

# Inhalt

Vorwort .....	23
---------------	----

## TEIL I Grundlagen

<b>1 Das C++-Handbuch</b> .....	<b>29</b>
---------------------------------	-----------

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

<b>2 Programmieren in C++</b> .....	<b>35</b>
-------------------------------------	-----------

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
2.4 Entwicklungsumgebungen .....	39
2.5 Die Kommandozeile unter Ubuntu .....	40
2.5.1 Ein Programm erstellen .....	41
2.5.2 Automatisieren mit Makefile .....	43
2.6 Die IDE »Microsoft Visual Studio Community« unter Windows .....	44
2.7 Das Beispielprogramm beschleunigen .....	46

<b>3 C++ für Umsteiger</b> .....	<b>49</b>
----------------------------------	-----------

<b>4</b>	<b>Die Grundbausteine von C++</b>	<b>55</b>
<b>4.1</b>	<b>Kommentare</b>	<b>58</b>
<b>4.2</b>	<b>Die »include«-Direktive</b>	<b>58</b>
<b>4.3</b>	<b>Die Standardbibliothek</b>	<b>58</b>
<b>4.4</b>	<b>Die Funktion »main()«</b>	<b>59</b>
<b>4.5</b>	<b>Typen</b>	<b>59</b>
<b>4.6</b>	<b>Variablen</b>	<b>60</b>
<b>4.7</b>	<b>Initialisierung</b>	<b>60</b>
<b>4.8</b>	<b>Ausgabe auf der Konsole</b>	<b>61</b>
<b>4.9</b>	<b>Anweisungen</b>	<b>61</b>
<b>4.10</b>	<b>Ohne Eile erklärt</b>	<b>62</b>
4.10.1	Leerräume, Bezeichner und Token	64
4.10.2	Kommentare	65
4.10.3	Funktionen und Argumente	66
4.10.4	Seiteneffekt-Operatoren	67
4.10.5	Die »main«-Funktion	68
4.10.6	Anweisungen	70
4.10.7	Ausdrücke	72
4.10.8	Zuweisungen	74
4.10.9	Typen	75
4.10.10	Variablen – Deklaration, Definition und Initialisierung	80
4.10.11	Initialisieren mit »auto«	81
4.10.12	Details zur »include«-Direktive und »include« direkt	81
4.10.13	Eingabe und Ausgabe	83
4.10.14	Der Namensraum »std«	83
<b>4.11</b>	<b>Operatoren</b>	<b>85</b>
4.11.1	Operatoren und Operanden	86
4.11.2	Überblick über Operatoren	87
4.11.3	Arithmetische Operatoren	87
4.11.4	Bitweise Arithmetik	88
4.11.5	Zusammengesetzte Zuweisung	91
4.11.6	Post- und Präinkrement sowie Post- und Prädekrement	92
4.11.7	Relationale Operatoren	92
4.11.8	Logische Operatoren	93
4.11.9	Pointer- und Dereferenzierungsoperatoren	94
4.11.10	Besondere Operatoren	95
4.11.11	Funktionsähnliche Operatoren	96
4.11.12	Operatorreihenfolge	97

<b>4.12</b>	<b>Eingebaute Datentypen</b>	98
4.12.1	Übersicht	99
4.12.2	Eingebaute Datentypen initialisieren	101
4.12.3	Ganzzahlen	101
4.12.4	Fließkommazahlen	112
4.12.5	Wahrheitswerte	125
4.12.6	Zeichentypen	127
4.12.7	Komplexe Zahlen	129
<b>4.13</b>	<b>Undefiniertes und unspezifiziertes Verhalten</b>	132

---

## 5 Guter Code, 1. Dan: Lesbar programmieren 135

---

<b>5.1</b>	<b>Kommentare</b>	135
<b>5.2</b>	<b>Dokumentation</b>	136
<b>5.3</b>	<b>Einrückungen und Zeilenlänge</b>	137
<b>5.4</b>	<b>Zeilen pro Funktion und Datei</b>	138
<b>5.5</b>	<b>Klammern und Leerzeichen</b>	139
<b>5.6</b>	<b>Namen</b>	140

---

## 6 Höhere Datentypen 143

---

<b>6.1</b>	<b>Strings und Streams</b>	144
<b>6.2</b>	<b>Der Zeichenkettentyp »string«</b>	144
6.2.1	Initialisierung	145
6.2.2	Funktionen und Methoden	146
6.2.3	Andere Stringtypen	147
<b>6.3</b>	<b>Streams</b>	148
6.3.1	Eingabe- und Ausgabeoperatoren	149
6.3.2	»getline«	151
6.3.3	Dateien für die Ein- und Ausgabe	151
6.3.4	Manipulatoren	153
6.3.5	Der Manipulator »endl«	153
<b>6.4</b>	<b>Behälter und Zeiger</b>	155
6.4.1	Container	155
6.4.2	Parametrisierte Typen	155
<b>6.5</b>	<b>Die einfachen Sequenzcontainer</b>	156
6.5.1	»array«	156

6.5.2	»vector«	158
<b>6.6</b>	<b>Algorithmen</b>	<b>161</b>
<b>6.7</b>	<b>Zeiger und C-Arrays</b>	<b>161</b>
6.7.1	Zeigertypen	162
6.7.2	C-Arrays	162

## **7 Funktionen** 163

---

<b>7.1</b>	<b>Deklaration und Definition einer Funktion</b>	<b>164</b>
<b>7.2</b>	<b>Funktionstyp</b>	<b>165</b>
<b>7.3</b>	<b>Funktionen verwenden</b>	<b>165</b>
<b>7.4</b>	<b>Eine Funktion definieren</b>	<b>167</b>
<b>7.5</b>	<b>Mehr zu Parametern</b>	<b>168</b>
7.5.1	Call-by-Value	168
7.5.2	Call-by-Reference	169
7.5.3	Konstante Referenzen	170
7.5.4	Aufruf als Wert, Referenz oder konstante Referenz?	170
<b>7.6</b>	<b>Funktionskörper</b>	<b>172</b>
<b>7.7</b>	<b>Parameter umwandeln</b>	<b>173</b>
<b>7.8</b>	<b>Funktionen überladen</b>	<b>175</b>
<b>7.9</b>	<b>Default-Parameter</b>	<b>177</b>
<b>7.10</b>	<b>Beliebig viele Argumente</b>	<b>179</b>
<b>7.11</b>	<b>Alternative Schreibweise zur Funktionsdeklaration</b>	<b>179</b>
<b>7.12</b>	<b>Spezialitäten</b>	<b>180</b>
7.12.1	»noexcept«	180
7.12.2	Inline-Funktionen	181
7.12.3	»constexpr«	182
7.12.4	Gelöschte Funktionen	182
7.12.5	Spezialitäten bei Klassenmethoden	182

## **8 Anweisungen im Detail** 185

---

<b>8.1</b>	<b>Der Anweisungsblock</b>	<b>188</b>
<b>8.2</b>	<b>Die leere Anweisung</b>	<b>190</b>
<b>8.3</b>	<b>Deklarationsanweisung</b>	<b>191</b>

8.4	Die Ausdrucksanweisung .....	192
8.5	Die »if«-Anweisung .....	193
8.6	Die »while«-Schleife .....	195
8.7	Die »do-while«-Schleife .....	197
8.8	Die »for«-Schleife .....	198
8.9	Die bereichsbasierte »for«-Schleife .....	200
8.10	Die »switch«-Verzweigung .....	202
8.11	Die »break«-Anweisung .....	206
8.12	Die »continue«-Anweisung .....	207
8.13	Die »return«-Anweisung .....	207
8.14	Die »goto«-Anweisung .....	209
8.15	Der »try-catch«-Block und »throw« .....	210
8.16	Zusammenfassung .....	212

## 9 Ausdrücke im Detail 213

---

9.1	Berechnungen und Seiteneffekte .....	214
9.2	Arten von Ausdrücken .....	215
9.3	Literale .....	216
9.4	Bezeichner .....	216
9.5	Klammern .....	217
9.6	Funktionsaufruf und Index-Zugriff .....	218
9.7	Zuweisung .....	218
9.8	Typumwandlung .....	220

## 10 Fehlerbehandlung 221

---

10.1	Fehlerbehandlung mit Fehlercodes .....	223
10.2	Was ist eine Ausnahme? .....	226
10.2.1	Ausnahmen auslösen und behandeln .....	227
10.2.2	Aufrufstapel abwickeln .....	228
10.3	Kleinere Fehlerbehandlungen .....	229
10.4	Weiterwerfen – »rethrow« .....	229

<b>10.5</b>	<b>Die Reihenfolge im »catch«</b>	230
10.5.1	Kein »finally«	231
10.5.2	Exceptions der Standardbibliothek	231
<b>10.6</b>	<b>Typen für Exceptions</b>	232
<b>10.7</b>	<b>Wenn eine Exception aus »main« herausfällt</b>	233

---

## 11 Guter Code, 2. Dan: Modularisierung 235

---

<b>11.1</b>	<b>Programm, Bibliothek, Objektdatei</b>	235
<b>11.2</b>	<b>Bausteine</b>	236
<b>11.3</b>	<b>Trennen der Funktionalitäten</b>	237
<b>11.4</b>	<b>Ein modulares Beispielprojekt</b>	238
11.4.1	Namensräume	241
11.4.2	Implementierung	242
11.4.3	Die Bibliothek nutzen	247
<b>11.5</b>	<b>Spezialthema: Unity-Builds</b>	249

## TEIL II Objektorientierte Programmierung und mehr

---

## 12 Von der Struktur zur Klasse 253

---

<b>12.1</b>	<b>Initialisierung</b>	255
<b>12.2</b>	<b>Rückgabe eigener Typen</b>	256
<b>12.3</b>	<b>Methoden statt Funktionen</b>	257
<b>12.4</b>	<b>Das bessere »drucke«</b>	260
<b>12.5</b>	<b>Eine Ausgabe wie jede andere</b>	262
<b>12.6</b>	<b>Methoden inline definieren</b>	263
<b>12.7</b>	<b>Implementierung und Definition trennen</b>	264
<b>12.8</b>	<b>Initialisierung per Konstruktor</b>	265
12.8.1	Member-Defaultwerte in der Deklaration	267
12.8.2	Konstruktor-Delegation	268
12.8.3	Defaultwerte für die Konstruktor-Parameter	269
12.8.4	»init«-Methode nicht im Konstruktor aufrufen	270
12.8.5	Exceptions im Konstruktor	271

<b>12.9 Struktur oder Klasse?</b>	271
12.9.1 Kapselung	273
12.9.2 »public« und »private«, Struktur und Klasse	273
12.9.3 Daten mit »struct«, Verhalten mit »class«	273
12.9.4 Initialisierung von Typen mit privaten Daten	274
<b>12.10 Zwischenergebnis</b>	275
<b>12.11 Verwendung eigener Datentypen</b>	276
12.11.1 Klassen als Werte verwenden	278
12.11.2 Konstruktoren nutzen	281
12.11.3 Typumwandlungen	282
12.11.4 Kapseln und entkapseln	284
12.11.5 Typen lokal einen Namen geben	288
<b>12.12 Typinferenz mit »auto«</b>	291
<b>12.13 Eigene Klassen in Standardcontainern</b>	294

## **13 Namensräume und Qualifizierer** 297

---

<b>13.1 Der Namensraum »std«</b>	297
<b>13.2 Anonymer Namensraum</b>	301
<b>13.3 »static« macht lokal</b>	303
<b>13.4 »static« teilt gern</b>	303
<b>13.5 »static« macht dauerhaft</b>	306
13.5.1 »inline namespace«	308
<b>13.6 Zusammenfassung</b>	309
<b>13.7 Const</b>	310
13.7.1 Const-Parameter	311
13.7.2 Const-Methoden	312
13.7.3 Const-Variablen	313
13.7.4 Const-Rückgaben	314
13.7.5 Const zusammen mit »static«	318
13.7.6 Noch konstanter mit »constexpr«	319
13.7.7 Un-Const mit »mutable«	322
13.7.8 Const-Korrektheit	323
13.7.9 Zusammenfassung	324
<b>13.8 Flüchtig mit »volatile«</b>	324

<b>14.1</b>	<b>Arten des Tests</b>	327
14.1.1	Refactoring	329
14.1.2	Unittests	329
14.1.3	Sozial oder solitär	331
14.1.4	Doppelgänger	333
14.1.5	Suites	334
<b>14.2</b>	<b>Frameworks</b>	335
14.2.1	Arrange, Act, Assert	337
14.2.2	Frameworks zur Auswahl	338
<b>14.3</b>	<b>Boost.Test</b>	339
<b>14.4</b>	<b>Hilfsmakros für Assertions</b>	343
<b>14.5</b>	<b>Ein Beispielprojekt mit Unittests</b>	346
14.5.1	Privates und Öffentliches testen	347
14.5.2	Ein automatisches Testmodul	348
14.5.3	Test compilieren	351
14.5.4	Die Testsuite selbst zusammenbauen	351
14.5.5	Testen von Privatem	355
14.5.6	Parametrisierte Tests	356

## 15 Vererbung

<b>15.1</b>	<b>Beziehungen</b>	360
15.1.1	Hat-ein-Komposition	360
15.1.2	Hat-ein-Aggregation	360
15.1.3	Ist-ein-Vererbung	361
15.1.4	Ist-Instanz-von versus Ist-ein-Beziehung	362
<b>15.2</b>	<b>Vererbung in C++</b>	362
<b>15.3</b>	<b>Hat-ein versus ist-ein</b>	363
<b>15.4</b>	<b>Gemeinsamkeiten finden</b>	364
<b>15.5</b>	<b>Abgeleitete Typen erweitern</b>	366
<b>15.6</b>	<b>Methoden überschreiben</b>	367
<b>15.7</b>	<b>Wie Methoden funktionieren</b>	368
<b>15.8</b>	<b>Virtuelle Methoden</b>	370
<b>15.9</b>	<b>Konstruktoren in Klassenhierarchien</b>	372



<b>15.10 Typumwandlung in Klassenhierarchien</b>	373
15.10.1 Die Vererbungshierarchie aufwärts umwandeln	373
15.10.2 Die Vererbungshierarchie abwärts umwandeln	374
15.10.3 Referenzen behalten auch die Typinformation	374
<b>15.11 Wann virtuell?</b>	375
<b>15.12 Andere Designs zur Erweiterbarkeit</b>	376

## 16 Der Lebenszyklus von Klassen 379

---

<b>16.1 Erzeugung und Zerstörung</b>	380
<b>16.2 Temporary: kurzlebige Werte</b>	382
<b>16.3 Der Destruktor zum Konstruktor</b>	383
16.3.1 Kein Destruktor nötig	385
16.3.2 Ressourcen im Destruktor	385
<b>16.4 Yoda-Bedingung</b>	387
<b>16.5 Konstruktion, Destruktion und Exceptions</b>	389
<b>16.6 Kopieren</b>	390
<b>16.7 Zuweisungsoperator</b>	393
<b>16.8 Streichen von Methoden</b>	396
<b>16.9 Verschiebeoperationen</b>	398
16.9.1 Was der Compiler generiert	402
<b>16.10 Operatoren</b>	403
<b>16.11 Eigene Operatoren in einem Datentyp</b>	406
<b>16.12 Besondere Klassenformen</b>	411
16.12.1 Abstrakte Klassen und Methoden	411
16.12.2 Aufzählungsklassen	412

## 17 Guter Code, 4. Dan: Sicherheit, Qualität und Nachhaltigkeit 415

---

<b>17.1 Die Nuller-Regel</b>	415
17.1.1 Die großen Fünf	415
17.1.2 Hilfskonstrukt per Verbot	416
17.1.3 Die Nuller-Regel und ihr Einsatz	417
17.1.4 Ausnahmen von der Nuller-Regel	418

<b>17.2</b>	<b>RAII – Resource Acquisition Is Initialization</b>	420
17.2.1	Ein Beispiel mit C	420
17.2.2	Besitzende Raw-Pointer	422
17.2.3	Von C nach C++	423
17.2.4	Es muss nicht immer eine Exception sein	425
17.2.5	Mehrere Konstruktoren	426
17.2.6	Mehrphasige Initialisierung	426
17.2.7	Definieren, wo es gebraucht wird	427
17.2.8	Nothrow-new	427

---

## 18 Spezielles für Klassen 429

---

<b>18.1</b>	<b>Darf alles sehen – »friend«-Klassen</b>	429
<b>18.2</b>	<b>non-public-Vererbung</b>	434
18.2.1	Auswirkungen auf die Außenwelt	435
18.2.2	Nicht-öffentliche Vererbung in der Praxis	437
<b>18.3</b>	<b>Signatur-Klassen als Interfaces</b>	439
<b>18.4</b>	<b>Multiple Vererbung</b>	443
18.4.1	Multiple Vererbung in der Praxis	444
18.4.2	Achtung bei Typumwandlungen von Zeigern	448
18.4.3	Das Beobachter-Muster als praktisches Beispiel	450
<b>18.5</b>	<b>Rautenförmige multiple Vererbung – »virtual« für Klassenhierarchien</b>	452
<b>18.6</b>	<b>Literale Datentypen – »constexpr« für Konstruktoren</b>	456

---

## 19 Guter Code, 5. Dan: Klassisches objektorientiertes Design 459

---

<b>19.1</b>	<b>Objekte in C++</b>	461
<b>19.2</b>	<b>Objektorientiert designen</b>	462
19.2.1	SOLID	462
19.2.2	Seien Sie nicht STUPID	480

## TEIL III Fortgeschrittene Themen

<b>20</b>	<b>Zeiger</b>	483
<hr/>		
20.1	Adressen .....	484
20.2	Zeiger .....	485
20.3	Gefahren von Aliasing .....	487
20.4	Heapspeicher und Stapelspeicher .....	489
20.4.1	Der Stapel .....	489
20.4.2	Der Heap .....	490
20.5	Smarte Pointer .....	492
20.5.1	»unique_ptr« .....	494
20.5.2	»shared_ptr« .....	498
20.6	Rohe Zeiger .....	501
20.7	C-Arrays .....	506
20.7.1	Rechnen mit Zeigern .....	506
20.7.2	Verfall von C-Arrays .....	507
20.7.3	Dynamische C-Arrays .....	509
20.7.4	Zeichenkettenliterale .....	510
20.8	Iteratoren .....	511
20.9	Zeiger als Iteratoren .....	513
20.10	Zeiger im Container .....	513
20.11	Die Ausnahme: wann das Wegräumen nicht nötig ist .....	514
<b>21</b>	<b>Makros</b>	517
<hr/>		
21.1	Der Präprozessor .....	518
21.2	Vorsicht vor fehlenden Klammern .....	521
21.3	Vorsicht vor Mehrfachausführung .....	522
21.4	Typvariabilität von Makros .....	523
21.5	Zusammenfassung .....	526

<b>22</b>	<b>Schnittstelle zu C</b>	<b>527</b>
<hr/>		
22.1	Mit Bibliotheken arbeiten .....	528
22.2	C-Header .....	529
22.3	C-Ressourcen .....	532
22.4	»void«-Pointer .....	532
22.5	Daten lesen .....	533
22.6	Das Hauptprogramm .....	534
22.7	Zusammenfassung .....	535
<b>23</b>	<b>Templates</b>	<b>537</b>
<hr/>		
23.1	Funktionstemplates .....	538
23.1.1	Überladung .....	539
23.1.2	Ein Typ als Parameter .....	540
23.1.3	Funktionskörper eines Funktionstemplates .....	540
23.1.4	Zahlen als Templateparameter .....	542
23.1.5	Viele Funktionen .....	543
23.1.6	Parameter mit Extras .....	544
23.1.7	Methodentemplates sind auch nur Funktionstemplates .....	546
23.2	Funktionstemplates in der Standardbibliothek .....	547
23.2.1	Iteratoren statt Container als Templateparameter .....	548
23.2.2	Beispiel: Informationen über Zahlen .....	550
23.3	Eine Klasse als Funktion .....	551
23.3.1	Werte für einen »function«-Parameter .....	552
23.3.2	C-Funktionspointer .....	553
23.3.3	Die etwas andere Funktion .....	555
23.3.4	Praktische Funktoren .....	558
23.3.5	Algorithmen mit Funktoren .....	559
23.3.6	Anonyme Funktionen alias Lambda-Ausdrücke .....	560
23.3.7	Templatefunktionen ohne »template«, aber mit »auto« .....	564
23.4	Templateklassen .....	565
23.4.1	Klassentemplates implementieren .....	565
23.4.2	Methoden von Klassentemplates implementieren .....	566
23.4.3	Objekte aus Klassentemplates erzeugen .....	568
23.4.4	Klassentemplates mit mehreren formalen Datentypen .....	570
23.4.5	Klassentemplates mit Non-Type-Parameter .....	571
23.4.6	Klassentemplates mit Default .....	573
23.4.7	Klassentemplates spezialisieren .....	574

<b>23.5</b>	<b>Templates mit variabler Argumentanzahl</b>	576
<b>23.6</b>	<b>Eigene Literale</b>	580
23.6.1	Was sind Literale?	581
23.6.2	Namensregeln	582
23.6.3	Phasenweise	582
23.6.4	Überladungsvarianten	583
23.6.5	Benutzerdefiniertes Literal mittels Template	584
23.6.6	Roh oder gekocht	587
23.6.7	Automatisch zusammengefügt	588
23.6.8	Unicodeliterale	588

## TEIL IV Die Standardbibliothek

### 24 Container 593

---

<b>24.1</b>	<b>Grundlagen</b>	594
24.1.1	Wiederkehrend	594
24.1.2	Abstrakt	595
24.1.3	Operationen	596
24.1.4	Komplexität	597
24.1.5	Container und ihre Iteratoren	598
24.1.6	Algorithmen	600
<b>24.2</b>	<b>Iteratoren-Grundlagen</b>	601
24.2.1	Iteratoren aus Containern	602
24.2.2	Mehr Funktionalität mit Iteratoren	603
<b>24.3</b>	<b>Allokatoren: Speicherfragen</b>	605
<b>24.4</b>	<b>Container-Gemeinsamkeiten</b>	607
<b>24.5</b>	<b>Ein Überblick über die Standardcontainer-Klassen</b>	608
24.5.1	Typalias der Container	609
<b>24.6</b>	<b>Die sequenziellen Containerklassen</b>	612
24.6.1	Gemeinsamkeiten und Unterschiede	614
24.6.2	Methoden von Sequenzcontainern	615
24.6.3	»vector«	618
24.6.4	»array«	631
24.6.5	»deque«	636
24.6.6	»list«	639
24.6.7	»forward_list«	642
<b>24.7</b>	<b>Assoziativ und geordnet</b>	647
24.7.1	Gemeinsamkeiten und Unterschiede	648

24.7.2	Methoden der geordneten assoziativen Container .....	649
24.7.3	»set« .....	651
24.7.4	»map« .....	663
24.7.5	»multiset« .....	670
24.7.6	»multimap« .....	674
<b>24.8</b>	<b>Nur assoziativ und nicht garantiert .....</b>	<b>678</b>
24.8.1	Gemeinsamkeiten und Unterschiede .....	682
24.8.2	Methoden der ungeordneten assoziativen Container .....	684
24.8.3	»unordered_set« .....	685
24.8.4	»unordered_map« .....	693
24.8.5	»unordered_multiset« .....	698
24.8.6	»unordered_multimap« .....	703
<b>24.9</b>	<b>Container-Adapter .....</b>	<b>706</b>
<b>24.10</b>	<b>Sonderfälle: »string«, »basic_string« und »vector&lt;char&gt;« .....</b>	<b>707</b>
<b>24.11</b>	<b>Sonderfälle: »vector&lt;bool&gt;«, »array&lt;bool,n&gt;« und »bitset&lt;n&gt;« .....</b>	<b>708</b>
24.11.1	Dynamisch und kompakt: »vector<bool>« .....	708
24.11.2	Statisch: »array<bool,n>« und »bitset<n>« .....	708
<b>24.12</b>	<b>Sonderfall: Value-Array mit »valarray&lt;&gt;« .....</b>	<b>711</b>

## **25 Container-Unterstützung** 721

---

<b>25.1</b>	<b>Algorithmen .....</b>	<b>721</b>
<b>25.2</b>	<b>Iteratoren .....</b>	<b>723</b>
<b>25.3</b>	<b>Iterator-Adapter .....</b>	<b>723</b>
<b>25.4</b>	<b>Algorithmen der Standardbibliothek .....</b>	<b>724</b>
<b>25.5</b>	<b>Liste der Algorithmusfunktionen .....</b>	<b>726</b>
<b>25.6</b>	<b>Kopie statt Zuweisung – Werte in uninitialisierten Speicherbereichen .....</b>	<b>740</b>
<b>25.7</b>	<b>Eigene Algorithmen .....</b>	<b>741</b>

## **26 Guter Code, 6. Dan: Für jede Aufgabe der richtige Container** 745

---

<b>26.1</b>	<b>Alle Container nach Aspekten sortiert .....</b>	<b>745</b>
26.1.1	Wann ist ein »vector« nicht die beste Wahl? .....	745
26.1.2	Immer sortiert: »set«, »map«, »multiset« und »multimap« .....	746
26.1.3	Im Speicher hintereinander: »vector«, »array« .....	746

26.1.4	Einfügung billig: »list« .....	747
26.1.5	Wenig Speicheroverhead: »vector«, »array« .....	748
26.1.6	Größe dynamisch: alle außer »array« .....	749
<b>26.2</b>	<b>Rezepte für Container</b> .....	750
26.2.1	Zwei Phasen? »vector« als guter »set«-Ersatz .....	751
26.2.2	Den Inhalt eines Containers auf einem Stream ausgeben .....	752
26.2.3	So statisch ist »array« gar nicht .....	753
<b>26.3</b>	<b>Iteratoren sind mehr als nur Zeiger</b> .....	756
<b>26.4</b>	<b>Algorithmen je nach Container unterschiedlich implementieren</b> .....	758

---

## 27 Streams 761

---

<b>27.1</b>	<b>Ein- und Ausgabekonzept</b> .....	761
<b>27.2</b>	<b>Globale, vordefinierte Standardstreams</b> .....	762
<b>27.3</b>	<b>Methoden für die Aus- und Eingabe von Streams</b> .....	764
27.3.1	Methoden für die unformatierte Ausgabe .....	764
27.3.2	Methoden für die (unformatierte) Eingabe .....	765
<b>27.4</b>	<b>Fehlerbehandlung und Zustand von Streams</b> .....	768
27.4.1	Methoden für die Behandlung von Fehlern bei Streams .....	769
<b>27.5</b>	<b>Streams manipulieren und formatieren</b> .....	771
27.5.1	Manipulatoren .....	772
27.5.2	Eigene Manipulatoren ohne Argumente erstellen .....	777
27.5.3	Eigene Manipulatoren mit Argumenten erstellen .....	779
27.5.4	Format-Flags direkt ändern .....	780
<b>27.6</b>	<b>Streams für die Dateiein- und Dateiausgabe</b> .....	782
27.6.1	Die Streams »ifstream«, »ofstream« und »fstream« .....	783
27.6.2	Verbindung zu einer Datei herstellen .....	783
27.6.3	Lesen und Schreiben .....	788
27.6.4	Wahlfreier Zugriff .....	795
<b>27.7</b>	<b>Streams für Strings</b> .....	796
<b>27.8</b>	<b>Streampuffer</b> .....	801

---

## 28 Standardbibliothek – Extras 805

---

<b>28.1</b>	<b>»pair« und »tuple«</b> .....	805
28.1.1	Mehrere Werte zurückgeben .....	806

<b>28.2</b>	<b>Reguläre Ausdrücke</b>	812
28.2.1	Matchen und Suchen	813
28.2.2	Ergebnis und Teile davon	813
28.2.3	Gefundenes Ersetzen	814
28.2.4	Reich an Varianten	814
28.2.5	Iteratoren	815
28.2.6	Matches	816
28.2.7	Optionen	816
28.2.8	Geschwindigkeit	816
28.2.9	Standardsyntax leicht gekürzt	817
28.2.10	Anmerkungen zu regulären Ausdrücken in C++	818
<b>28.3</b>	<b>Zufall</b>	821
28.3.1	Einen Würfel werfen	822
28.3.2	Echter Zufall	824
28.3.3	Andere Generatoren	824
28.3.4	Verteilungen	826
<b>28.4</b>	<b>Mathematisches</b>	830
28.4.1	Brüche und Zeiten – »<ratio>« und »<chrono>«	830
28.4.2	Numerik mit »<numeric>«	843
28.4.3	Vordefinierte Suffixe für benutzerdefinierte Literale	847
<b>28.5</b>	<b>Systemfehlerbehandlung mit »system_error«</b>	849
28.5.1	»error_code« und »error_condition«	851
28.5.2	Fehlerkategorien	855
28.5.3	Eigene Fehlercodes	855
28.5.4	»system_error«-Exception	857
<b>28.6</b>	<b>Laufzeit-Typinformationen – »&lt;typeinfo&gt;« und »&lt;typeid&gt;«</b>	858
<b>28.7</b>	<b>Hilfsklassen rund um Funktoren – »&lt;functional&gt;«</b>	862
28.7.1	Funktionsobjekte	862
28.7.2	Funktionsgeneratoren	866
<b>28.8</b>	<b>Ausblick auf C++17</b>	869
28.8.1	»variant«	869
28.8.2	»optional« und »any«	870
28.8.3	»string_view«	871
28.8.4	»filesystem«	872
28.8.5	Spezielle mathematische Funktionen	872
28.8.6	»sample«	873
28.8.7	»search«	874
28.8.8	»byte«	874



## 29 Threads – Programmieren mit Mehrläufigkeit 875

---

<b>29.1 C++-Threading-Grundlagen</b>	876
29.1.1 Einer Threadfunktion Parameter übergeben	881
29.1.2 Einen Thread verschieben	886
29.1.3 Wie viele Threads starten?	888
29.1.4 Welcher Thread bin ich?	891
<b>29.2 Gemeinsame Daten</b>	891
29.2.1 Daten mit Mutexen schützen	892
29.2.2 Data Races	895
29.2.3 Interface-Design für Multithreading	896
29.2.4 Sperren können zum Patt führen	901
29.2.5 Flexibleres Sperren mit »unique_lock«	903
<b>29.3 Andere Möglichkeiten zur Synchronisation</b>	904
29.3.1 Nur einmal aufrufen mit »once_flag« und »call_once«	904
29.3.2 Sperren zählen mit »recursive_mutex«	907
<b>29.4 Im eigenen Speicher mit »thread_local«</b>	908
<b>29.5 Mit »condition_variable« auf Ereignisse warten</b>	909
<b>29.6 Einmal warten mit »future«</b>	914
29.6.1 Ausnahmebehandlung bei »future«	919
29.6.2 »promise«	921
<b>29.7 Atomics</b>	924
<b>29.8 Zusammenfassung</b>	930
<b>29.9 Ausblick auf C++17</b>	932

## TEIL V Über den Standard hinaus

## 30 Guter Code, 7. Dan: Richtlinien 935

---

<b>30.1 Guideline Support Library</b>	936
<b>30.2 C++ Core Guidelines</b>	937
30.2.1 Motivation	937
30.2.2 Typsicherheit	938
30.2.3 Nutzen Sie RAII	940
30.2.4 Klassenhierarchien	942
30.2.5 Generische Programmierung	945
30.2.6 Lassen Sie sich nicht von Anachronismen verwirren	948

<b>31.1 Ein erstes Miniprogramm</b>	955
31.1.1 Kurze Übersicht über die Oberfläche von Qt Creator	956
31.1.2 Ein einfaches Projekt erstellen	957
<b>31.2 Objektbäume und Besitz</b>	966
<b>31.3 Signale und Slots</b>	967
31.3.1 Verbindung zwischen Signal und Slot herstellen	967
31.3.2 Signal und Slot mithilfe der Qt-Referenz ermitteln	970
<b>31.4 Klassenhierarchie von Qt</b>	987
31.4.1 Basisklasse »QObject«	987
31.4.2 Weitere wichtige Klassen	987
<b>31.5 Eigene Widgets mit dem Qt Designer erstellen</b>	990
<b>31.6 Widgets anordnen</b>	996
31.6.1 Grundlegende Widgets für das Layout	996
<b>31.7 Dialoge erstellen mit »QDialog«</b>	1000
<b>31.8 Vorgefertigte Dialoge von Qt</b>	1008
31.8.1 »QMessageBox« – der klassische Nachrichtendialog	1008
31.8.2 »QFileDialog« – der Dateiauswahldialog	1009
31.8.3 »QInputDialog« – Dialog zur Eingabe von Daten	1011
31.8.4 Weitere Dialoge	1015
<b>31.9 Eigenen Dialog mit dem Qt Designer erstellen</b>	1015
<b>31.10 Grafische Bedienelemente von Qt (Qt-Widgets)</b>	1031
31.10.1 Schaltflächen (Basisklasse »QAbstractButton«)	1031
31.10.2 Container-Widgets (Behälter-Widgets)	1033
31.10.3 Widgets zur Zustandsanzeige	1034
31.10.4 Widgets zur Eingabe	1035
31.10.5 Onlinehilfen	1036
<b>31.11 Anwendungen in einem Hauptfenster</b>	1037
31.11.1 Die Klasse für das Hauptfenster »QMainWindow«	1037
<b>31.12 Zusammenfassung</b>	1048
<b>Cheat Sheet</b>	1052
<b>Index</b>	1055