests	350
14.1.3	Sozial oder solitär	351
14.1.4	Doppelgänger	353
14.1.5	Suites	354
14.2	Frameworks	355
14.2.1	Arrange, Act, Assert	357
14.2.2	Frameworks zur Auswahl	359
14.3	Boost.Test	359
14.4	Hilfsmakros für Assertions	363
14.5	Ein Beispielprojekt mit Unittests	366
14.5.1	Privates und öffentliches Testen	368
14.5.2	Ein automatisches Testmodul	369
14.5.3	Test kompilieren	371
14.5.4	Die Testsuite selbst zusammenbauen	372
14.5.5	Testen von Privatem	376
14.5.6	Parametrisierte Tests	377
15	Vererbung	379
15.1	Beziehungen	380
15.1.1	Hat-ein-Komposition	380
15.1.2	Hat-ein-Aggregation	380
15.1.3	Ist-ein-Vererbung	381
15.1.4	Ist-Instanz-von versus Ist-ein-Beziehung	382
15.2	Vererbung in C++	383
15.3	Hat-ein versus ist-ein	384
15.4	Gemeinsamkeiten finden	384
15.5	Abgeleitete Typen erweitern	387
15.6	Methoden überschreiben	388
15.7	Wie Methoden funktionieren	389
15.8	Virtuelle Methoden	390
15.9	Konstruktoren in Klassenhierarchien	392

15.10	Typumwandlung in Klassenhierarchien	394
15.10.1	Die Vererbungshierarchie aufwärts umwandeln	394
15.10.2	Die Vererbungshierarchie abwärts umwandeln	394
15.10.3	Referenzen behalten auch die Typinformation	395
15.11	Wann virtuell?	396
15.12	Andere Designs zur Erweiterbarkeit	397

16

Der Lebenszyklus von Klassen

399

16.1	Erzeugung und Zerstörung	400
16.2	Temporary: kurzlebige Werte	402
16.3	Der Destruktor zum Konstruktor	404
16.3.1	Kein Destruktor nötig	406
16.3.2	Ressourcen im Destruktor	406
16.4	Yoda-Bedingung	408
16.5	Konstruktion, Destruktion und Exceptions	410
16.6	Kopieren	411
16.7	Zuweisungsoperator	414
16.8	Streichen von Methoden	417
16.9	Verschiebeoperationen	419
16.9.1	Was der Compiler generiert	423
16.10	Operatoren	424
16.11	Eigene Operatoren in einem Datentyp	427
16.12	Besondere Klassenformen	432
16.12.1	Abstrakte Klassen und Methoden	432
16.12.2	Aufzählungsklassen	434

17

**Guter Code, 4. Dan:
Sicherheit, Qualität und Nachhaltigkeit**

437

17.1	Die Nullerregel	437
17.1.1	Die großen Fünf	437
17.1.2	Hilfskonstrukt per Verbot	438
17.1.3	Die Nullerregel und ihr Einsatz	439
17.1.4	Ausnahmen von der Nullerregel	440

17.2	RAII – Resource Acquisition Is Initialization	443
17.2.1	Ein Beispiel mit C	443
17.2.2	Besitzende Raw-Pointer	445
17.2.3	Von C nach C++	446
17.2.4	Es muss nicht immer eine Exception sein	449
17.2.5	Mehrere Konstruktoren	450
17.2.6	Mehrphasige Initialisierung	450
17.2.7	Definieren, wo es gebraucht wird	450
17.2.8	Nothrow-new	451

18 Spezielles für Klassen 453

18.1	Dürfen alles sehen – »friend«-Klassen	453
18.2	Non-public-Vererbung	457
18.2.1	Auswirkungen auf die Außenwelt	459
18.2.2	Nicht öffentliche Vererbung in der Praxis	461
18.3	Signaturklassen als Interfaces	463
18.4	Multiple Vererbung	467
18.4.1	Multiple Vererbung in der Praxis	469
18.4.2	Achtung bei Typumwandlungen von Zeigern	472
18.4.3	Das Beobachter-Muster als praktisches Beispiel	475
18.5	Rautenförmige multiple Vererbung – »virtual« für Klassenhierarchien	476
18.6	Literale Datentypen – »constexpr« für Konstruktoren	480

19 Guter Code, 5. Dan: Klassisches objektorientiertes Design 483

19.1	Objekte in C++	485
19.2	Objektorientiert designen	486
19.2.1	SOLID	486
19.2.2	Seien Sie nicht STUPID	504

TEIL III Fortgeschrittene Themen

20 Zeiger 507

20.1 Adressen 508

20.2 Zeiger 509

20.3 Gefahren von Aliasing 511

20.4 Heapspeicher und Stapelspeicher 513

 20.4.1 Der Stapel 513

 20.4.2 Der Heap 515

20.5 Smarte Pointer 516

 20.5.1 »unique_ptr« 518

 20.5.2 »shared_ptr« 522

20.6 Rohe Zeiger 526

20.7 C-Arrays 530

 20.7.1 Rechnen mit Zeigern 531

 20.7.2 Verfall von C-Arrays 532

 20.7.3 Dynamische C-Arrays 534

 20.7.4 Zeichenkettenliterale 535

20.8 Iteratoren 536

20.9 Zeiger als Iteratoren 538

20.10 Zeiger im Container 538

20.11 Die Ausnahme: wann das Wegräumen nicht nötig ist 539

21 Makros 541

21.1 Der Präprozessor 542

21.2 Vorsicht vor fehlenden Klammern 546

21.3 Vorsicht vor Mehrfachausführung 547

21.4 Typvariabilität von Makros 548

21.5 Zusammenfassung 551

22	Schnittstelle zu C	553
22.1	Mit Bibliotheken arbeiten	554
22.2	C-Header	555
22.3	C-Ressourcen	558
22.4	»void«-Pointer	559
22.5	Daten lesen	559
22.6	Das Hauptprogramm	561
22.7	Zusammenfassung	561
23	Templates	563
23.1	Funktionstemplates	564
23.1.1	Überladung	565
23.1.2	Ein Typ als Parameter	566
23.1.3	Funktionskörper eines Funktionstemplates	566
23.1.4	Zahlen als Templateparameter	569
23.1.5	Viele Funktionen	570
23.1.6	Parameter mit Extras	570
23.1.7	Methodentemplates sind auch nur Funktionstemplates	573
23.2	Funktionstemplates in der Standardbibliothek	574
23.2.1	Iteratoren statt Container als Templateparameter	575
23.2.2	Beispiel: Informationen über Zahlen	577
23.3	Eine Klasse als Funktion	578
23.3.1	Werte für einen »function«-Parameter	579
23.3.2	C-Funktionspointer	580
23.3.3	Die etwas andere Funktion	582
23.3.4	Praktische Funktoren	585
23.3.5	Algorithmen mit Funktoren	587
23.3.6	Anonyme Funktionen alias Lambda-Ausdrücke	587
23.3.7	Templatefunktionen ohne »template«, aber mit »auto«	592
23.4	Templateklassen	593
23.4.1	Klassentemplates implementieren	593
23.4.2	Methoden von Klassentemplates implementieren	594
23.4.3	Objekte aus Klassentemplates erzeugen	596
23.4.4	Klassentemplates mit mehreren formalen Datentypen	600
23.4.5	Klassentemplates mit Non-Type-Parameter	601
23.4.6	Klassentemplates mit Default	603
23.4.7	Klassentemplates spezialisieren	605

23.5	Templates mit variabler Argumentanzahl	607
23.6	Eigene Literale	611
23.6.1	Was sind Literale?	612
23.6.2	Namensregeln	613
23.6.3	Phasenweise	613
23.6.4	Überladungsvarianten	614
23.6.5	Benutzerdefiniertes Literal mittels Template	615
23.6.6	Roh oder gekocht	618
23.6.7	Automatisch zusammengefügt	619
23.6.8	Unicodeliterale	620

TEIL IV Die Standardbibliothek

24	Container	623
<hr/>		
24.1	Grundlagen	624
24.1.1	Wiederkehrend	624
24.1.2	Abstrakt	625
24.1.3	Operationen	626
24.1.4	Komplexität	627
24.1.5	Container und ihre Iteratoren	629
24.1.6	Algorithmen	631
24.2	Iteratoren-Grundlagen	631
24.2.1	Iteratoren aus Containern	632
24.2.2	Mehr Funktionalität mit Iteratoren	634
24.3	Allokatoren: Speicherfragen	635
24.4	Containergemeinsamkeiten	638
24.5	Ein Überblick über die Standardcontainerklassen	639
24.5.1	Typalias der Container	640
24.6	Die sequenziellen Containerklassen	643
24.6.1	Gemeinsamkeiten und Unterschiede	645
24.6.2	Methoden von Sequenzcontainern	647
24.6.3	»vector«	649
24.6.4	»array«	663
24.6.5	»deque«	669
24.6.6	»list«	672
24.6.7	»forward_list«	675
24.7	Assoziativ und geordnet	680
24.7.1	Gemeinsamkeiten und Unterschiede	681
24.7.2	Methoden der geordneten assoziativen Container	682

24.7.3	»set«	684
24.7.4	»map«	697
24.7.5	»multiset«	704
24.7.6	»multimap«	708
24.8	Nur assoziativ und nicht garantiert	712
24.8.1	Gemeinsamkeiten und Unterschiede	717
24.8.2	Methoden der ungeordneten assoziativen Container	719
24.8.3	»unordered_set«	720
24.8.4	»unordered_map«	729
24.8.5	»unordered_multiset«	733
24.8.6	»unordered_multimap«	739
24.9	Containeradapter	742
24.10	Sonderfälle: »string«, »basic_string« und »vector<char>«	743
24.11	Sonderfälle: »vector<bool>«, »array<bool,n>« und »bitset<n>«	744
24.11.1	Dynamisch und kompakt: »vector<bool>«	744
24.11.2	Statisch: »array<bool,n>« und »bitset<n>«	744
24.12	Sonderfall: Value-Array mit »valarray<>«	747
25	Containerunterstützung	757
<hr/>		
25.1	Algorithmen	757
25.2	Iteratoren	758
25.3	Iteratoradapter	759
25.4	Algorithmen der Standardbibliothek	760
25.5	Parallele Ausführung	762
25.6	Liste der Algorithmusfunktionen	765
25.7	Berechnungen auf Containern mit »<numeric>«	780
25.8	Kopie statt Zuweisung – Werte in uninitialisierten Speicherbereichen	785
25.9	Eigene Algorithmen	787
26	Guter Code, 6. Dan: Für jede Aufgabe der richtige Container	791
<hr/>		
26.1	Alle Container nach Aspekten sortiert	791
26.1.1	Wann ist ein »vector« nicht die beste Wahl?	791
26.1.2	Immer sortiert: »set«, »map«, »multiset« und »multimap«	792

26.1.3	Im Speicher hintereinander: »vector«, »array«	793
26.1.4	Einfügung billig: »list«	794
26.1.5	Wenig Speicheroverhead: »vector«, »array«	794
26.1.6	Größe dynamisch: alle außer »array«	795
26.2	Rezepte für Container	796
26.2.1	Zwei Phasen? »vector« als guter »set«-Ersatz	797
26.2.2	Den Inhalt eines Containers auf einem Stream ausgeben	798
26.2.3	So statisch ist »array« gar nicht	799
26.3	Iteratoren sind mehr als nur Zeiger	802
26.4	Algorithmen je nach Container unterschiedlich implementieren	804

27

Streams und Dateien

807

27.1	Ein- und Ausgabekonzept	807
27.2	Globale, vordefinierte Standardstreams	808
27.3	Methoden für die Aus- und Eingabe von Streams	810
27.3.1	Methoden für die unformatierte Ausgabe	810
27.3.2	Methoden für die (unformatierte) Eingabe	812
27.4	Fehlerbehandlung und Zustand von Streams	814
27.4.1	Methoden für die Behandlung von Fehlern bei Streams	815
27.5	Streams manipulieren und formatieren	818
27.5.1	Manipulatoren	818
27.5.2	Eigene Manipulatoren ohne Argumente erstellen	824
27.5.3	Eigene Manipulatoren mit Argumenten erstellen	825
27.5.4	Format-Flags direkt ändern	826
27.6	Streams für die Dateiein- und Dateiausgabe	829
27.6.1	Die Streams »ifstream«, »ofstream« und »fstream«	830
27.6.2	Verbindung zu einer Datei herstellen	830
27.6.3	Lesen und Schreiben	835
27.6.4	Wahlfreier Zugriff	842
27.7	Streams für Strings	843
27.8	Streampuffer	848
27.9	»filesystem«	851

28	Standardbibliothek – Extras	855
28.1	»pair« und »tuple«	855
28.1.1	Mehrere Werte zurückgeben	856
28.2	Reguläre Ausdrücke	863
28.2.1	Matchen und Suchen	864
28.2.2	Ergebnis und Teile davon	864
28.2.3	Gefundenes Ersetzen	865
28.2.4	Reich an Varianten	865
28.2.5	Iteratoren	866
28.2.6	Matches	866
28.2.7	Optionen	866
28.2.8	Geschwindigkeit	867
28.2.9	Standardsyntax leicht gekürzt	868
28.2.10	Anmerkungen zu regulären Ausdrücken in C++	869
28.3	Zufall	872
28.3.1	Einen Würfel werfen	873
28.3.2	Echter Zufall	875
28.3.3	Andere Generatoren	875
28.3.4	Verteilungen	877
28.4	Mathematisches	881
28.4.1	Brüche und Zeiten – »<ratio>« und »<chrono>«	881
28.4.2	Vordefinierte Suffixe für benutzerdefinierte Literale	895
28.5	Systemfehlerbehandlung mit »system_error«	897
28.5.1	»error_code« und »error_condition«	899
28.5.2	Fehlerkategorien	903
28.5.3	Eigene Fehlercodes	903
28.5.4	»system_error«-Exception	905
28.6	Laufzeittypinformationen – »<typeinfo>« und »<typeid>«	906
28.7	Hilfsklassen rund um Funktoren – »<functional>«	910
28.7.1	Funktionsobjekte	911
28.7.2	Funktionsgeneratoren	915
28.8	»optional« für einen oder keinen Wert	917
28.9	»variant« für einen von mehreren Typen	918
28.10	»any« hält jeden Typ	920
28.11	Spezielle mathematische Funktionen	921
28.12	Schnelle Umwandlung mit »charconv«	921

29	Threads – Programmieren mit Mehrläufigkeit	925
29.1	C++-Threading-Grundlagen	926
29.1.1	Einer Threadfunktion Parameter übergeben	932
29.1.2	Einen Thread verschieben	937
29.1.3	Wie viele Threads starten?	939
29.1.4	Welcher Thread bin ich?	941
29.2	Gemeinsame Daten	942
29.2.1	Daten mit Mutexen schützen	943
29.2.2	Data Races	945
29.2.3	Interface-Design für Multithreading	947
29.2.4	Sperren können zum Patt führen	951
29.2.5	Flexibleres Sperren mit »unique_lock«	954
29.3	Andere Möglichkeiten zur Synchronisation	955
29.3.1	Nur einmal aufrufen mit »once_flag« und »call_once«	955
29.3.2	Sperren zählen mit »recursive_mutex«	957
29.4	Im eigenen Speicher mit »thread_local«	959
29.5	Mit »condition_variable« auf Ereignisse warten	960
29.6	Einmal warten mit »future«	965
29.6.1	Ausnahmebehandlung bei »future«	970
29.6.2	»promise«	972
29.7	Atomics	976
29.8	Zusammenfassung	981

TEIL V Über den Standard hinaus

30	Guter Code, 7. Dan: Richtlinien	987
30.1	Guideline Support Library	988
30.2	C++ Core Guidelines	989
30.2.1	Motivation	990
30.2.2	Typsicherheit	991
30.2.3	Nutzen Sie RAII	992
30.2.4	Klassenhierarchien	995
30.2.5	Generische Programmierung	998
30.2.6	Lassen Sie sich nicht von Anachronismen verwirren	1000

31	GUI-Programmierung mit Qt	1003
31.1	Ein erstes Miniprogramm	1007
31.1.1	Kurze Übersicht über die Oberfläche von Qt Creator	1008
31.1.2	Ein einfaches Projekt erstellen	1009
31.2	Objektbäume und Besitz	1018
31.3	Signale und Slots	1019
31.3.1	Verbindung zwischen Signal und Slot herstellen	1020
31.3.2	Signal und Slot mithilfe der Qt-Referenz ermitteln	1022
31.4	Klassenhierarchie von Qt	1039
31.4.1	Basisklasse »QObject«	1039
31.4.2	Weitere wichtige Klassen	1039
31.5	Eigene Widgets mit dem Qt Designer erstellen	1042
31.6	Widgets anordnen	1048
31.6.1	Grundlegende Widgets für das Layout	1048
31.7	Dialoge erstellen mit »QDialog«	1052
31.8	Vorgefertigte Dialoge von Qt	1059
31.8.1	»QMessageBox« – der klassische Nachrichtendialog	1060
31.8.2	»QFileDialog« – der Dateiauswahldialog	1061
31.8.3	»QInputDialog« – Dialog zur Eingabe von Daten	1063
31.8.4	Weitere Dialoge	1067
31.9	Eigenen Dialog mit dem Qt Designer erstellen	1067
31.10	Grafische Bedienelemente von Qt (Qt-Widgets)	1083
31.10.1	Schaltflächen (Basisklasse »QAbstractButton«)	1083
31.10.2	Container-Widgets (Behälter-Widgets)	1085
31.10.3	Widgets zur Zustandsanzeige	1086
31.10.4	Widgets zur Eingabe	1087
31.10.5	Onlinehilfen	1088
31.11	Anwendungen in einem Hauptfenster	1089
31.11.1	Die Klasse für das Hauptfenster »QMainWindow«	1089
31.12	Zusammenfassung	1100
	Cheat Sheet	1104
	Index	1107