

**C++**

Das umfassende Handbuch

# **DAS INHALTS- VERZEICHNIS**

» Hier geht's  
direkt  
zum Buch

# Inhalt

Vorwort .....	25
---------------	----

## TEIL I Grundlagen

<b>1 Das C++-Handbuch</b> .....	<b>31</b>
---------------------------------	-----------

---

1.1 Neu und modern .....	32
1.2 »Dan«-Kapitel .....	32
1.3 Darstellung in diesem Buch .....	33
1.4 Verwendete Formatierungen .....	33
1.5 Sorry for my Denglish .....	34

<b>2 Programmieren in C++</b> .....	<b>37</b>
-------------------------------------	-----------

---

2.1 Übersetzen .....	38
2.2 Übersetzungsphasen .....	39
2.3 Aktuelle Compiler .....	40
2.3.1 Gnu C++ .....	40
2.3.2 Clang++ von LLVM .....	40
2.3.3 Microsoft Visual C++ .....	41
2.3.4 Compiler im Container .....	41
2.4 Entwicklungsumgebungen .....	41
2.5 Die Kommandozeile unter Ubuntu .....	43
2.5.1 Ein Programm erstellen .....	44
2.5.2 Automatisieren mit Makefile .....	46
2.6 Die IDE »Visual Studio Code« unter Windows .....	47
2.7 Das Beispielprogramm beschleunigen .....	54

**3 C++ für Umsteiger** 55

---

**4 Die Grundbausteine von C++** 63

---

- 4.1 Ein schneller Überblick** ..... 66
  - 4.1.1 Kommentare ..... 66
  - 4.1.2 Die »include«-Direktive ..... 67
  - 4.1.3 Die Standardbibliothek ..... 67
  - 4.1.4 Die Funktion »main()« ..... 67
  - 4.1.5 Typen ..... 68
  - 4.1.6 Variablen ..... 68
  - 4.1.7 Initialisierung ..... 69
  - 4.1.8 Ausgabe auf der Konsole ..... 70
  - 4.1.9 Anweisungen ..... 70
- 4.2 Ohne Eile erklärt** ..... 71
  - 4.2.1 Leerräume, Bezeichner und Token ..... 73
  - 4.2.2 Kommentare ..... 74
  - 4.2.3 Funktionen und Argumente ..... 75
  - 4.2.4 Seiteneffekt-Operatoren ..... 76
  - 4.2.5 Die »main«-Funktion ..... 78
  - 4.2.6 Anweisungen ..... 79
  - 4.2.7 Ausdrücke ..... 82
  - 4.2.8 Zuweisungen ..... 84
  - 4.2.9 Typen ..... 85
  - 4.2.10 Variablen – Deklaration, Definition und Initialisierung ..... 91
  - 4.2.11 Initialisieren mit »auto« ..... 93
  - 4.2.12 Details zur »include«-Direktive und »include« direkt ..... 95
  - 4.2.13 Module ..... 96
  - 4.2.14 Eingabe und Ausgabe ..... 97
  - 4.2.15 Der Namensraum »std« ..... 98
- 4.3 Operatoren** ..... 100
  - 4.3.1 Operatoren und Operanden ..... 101
  - 4.3.2 Überblick über Operatoren ..... 102
  - 4.3.3 Arithmetische Operatoren ..... 103
  - 4.3.4 Bitweise Arithmetik ..... 104
  - 4.3.5 Zusammengesetzte Zuweisung ..... 107
  - 4.3.6 Post- und Präinkrement sowie Post- und Prädecrement ..... 108
  - 4.3.7 Relationale Operatoren ..... 108

---

4.3.8	Logische Operatoren .....	109
4.3.9	Pointer- und Dereferenzierungsoperatoren .....	111
4.3.10	Besondere Operatoren .....	112
4.3.11	Funktionsähnliche Operatoren .....	114
4.3.12	Operatorreihenfolge .....	115
<b>4.4</b>	<b>Eingebaute Datentypen</b> .....	<b>116</b>
4.4.1	Übersicht .....	117
4.4.2	Eingebaute Datentypen initialisieren .....	119
4.4.3	Ganzzahlen .....	119
4.4.4	Fließkommazahlen .....	134
4.4.5	Wahrheitswerte .....	149
4.4.6	Zeichentypen .....	151
4.4.7	Komplexe Zahlen .....	154
<b>4.5</b>	<b>Undefiniertes und unspezifiziertes Verhalten</b> .....	<b>157</b>
<b>5</b>	<b>Guter Code, 1. Dan: Lesbar programmieren</b> .....	<b>159</b>
<hr/>		
<b>5.1</b>	<b>Kommentare</b> .....	<b>160</b>
<b>5.2</b>	<b>Dokumentation</b> .....	<b>160</b>
<b>5.3</b>	<b>Einrückungen und Zeilenlänge</b> .....	<b>161</b>
<b>5.4</b>	<b>Zeilen pro Funktion und Datei</b> .....	<b>162</b>
<b>5.5</b>	<b>Klammern und Leerzeichen</b> .....	<b>163</b>
<b>5.6</b>	<b>Namen</b> .....	<b>164</b>
<b>6</b>	<b>Höhere Datentypen</b> .....	<b>167</b>
<hr/>		
<b>6.1</b>	<b>Der Zeichenkettentyp »string«</b> .....	<b>168</b>
6.1.1	Initialisierung .....	169
6.1.2	Funktionen und Methoden .....	171
6.1.3	Andere Stringtypen .....	172
6.1.4	Nur zur Ansicht: »string_view« .....	173
<b>6.2</b>	<b>Streams</b> .....	<b>174</b>
6.2.1	Eingabe- und Ausgabeoperatoren .....	175
6.2.2	»getline« .....	177
6.2.3	Dateien für die Ein- und Ausgabe .....	177

6.2.4	Manipulatoren .....	179
6.2.5	Der Manipulator »endl« .....	181
<b>6.3</b>	<b>Behälter und Zeiger</b> .....	<b>181</b>
6.3.1	Container .....	181
6.3.2	Parametrisierte Typen .....	182
<b>6.4</b>	<b>Die einfachen Sequenzcontainer</b> .....	<b>183</b>
6.4.1	»array« .....	184
6.4.2	»vector« .....	186
<b>6.5</b>	<b>Algorithmen</b> .....	<b>189</b>
<b>6.6</b>	<b>Zeiger und C-Arrays</b> .....	<b>189</b>
6.6.1	Zeigertypen .....	190
6.6.2	C-Arrays .....	190

## **7 Funktionen** 191

---

<b>7.1</b>	<b>Deklaration und Definition einer Funktion</b> .....	<b>192</b>
<b>7.2</b>	<b>Funktionstyp</b> .....	<b>193</b>
<b>7.3</b>	<b>Funktionen verwenden</b> .....	<b>194</b>
<b>7.4</b>	<b>Eine Funktion definieren</b> .....	<b>195</b>
<b>7.5</b>	<b>Mehr zu Parametern</b> .....	<b>197</b>
7.5.1	Call-by-Value .....	197
7.5.2	Call-by-Reference .....	198
7.5.3	Konstante Referenzen .....	199
7.5.4	Aufruf als Wert, Referenz oder konstante Referenz? .....	200
<b>7.6</b>	<b>Funktionskörper</b> .....	<b>201</b>
<b>7.7</b>	<b>Parameter umwandeln</b> .....	<b>203</b>
<b>7.8</b>	<b>Funktionen überladen</b> .....	<b>205</b>
<b>7.9</b>	<b>Defaultparameter</b> .....	<b>208</b>
<b>7.10</b>	<b>Beliebig viele Argumente</b> .....	<b>209</b>
<b>7.11</b>	<b>Alternative Schreibweise zur Funktionsdeklaration</b> .....	<b>210</b>
<b>7.12</b>	<b>Spezialitäten</b> .....	<b>211</b>
7.12.1	»noexcept« .....	211
7.12.2	Inline-Funktionen .....	212
7.12.3	»constexpr« .....	212

7.12.4	Gelöschte Funktionen .....	213
7.12.5	Spezialitäten bei Klassenmethoden .....	213

## **8 Anweisungen im Detail** 215

---

<b>8.1</b>	<b>Der Anweisungsblock</b> .....	218
8.1.1	Frei stehende Blöcke und Gültigkeit von Variablen .....	219
<b>8.2</b>	<b>Die leere Anweisung</b> .....	221
<b>8.3</b>	<b>Deklarationsanweisung</b> .....	221
8.3.1	Strukturiertes Binden .....	223
<b>8.4</b>	<b>Die Ausdrucksanweisung</b> .....	224
<b>8.5</b>	<b>Die »if«-Anweisung</b> .....	224
8.5.1	»if« mit Initialisierer .....	227
8.5.2	Compilezeit »if« .....	228
<b>8.6</b>	<b>Die »while«-Schleife</b> .....	229
<b>8.7</b>	<b>Die »do-while«-Schleife</b> .....	231
<b>8.8</b>	<b>Die »for«-Schleife</b> .....	232
<b>8.9</b>	<b>Die bereichsbasierte »for«-Schleife</b> .....	234
<b>8.10</b>	<b>Die »switch«-Verzweigung</b> .....	236
<b>8.11</b>	<b>Die »break«-Anweisung</b> .....	240
<b>8.12</b>	<b>Die »continue«-Anweisung</b> .....	241
<b>8.13</b>	<b>Die »return«-Anweisung</b> .....	242
<b>8.14</b>	<b>Die »goto«-Anweisung</b> .....	243
<b>8.15</b>	<b>Der »try-catch«-Block und »throw«</b> .....	245
<b>8.16</b>	<b>Zusammenfassung</b> .....	247

## **9 Ausdrücke im Detail** 249

---

<b>9.1</b>	<b>Berechnungen und Seiteneffekte</b> .....	250
<b>9.2</b>	<b>Arten von Ausdrücken</b> .....	251
<b>9.3</b>	<b>Literale</b> .....	253

<b>9.4</b>	<b>Bezeichner</b> .....	253
<b>9.5</b>	<b>Klammern</b> .....	254
<b>9.6</b>	<b>Funktionsaufruf und Indexzugriff</b> .....	254
<b>9.7</b>	<b>Zuweisung</b> .....	255
<b>9.8</b>	<b>Typumwandlung</b> .....	257

## **10 Fehlerbehandlung** 259

---

<b>10.1</b>	<b>Fehlerbehandlung mit Fehlercodes</b> .....	261
<b>10.2</b>	<b>Was ist eine Ausnahme?</b> .....	264
10.2.1	Ausnahmen auslösen und behandeln .....	266
10.2.2	Aufrufstapel abwickeln .....	266
<b>10.3</b>	<b>Kleinere Fehlerbehandlungen</b> .....	267
<b>10.4</b>	<b>Weiterwerfen – »rethrow«</b> .....	268
<b>10.5</b>	<b>Die Reihenfolge im »catch«</b> .....	268
10.5.1	Kein »finally« .....	270
10.5.2	Exceptions der Standardbibliothek .....	270
<b>10.6</b>	<b>Typen für Exceptions</b> .....	271
<b>10.7</b>	<b>Wenn eine Exception aus »main« herausfällt</b> .....	272

## **11 Guter Code, 2. Dan: Modularisierung** 273

---

<b>11.1</b>	<b>Programm, Bibliothek, Objektdatei</b> .....	273
<b>11.2</b>	<b>Bausteine</b> .....	274
<b>11.3</b>	<b>Trennen der Funktionalitäten</b> .....	275
<b>11.4</b>	<b>Ein modulares Beispielprojekt</b> .....	277
11.4.1	Namensräume .....	280
11.4.2	Implementierung .....	281
11.4.3	Die Bibliothek nutzen .....	287

## TEIL II Objektorientierte Programmierung und mehr

<b>12 Von der Struktur zur Klasse</b>	291
<b>12.1 Initialisierung</b>	294
<b>12.2 Rückgabe eigener Typen</b>	295
<b>12.3 Methoden statt Funktionen</b>	297
<b>12.4 Das bessere »drucke«</b>	300
<b>12.5 Eine Ausgabe wie jede andere</b>	302
<b>12.6 Methoden inline definieren</b>	303
<b>12.7 Implementierung und Definition trennen</b>	304
<b>12.8 Initialisierung per Konstruktor</b>	305
12.8.1 Member-Defaultwerte in der Deklaration	308
12.8.2 Konstruktor-Delegation	309
12.8.3 Defaultwerte für die Konstruktorparameter	310
12.8.4 »init«-Methode nicht im Konstruktor aufrufen	312
12.8.5 Exceptions im Konstruktor	313
<b>12.9 Struktur oder Klasse?</b>	313
12.9.1 Kapselung	314
12.9.2 »public« und »private«, Struktur und Klasse	315
12.9.3 Daten mit »struct«, Verhalten mit »class«	315
12.9.4 Initialisierung von Typen mit privaten Daten	316
<b>12.10 Zwischenergebnis</b>	318
<b>12.11 Eigene Datentypen verwenden</b>	318
12.11.1 Klassen als Werte verwenden	321
12.11.2 Konstruktoren nutzen	324
12.11.3 Typumwandlungen	324
12.11.4 Kapseln und entkapseln	326
12.11.5 Typen lokal einen Namen geben	332
<b>12.12 Typinferenz mit »auto«</b>	335
<b>12.13 Eigene Klassen in Standardcontainern</b>	339
12.13.1 Drei-Wege-Vergleich: der Spaceship-Operator	341



---

<b>13</b>	<b>Namensräume und Qualifizierer</b>	343
<hr/>		
13.1	Der Namensraum »std«	344
13.2	Anonymer Namensraum	347
13.3	»static« macht lokal	349
13.4	»static« teilt gern	350
13.5	Ferne Initialisierung oder »static inline«-Datenfelder	353
13.6	Garantiert zur Compilezeit initialisiert mit »constinit«	354
13.7	»static« macht dauerhaft	354
13.8	»inline namespace«	356
13.9	Zusammenfassung	358
13.10	»const«	358
13.10.1	Const-Parameter	359
13.10.2	Const-Methoden	361
13.10.3	Const-Variablen	362
13.10.4	Const-Rückgaben	364
13.10.5	»const« zusammen mit »static«	368
13.10.6	Noch konstanter mit »constexpr«	369
13.10.7	»if constexpr« für Übersetzungszeit-Entscheidungen	373
13.10.8	C++20: »consteval«	374
13.10.9	»if consteval«	376
13.10.10	Un-Const mit »mutable«	377
13.10.11	Const-Korrektheit	378
13.10.12	Zusammenfassung	379
13.11	Flüchtig mit »volatile«	380
<b>14</b>	<b>Guter Code, 3. Dan: Testen</b>	383
<hr/>		
14.1	Arten des Tests	383
14.1.1	Refactoring	385
14.1.2	Unittests	386
14.1.3	Sozial oder solitär	387
14.1.4	Doppelgänger	389
14.1.5	Suites	391

---

<b>14.2 Frameworks</b> .....	391
14.2.1 Arrange, Act, Assert .....	394
14.2.2 Frameworks zur Auswahl .....	395
<b>14.3 Boost.Test</b> .....	396
<b>14.4 Hilfsmakros für Assertions</b> .....	400
<b>14.5 Ein Beispielprojekt mit Unittests</b> .....	403
14.5.1 Privates und öffentliches Testen .....	405
14.5.2 Ein automatisches Testmodul .....	406
14.5.3 Test kompilieren .....	408
14.5.4 Die Testsuite selbst zusammenbauen .....	409
14.5.5 Testen von Privatem .....	414
14.5.6 Parametrisierte Tests .....	414
<b>15 Vererbung</b> .....	417

---

<b>15.1 Beziehungen</b> .....	418
15.1.1 Hat-ein-Komposition .....	418
15.1.2 Hat-ein-Aggregation .....	419
15.1.3 Ist-ein-Vererbung .....	420
15.1.4 Ist-Instanz-von versus Ist-ein-Beziehung .....	420
<b>15.2 Vererbung in C++</b> .....	421
<b>15.3 Hat-ein versus ist-ein</b> .....	422
<b>15.4 Gemeinsamkeiten finden</b> .....	422
<b>15.5 Abgeleitete Typen erweitern</b> .....	425
<b>15.6 Methoden überschreiben</b> .....	426
<b>15.7 Wie Methoden funktionieren</b> .....	427
<b>15.8 Virtuelle Methoden</b> .....	429
<b>15.9 Konstruktoren in Klassenhierarchien</b> .....	431
<b>15.10 Typumwandlung in Klassenhierarchien</b> .....	433
15.10.1 Die Vererbungshierarchie aufwärts umwandeln .....	433
15.10.2 Die Vererbungshierarchie abwärts umwandeln .....	433
15.10.3 Referenzen behalten auch die Typinformation .....	434
<b>15.11 Wann virtuell?</b> .....	434
<b>15.12 Andere Designs zur Erweiterbarkeit</b> .....	436

<b>16</b>	<b>Der Lebenszyklus von Klassen</b>	439
<hr/>		
16.1	Erzeugung und Zerstörung .....	440
16.2	Temporary: kurzlebige Werte .....	442
16.3	Der Destruktor zum Konstruktor .....	444
16.3.1	Kein Destruktor nötig .....	446
16.3.2	Ressourcen im Destruktor .....	446
16.4	Yoda-Bedingung .....	449
16.5	Konstruktion, Destruktion und Exceptions .....	450
16.6	Kopieren .....	452
16.7	Zuweisungsoperator .....	455
16.8	Streichen von Methoden .....	459
16.9	Verschiebeoperationen .....	461
16.9.1	Was der Compiler generiert .....	465
16.10	Operatoren .....	466
16.11	Eigene Operatoren in einem Datentyp .....	470
16.12	Besondere Klassenformen .....	478
16.12.1	Abstrakte Klassen und Methoden .....	478
16.12.2	Aufzählungsklassen .....	480
<b>17</b>	<b>Guter Code, 4. Dan: Sicherheit, Qualität und Nachhaltigkeit</b>	483
<hr/>		
17.1	Die Nullerregel .....	483
17.1.1	Die großen Fünf .....	483
17.1.2	Hilfskonstrukt per Verbot .....	484
17.1.3	Die Nullerregel und ihr Einsatz .....	485
17.1.4	Ausnahmen von der Nullerregel .....	486
17.1.5	Nullerregel, Dreierregel, Fünferregel, Viereinhalberregel .....	489
17.2	RAII – Resource Acquisition Is Initialization .....	489
17.2.1	Ein Beispiel mit C .....	490
17.2.2	Besitzende Raw-Pointer .....	492
17.2.3	Von C nach C++ .....	493
17.2.4	Es muss nicht immer eine Exception sein .....	495
17.2.5	Mehrere Konstruktoren .....	496

17.2.6	Mehrphasige Initialisierung .....	497
17.2.7	Definieren, wo es gebraucht wird .....	497
17.2.8	Nothrow-new .....	497

---

## 18 Spezielles für Klassen 499

---

<b>18.1</b>	<b>Dürfen alles sehen – »friend«-Klassen</b> .....	499
18.1.1	»friend class«-Beispiel .....	501
<b>18.2</b>	<b>Non-public-Vererbung</b> .....	504
18.2.1	Auswirkungen auf die Außenwelt .....	505
18.2.2	Nicht öffentliche Vererbung in der Praxis .....	508
<b>18.3</b>	<b>Signaturklassen als Interfaces</b> .....	510
<b>18.4</b>	<b>Multiple Vererbung</b> .....	514
18.4.1	Multiple Vererbung in der Praxis .....	516
18.4.2	Achtung bei Typumwandlungen von Zeigern .....	520
18.4.3	Das Beobachter-Muster als praktisches Beispiel .....	522
<b>18.5</b>	<b>Rautenförmige multiple Vererbung – »virtual« für Klassenhierarchien</b> .....	524
<b>18.6</b>	<b>Literale Datentypen – »constexpr« für Konstruktoren</b> .....	528

---

## 19 Guter Code, 5. Dan: Klassisches objektorientiertes Design 531

---

<b>19.1</b>	<b>Objekte in C++</b> .....	533
<b>19.2</b>	<b>Objektorientiert designen</b> .....	534
19.2.1	SOLID .....	535
19.2.2	Seien Sie nicht STUPID .....	553

## TEIL III Fortgeschrittene Themen

---

## 20 Zeiger 557

---

<b>20.1</b>	<b>Adressen</b> .....	558
<b>20.2</b>	<b>Zeiger</b> .....	560

<b>20.3 Gefahren von Aliasing</b> .....	562
<b>20.4 Heapspeicher und Stapelspeicher</b> .....	563
20.4.1 Der Stapel .....	563
20.4.2 Der Heap .....	565
<b>20.5 Smarte Pointer</b> .....	567
20.5.1 »unique_ptr« .....	569
20.5.2 »shared_ptr« .....	573
<b>20.6 Rohe Zeiger</b> .....	576
20.6.1 Eine Referenz können Sie nicht neu zuweisen .....	579
20.6.2 Sie können keine Referenzen in einem Container speichern .....	579
<b>20.7 C-Arrays</b> .....	582
20.7.1 Rechnen mit Zeigern .....	582
20.7.2 Verfall von C-Arrays .....	584
20.7.3 Dynamische C-Arrays .....	585
20.7.4 Zeichenkettenliterale .....	587
<b>20.8 Iteratoren</b> .....	588
<b>20.9 Zeiger als Iteratoren</b> .....	590
<b>20.10 Zeiger im Container</b> .....	590
<b>20.11 Die Ausnahme: wann das Wegräumen nicht nötig ist</b> .....	591

## **21 Makros** 595

---

<b>21.1 Der Präprozessor</b> .....	596
<b>21.2 Vorsicht vor fehlenden Klammern</b> .....	600
<b>21.3 Featuremakros</b> .....	601
<b>21.4 Information über den Quelltext</b> .....	602
<b>21.5 Warnung vor Mehrfachausführung</b> .....	603
<b>21.6 Typvariabilität von Makros</b> .....	604
<b>21.7 Zusammenfassung</b> .....	607

## **22 Schnittstelle zu C** 609

---

<b>22.1 Mit Bibliotheken arbeiten</b> .....	610
<b>22.2 C-Header</b> .....	611

<b>22.3 C-Ressourcen</b> .....	614
<b>22.4 »void«-Pointer</b> .....	615
<b>22.5 Daten lesen</b> .....	616
<b>22.6 Das Hauptprogramm</b> .....	617
<b>22.7 Zusammenfassung</b> .....	618

## **23 Templates** 619

<b>23.1 Funktionstemplates</b> .....	621
23.1.1 Überladung .....	621
23.1.2 Ein Typ als Parameter .....	622
23.1.3 Funktionskörper eines Funktionstemplates .....	623
23.1.4 Werte als Templateparameter .....	625
23.1.5 Viele Funktionen .....	627
23.1.6 Parameter mit Extras .....	628
23.1.7 Methodentemplates sind auch nur Funktionstemplates .....	630
<b>23.2 Funktionstemplates in der Standardbibliothek</b> .....	631
23.2.1 Ranges statt Container als Templateparameter .....	632
23.2.2 Beispiel: Informationen über Zahlen .....	635
<b>23.3 Eine Klasse als Funktion</b> .....	637
23.3.1 Werte für einen »function«-Parameter .....	638
23.3.2 C-Funktionspointer .....	639
23.3.3 Die etwas andere Funktion .....	641
23.3.4 Praktische Funktoren .....	644
23.3.5 Algorithmen mit Funktoren .....	646
23.3.6 Anonyme Funktionen alias Lambda-Ausdrücke .....	647
23.3.7 Templatefunktionen ohne »template«, aber mit »auto« .....	653
<b>23.4 C++ Concepts</b> .....	654
23.4.1 Wie Sie Concepts lesen .....	654
23.4.2 Wie Sie Concepts anwenden .....	658
23.4.3 Wie Sie Concepts schreiben .....	660
23.4.4 Semantische Einschränkungen .....	661
23.4.5 Zusammenfassung .....	661
<b>23.5 Templateklassen</b> .....	662
23.5.1 Klassentemplates implementieren .....	662
23.5.2 Methoden von Klassentemplates implementieren .....	663
23.5.3 Objekte aus Klassentemplates erzeugen .....	666

23.5.4	Klassentemplates mit mehreren formalen Datentypen .....	670
23.5.5	Klassentemplates mit Non-Type-Parameter .....	671
23.5.6	Klassentemplates mit Default .....	673
23.5.7	Klassentemplates spezialisieren .....	674
<b>23.6</b>	<b>Templates mit variabler Argumentanzahl .....</b>	<b>677</b>
23.6.1	»sizeof ...«-Operator .....	680
23.6.2	Parameter-Pack in Tupel konvertieren .....	681
<b>23.7</b>	<b>Eigene Literale .....</b>	<b>681</b>
23.7.1	Was sind Literale? .....	682
23.7.2	Namensregeln .....	683
23.7.3	Phasenweise .....	683
23.7.4	Überladungsvarianten .....	684
23.7.5	Benutzerdefiniertes Literal mittels Template .....	685
23.7.6	Roh oder gekocht .....	689
23.7.7	Automatisch zusammengefügt .....	690
23.7.8	Unicodeliterale .....	690

## TEIL IV Die Standardbibliothek

<b>24</b>	<b>Container .....</b>	<b>695</b>
<b>24.1</b>	<b>Grundlagen .....</b>	<b>696</b>
24.1.1	Wiederkehrend .....	696
24.1.2	Abstrakt .....	697
24.1.3	Operationen .....	699
24.1.4	Komplexität .....	700
24.1.5	Container und ihre Iteratoren .....	702
24.1.6	Ranges vereinfachen Iteratoren .....	704
24.1.7	Ranges, Views, Concepts, Adapter, Generatoren und Algorithmen ...	708
24.1.8	Algorithmen .....	709
<b>24.2</b>	<b>Iteratoren-Grundlagen .....</b>	<b>709</b>
24.2.1	Iteratoren aus Containern .....	711
24.2.2	Mehr Funktionalität mit Iteratoren .....	713
<b>24.3</b>	<b>Allokatoren: Speicherfragen .....</b>	<b>714</b>
<b>24.4</b>	<b>Containergemeinsamkeiten .....</b>	<b>717</b>

<b>24.5 Ein Überblick über die Standardcontainerklassen</b> .....	719
24.5.1 Typaliase der Container .....	720
<b>24.6 Die sequenziellen Containerklassen</b> .....	723
24.6.1 Gemeinsamkeiten und Unterschiede .....	725
24.6.2 Methoden von Sequenzcontainern .....	727
24.6.3 »vector« .....	730
24.6.4 »array« .....	750
24.6.5 »deque« .....	756
24.6.6 »list« .....	760
24.6.7 »forward_list« .....	763
<b>24.7 Assoziativ und geordnet</b> .....	769
24.7.1 Gemeinsamkeiten und Unterschiede .....	770
24.7.2 Methoden der geordneten assoziativen Container .....	771
24.7.3 »set« .....	773
24.7.4 »map« .....	788
24.7.5 »multiset« .....	795
24.7.6 »multimap« .....	801
<b>24.8 Nur assoziativ und nicht garantiert</b> .....	805
24.8.1 Hashtabellen .....	806
24.8.2 Gemeinsamkeiten und Unterschiede .....	810
24.8.3 Methoden der ungeordneten assoziativen Container .....	813
24.8.4 »unordered_set« .....	814
24.8.5 »unordered_map« .....	823
24.8.6 »unordered_multiset« .....	828
24.8.7 »unordered_multimap« .....	834
<b>24.9 Containeradapter</b> .....	837
<b>24.10 Sonderfälle: »string«, »basic_string« und »vector&lt;char&gt;«</b> .....	840
<b>24.11 Sonderfälle: »vector&lt;bool&gt;«, »array&lt;bool,n&gt;« und »bitset&lt;n&gt;«</b> .....	842
24.11.1 Dynamisch und kompakt: »vector<bool>« .....	842
24.11.2 Statisch: »array<bool,n>« und »bitset<n>« .....	842
<b>24.12 Sonderfall: Value-Array mit »valarray&lt;&gt;«</b> .....	845
24.12.1 Eigenschaften der Elemente .....	847
24.12.2 Initialisieren .....	849
24.12.3 Zuweisen .....	849
24.12.4 Einfügen und löschen .....	849
24.12.5 Zugreifen .....	850
24.12.6 Spezialität: Alle Daten manipulieren .....	850
24.12.7 Spezialität: Schneiden und maskieren .....	851



**25 Containerunterstützung** 855

---

- 25.1 Algorithmen ..... 857
- 25.2 Iteratoren und Ranges ..... 858
- 25.3 Iteratoradapter ..... 860
- 25.4 Algorithmen der Standardbibliothek ..... 861
- 25.5 Parallele Ausführung ..... 863
- 25.6 Liste der Algorithmusfunktionen und Range-Adapter ..... 866
  - 25.6.1 Liste der Range-Adapter und Views ..... 868
  - 25.6.2 Ranges als Parameter (und mehr) ..... 874
  - 25.6.3 Liste der (eigentlich) nicht verändernden Algorithmen ..... 876
  - 25.6.4 Inhärent modifizierende Algorithmen ..... 881
  - 25.6.5 Algorithmen rund um Partitionen ..... 886
  - 25.6.6 Algorithmen rund ums Sortieren und schnelle Suchen in  
sortierten Bereichen ..... 887
  - 25.6.7 Mischmengenalgorithmen, repräsentiert durch einen  
sortierten Bereich ..... 888
  - 25.6.8 Heapalgorithmen ..... 890
  - 25.6.9 Minimum und Maximum ..... 891
  - 25.6.10 Diverse Algorithmen ..... 891
- 25.7 Elemente verknüpfende Algorithmen aus »<numeric>« und  
»<ranges>« ..... 892
- 25.8 Kopie statt Zuweisung – Werte in uninitialisierten  
Speicherbereichen ..... 899
- 25.9 Eigene Algorithmen ..... 901
- 25.10 Eigene Views und Range-Adapter schreiben ..... 903

**26 Guter Code, 6. Dan: Für jede Aufgabe der richtige Container** 907

---

- 26.1 Alle Container nach Aspekten sortiert ..... 907
  - 26.1.1 Wann ist ein »vector« nicht die beste Wahl? ..... 907
  - 26.1.2 Immer sortiert: »set«, »map«, »multiset« und »multimap« ..... 908
  - 26.1.3 Im Speicher hintereinander: »vector«, »array« ..... 909

26.1.4	Einfügung billig: »list« .....	910
26.1.5	Wenig Speicheroverhead: »vector«, »array« .....	910
26.1.6	Größe dynamisch: alle außer »array« .....	912
<b>26.2</b>	<b>Rezepte für Container</b> .....	<b>913</b>
26.2.1	Zwei Phasen? »vector« als guter »set«-Ersatz .....	913
26.2.2	Den Inhalt eines Containers auf einem Stream ausgeben .....	915
26.2.3	So statisch ist »array« gar nicht .....	916
<b>26.3</b>	<b>Algorithmen je nach Container unterschiedlich implementieren</b> .....	<b>919</b>
<b>27</b>	<b>Streams, Dateien und Formatierung</b> .....	<b>921</b>
<b>27.1</b>	<b>Ein- und Ausgabekonzept mit Streams</b> .....	<b>922</b>
<b>27.2</b>	<b>Globale, vordefinierte Standardstreams</b> .....	<b>922</b>
27.2.1	Streamoperatoren << und >> .....	923
<b>27.3</b>	<b>Methoden für die Aus- und Eingabe von Streams</b> .....	<b>925</b>
27.3.1	Methoden für die unformatierte Ausgabe .....	925
27.3.2	Methoden für die (unformatierte) Eingabe .....	926
<b>27.4</b>	<b>Fehlerbehandlung und Zustand von Streams</b> .....	<b>929</b>
27.4.1	Methoden für die Behandlung von Fehlern bei Streams .....	930
<b>27.5</b>	<b>Streams manipulieren und formatieren</b> .....	<b>933</b>
27.5.1	Manipulatoren .....	933
27.5.2	Eigene Manipulatoren ohne Argumente erstellen .....	939
27.5.3	Eigene Manipulatoren mit Argumenten erstellen .....	941
27.5.4	Format-Flags direkt ändern .....	942
<b>27.6</b>	<b>Streams für die Dateiein- und Dateiausgabe</b> .....	<b>945</b>
27.6.1	Die Streams »ifstream«, »ofstream« und »fstream« .....	946
27.6.2	Verbindung zu einer Datei herstellen .....	946
27.6.3	Lesen und Schreiben .....	951
27.6.4	Wahlfreier Zugriff .....	958
27.6.5	Synchronisierte Streams für Threads .....	959
<b>27.7</b>	<b>Streams für Strings</b> .....	<b>961</b>
27.7.1	Unterschied zu »to_string« .....	964
27.7.2	»to_chars« und »format« sind flexibler als »to_string« .....	965
27.7.3	Lesen aus einem String .....	965

<b>27.8</b>	<b>Streampuffer</b> .....	966
27.8.1	Zugriff auf den Streampuffer von »iostream«-Objekten .....	967
27.8.2	»filebuf« .....	969
27.8.3	»stringbuf« .....	969
<b>27.9</b>	<b>»filesystem«</b> .....	969
<b>27.10</b>	<b>Formatieren</b> .....	971
27.10.1	Einfache Formatierung .....	972
27.10.2	Eigene Typen formatieren .....	974

## **28 Standardbibliothek – Extras** 979

---

<b>28.1</b>	<b>»pair« und »tuple«</b> .....	979
28.1.1	Mehrere Werte zurückgeben .....	980
<b>28.2</b>	<b>Reguläre Ausdrücke</b> .....	987
28.2.1	Matchen und Suchen .....	988
28.2.2	Ergebnis und Teile davon .....	988
28.2.3	Gefundenes Ersetzen .....	989
28.2.4	Reich an Varianten .....	990
28.2.5	Iteratoren .....	990
28.2.6	Matches .....	991
28.2.7	Optionen .....	991
28.2.8	Geschwindigkeit .....	992
28.2.9	Standardsyntax leicht gekürzt .....	992
28.2.10	Anmerkungen zu regulären Ausdrücken in C++ .....	994
<b>28.3</b>	<b>Zufall</b> .....	997
28.3.1	Einen Würfel werfen .....	998
28.3.2	Echter Zufall .....	1000
28.3.3	Andere Generatoren .....	1000
28.3.4	Verteilungen .....	1002
<b>28.4</b>	<b>Mathematisches</b> .....	1006
28.4.1	Brüche und Zeiten – »<ratio>« und »<chrono>« .....	1006
28.4.2	Vordefinierte Suffixe für benutzerdefinierte Literale .....	1030
<b>28.5</b>	<b>Systemfehlerbehandlung mit »system_error«</b> .....	1033
28.5.1	Überblick .....	1034
28.5.2	Prinzipien .....	1035
28.5.3	»error_code« und »error_condition« .....	1036
28.5.4	Fehlerkategorien .....	1040

28.5.5	Eigene Fehlercodes .....	1040
28.5.6	»system_error«-Exception .....	1041
<b>28.6</b>	<b>Laufzeitinformationen – »&lt;typeinfo&gt;« und »&lt;typeid&gt;« .....</b>	<b>1043</b>
<b>28.7</b>	<b>Hilfsklassen rund um Funktoren – »&lt;functional&gt;« .....</b>	<b>1047</b>
28.7.1	Funktionsobjekte .....	1048
28.7.2	Funktionsgeneratoren .....	1052
<b>28.8</b>	<b>»optional« für einen oder keinen Wert .....</b>	<b>1055</b>
<b>28.9</b>	<b>»variant« für einen von mehreren Typen .....</b>	<b>1056</b>
<b>28.10</b>	<b>»any« hält jeden Typ .....</b>	<b>1058</b>
<b>28.11</b>	<b>Spezielle mathematische Funktionen .....</b>	<b>1059</b>
<b>28.12</b>	<b>Schnelle Umwandlung mit »&lt;charconv&gt;« .....</b>	<b>1060</b>

## **29 Threads – Programmieren mit Mehrläufigkeit** 1063

<b>29.1</b>	<b>C++-Threading-Grundlagen .....</b>	<b>1064</b>
29.1.1	Starten von reinen Threads .....	1065
29.1.2	Einen Thread vorzeitig beenden .....	1067
29.1.3	Warten auf einen Thread .....	1068
29.1.4	Im startenden Thread Exceptions berücksichtigen .....	1069
29.1.5	Einer Threadfunktion Parameter übergeben .....	1072
29.1.6	Einen Thread verschieben .....	1077
29.1.7	Wie viele Threads starten? .....	1079
29.1.8	Welcher Thread bin ich? .....	1082
<b>29.2</b>	<b>Gemeinsame Daten .....</b>	<b>1082</b>
29.2.1	Data Races .....	1083
29.2.2	Riegel .....	1086
29.2.3	Barrieren .....	1086
29.2.4	Mutexe .....	1088
29.2.5	Interface-Design für Multithreading .....	1091
29.2.6	Sperrern können zum Patt führen .....	1096
29.2.7	Flexibleres Sperrern mit »unique_lock« .....	1098
<b>29.3</b>	<b>Andere Möglichkeiten zur Synchronisation .....</b>	<b>1100</b>
29.3.1	Nur einmal aufrufen mit »once_flag« und »call_once« .....	1100
29.3.2	Sperrern zählen mit »recursive_mutex« .....	1102
<b>29.4</b>	<b>Im eigenen Speicher mit »thread_local« .....</b>	<b>1104</b>

<b>29.5</b>	<b>Mit »condition_variable« auf Ereignisse warten</b>	1105
29.5.1	»notify_all«	1108
29.5.2	Ausgabe synchronisieren	1109
<b>29.6</b>	<b>Einmal warten mit »future«</b>	1110
29.6.1	Launch Policies	1112
29.6.2	Gewisse Zeit warten	1113
29.6.3	Ausnahmebehandlung bei »future«	1116
29.6.4	»promise«	1118
<b>29.7</b>	<b>Atomics</b>	1122
29.7.1	Übersicht über die Operationen	1123
29.7.2	Memory Order	1125
29.7.3	Beispiel	1126
<b>29.8</b>	<b>Koroutinen</b>	1127
29.8.1	Koroutinen im Compiler	1128
29.8.2	Generator	1129
29.8.3	Koroutinen mit »promise_type«	1130
<b>29.9</b>	<b>Zusammenfassung</b>	1133
29.9.1	Header »<thread>«	1134
29.9.2	Header »<latch>« und »<barrier>«	1134
29.9.3	Header »<mutex>« und »<shared mutex>«	1134
29.9.4	Header »<condition variable>«	1135
29.9.5	Header »<future>«	1135
29.9.6	Header »<atomic>«	1136
29.9.7	Header »<coroutine>«	1136
 <b>Anhang</b>		1137
<b>A</b>	<b>Guter Code, 7. Dan: Richtlinien</b>	1137
A.1	Guideline Support Library	1138
A.2	C++ Core Guidelines	1139
<b>B</b>	<b>Cheat Sheet</b>	1153
 Index		1157