

# Inhalt

Geleitwort .....	19
Vorwort .....	21

## 1 Wir richten uns ein und bringen es ans Laufen 25

---

1.1 Von der Quelldatei zum ausführbaren Programm .....	25
1.2 Übersicht über die Entwicklungsumgebungen .....	26
1.2.1 Auf allen Systemen vorhandene Entwicklungsumgebungen .....	27
1.2.2 Entwicklungsumgebungen für Microsoft Windows .....	28
1.2.3 Entwicklungsumgebungen für Linux und Unix-like .....	30
1.2.4 Entwicklungsumgebungen für Mac OS .....	31
1.2.5 Reine Compiler ohne Entwicklungsumgebung .....	31
1.3 Quellcode übersetzen .....	32
1.3.1 Übersetzen mit Entwicklungsumgebung .....	32
1.3.2 Übersetzen mit der Kommandozeile .....	35
1.4 C++-Referenzen zum Nachschlagen (bzw. Laden) .....	37
1.5 Das Hauptprogramm – die main()-Funktion .....	38
1.5.1 Das Programm bei der Ausführung .....	40
1.5.2 Kommandozeilenargumente an main() übergeben .....	41
1.5.3 Quellcode kommentieren .....	43
1.5.4 Programmierstil .....	43
1.6 Die einfachen Streams für die Ein-/Ausgabe .....	43
1.6.1 Ausgabe auf dem Bildschirm .....	45
1.6.2 Einlesen von der Tastatur .....	48
1.6.3 Die Ein- und Ausgabeoperatoren >> und << .....	48
1.7 Zusammenfassung .....	49

## 2 Die Basisdatentypen in C++ 51

---

2.1 Grundlegendes zu Datentypen .....	51
2.1.1 Erlaubte Bezeichner für den Zugriff auf Variablen .....	51
2.1.2 Deklaration und Definition .....	53

2.1.3	Initialisierung von Variablen .....	54
2.1.4	Vereinheitlichte Initialisierung mit C++11 .....	55
<b>2.2</b>	<b>Ganzzahldatentypen (Integer-Datentypen) .....</b>	<b>55</b>
2.2.1	Regeln für gültige Ganzzahlen .....	56
2.2.2	Ganzzahlen mit Werten initialisieren .....	57
2.2.3	Positive oder negative Ganzzahlen .....	58
2.2.4	Boolescher Datentyp für die Wahrheit .....	59
<b>2.3</b>	<b>Typen für Gleitkommazahlen .....</b>	<b>60</b>
<b>2.4</b>	<b>Typ(en) für Zeichen .....</b>	<b>62</b>
2.4.1	Der Datentyp char .....	62
2.4.2	Unicode-Unterstützung .....	65
<b>2.5</b>	<b>Typ auto (C++11) .....</b>	<b>68</b>
<b>2.6</b>	<b>Übersicht und Größe der Basisdatentypen .....</b>	<b>70</b>
2.6.1	Ganzzahltypen .....	70
2.6.2	Gleitkommazahlen .....	72
2.6.3	Byte-Größe ermitteln – sizeof-Operator .....	72
2.6.4	Sicherheit beim Kompilieren mit static_assert (C++11) .....	72
2.6.5	<limits> und std::numeric_limits .....	73
2.6.6	<climits> und <float> .....	75
2.6.7	Übersicht zu den fundamentalen Datentypen .....	76
<b>2.7</b>	<b>Rechnen mit C++ .....</b>	<b>77</b>
2.7.1	Arithmetische Operatoren .....	78
2.7.2	Unäre Gegenstücke .....	80
2.7.3	Wenn Wertebereiche überschritten werden .....	81
2.7.4	Rundungsfehler bei Gleitkommazahlen .....	85
2.7.5	Komplexe Zahlen – <complex> .....	87
2.7.6	Nützliche mathematische Funktionen – <cmath> .....	89
<b>2.8</b>	<b>Zufallszahlen (neu in C++11) .....</b>	<b>90</b>
<b>2.9</b>	<b>Konvertieren von Typen .....</b>	<b>92</b>
2.9.1	Automatische (implizite) Typumwandlung .....	92
2.9.2	Automatische Typumwandlung beschränken (C++11) .....	95
2.9.3	Explizite Typumwandlung .....	96
<b>3</b>	<b>Kontrollstrukturen .....</b>	<b>99</b>
<b>3.1</b>	<b>Bedingte Anweisung .....</b>	<b>99</b>
3.1.1	Vergleichsoperatoren .....	102
3.1.2	Logische Operatoren .....	105

3.1.3	Verzweigung (if-else-Anweisung) .....	108
3.1.4	Bedingter Ausdruck (?:) .....	110
<b>3.2</b>	<b>Fallunterscheidung (mehrfache Verzweigung) .....</b>	<b>111</b>
3.2.1	Fallunterscheidung mit switch .....	111
3.2.2	Mehrfache Verzweigung mit else-if-Anweisung(en) .....	113
<b>3.3</b>	<b>Schleifen (Wiederholungen) .....</b>	<b>115</b>
3.3.1	Zähloperatoren (Inkrement und Dekrement) .....	116
3.3.2	Kopfgesteuerte Schleife – while() .....	117
3.3.3	Fußgesteuerte Schleife – do while() .....	119
3.3.4	Zählschleife – for() .....	121
3.3.5	Mengenschleife – Range-based for (C++11) .....	123
3.3.6	Endlosschleife .....	124
3.3.7	Kontrollierte Sprunganweisungen .....	125

## **4      Jenseits der Basisdatentypen** 129

---

<b>4.1</b>	<b>Arrays .....</b>	<b>129</b>
4.1.1	Standardcontainer std::vector .....	130
4.1.2	Rohe Arrays (C-Style-Array) .....	135
4.1.3	Standardcontainer std::array (C++11) .....	139
4.1.4	Assoziatives Array .....	142
<b>4.2</b>	<b>Strings (Zeichenketten) .....</b>	<b>143</b>
4.2.1	Standardcontainer std::string .....	143
4.2.2	Rohe Strings (C-Style-Strings) .....	145
4.2.3	Unterstützung von Unicode (C++11) .....	150
4.2.4	Rohstringliterale (Raw-String) (C++11) .....	152
<b>4.3</b>	<b>Zeiger .....</b>	<b>153</b>
<b>4.4</b>	<b>Referenzen .....</b>	<b>164</b>
<b>4.5</b>	<b>Strukturen .....</b>	<b>166</b>
<b>4.6</b>	<b>Unions .....</b>	<b>177</b>
<b>4.7</b>	<b>Aufzählungstypen (C++11-Version) .....</b>	<b>180</b>
4.7.1	Zugriff auf die Bezeichner .....	181
4.7.2	Typ für enum festlegen .....	182
<b>4.8</b>	<b>Synonym-Technik .....</b>	<b>183</b>
4.8.1	typedef .....	183
4.8.2	Alias-Templates (C++11) .....	185
<b>4.9</b>	<b>Fazit .....</b>	<b>185</b>

<b>5.1</b>	<b>Funktionen definieren</b>	<b>187</b>
<b>5.2</b>	<b>Funktionen aufrufen</b>	<b>189</b>
<b>5.3</b>	<b>Funktionen deklarieren (Vorausdeklaration)</b>	<b>189</b>
<b>5.4</b>	<b>Exkurs: Gültigkeitsbereiche</b>	<b>191</b>
5.4.1	Globaler Gültigkeitsbereich	192
5.4.2	Lokaler Gültigkeitsbereich	193
<b>5.5</b>	<b>Funktionsparameter</b>	<b>195</b>
5.5.1	Funktionsparameter als Kopie (Call by Value)	196
5.5.2	Funktionsparameter als Referenz (Call by Reference)	198
5.5.3	Funktionsparameter als rohe Zeiger	200
5.5.4	Strukturen und Klassen als Funktionsparameter	202
5.5.5	Schreibschutz für die Referenzparameter mit const	204
5.5.6	Default-Parameter (Standardparameter)	204
<b>5.6</b>	<b>Rückgabewert aus einer Funktion</b>	<b>207</b>
5.6.1	Referenz als Rückgabewert	210
5.6.2	Rohe Zeiger als Rückgabewert	211
5.6.3	Dinge, die man besser nicht zurückgibt	211
5.6.4	Mehrere Werte zurückgeben – std::pair<>	213
<b>5.7</b>	<b>Funktionen überladen</b>	<b>215</b>
<b>5.8</b>	<b>Spezielle Funktionen</b>	<b>217</b>
5.8.1	Inline-Funktionen	217
5.8.2	Lambda-Funktionen (C++11)	218
5.8.3	main()-Funktion	219
5.8.4	Funktions- und Programmende	221
<b>5.9</b>	<b>Die neue Funktionssyntax (C++11)</b>	<b>221</b>
5.9.1	Die neue Rückgabesyntax	222
5.9.2	decltype	223
<b>5.10</b>	<b>Ausblick</b>	<b>226</b>

<b>6.1</b>	<b>Namensräume</b>	<b>227</b>
6.1.1	Einen neuen Namensraum erstellen	229
6.1.2	Namensraum verwenden	232

6.1.3	Aliases für Namensräume .....	236
6.1.4	Der Namensraum std .....	236
<b>6.2</b>	<b>Speicherklassenattribute .....</b>	<b>238</b>
6.2.1	Das Schlüsselwort extern .....	238
6.2.2	Das Schlüsselwort static .....	240
6.2.3	Aus Alt mach Neu und »deprecated« .....	243
<b>6.3</b>	<b>Typqualifikatoren .....</b>	<b>244</b>
6.3.1	Das Schlüsselwort const .....	244
6.3.2	Das Schlüsselwort volatile .....	246
<b>6.4</b>	<b>Spezielle Schlüsselwörter .....</b>	<b>247</b>
6.4.1	Das Schlüsselwort inline für Funktionen .....	247
6.4.2	Schlüsselwörter für Klassen (teilweise C++11) .....	247
6.4.3	Neue Schlüsselwörter mit C++11 .....	248
<b>6.5</b>	<b>Präprozessor-Direktiven .....</b>	<b>251</b>
6.5.1	#include .....	252
6.5.2	#define und #undef .....	253
6.5.3	Bedingte Kompilierung .....	256
6.5.4	Weitere Direktiven .....	259
<b>6.6</b>	<b>Modulare Programmierung – Code organisieren .....</b>	<b>260</b>
6.6.1	Module .....	261
6.6.2	Sinnvolle Quellcodeaufteilung .....	261
6.6.3	Die öffentliche Schnittstelle (Headerdatei) .....	264
6.6.4	Die privaten Dateien .....	265
6.6.5	Die Client-Datei .....	266
<b>7</b>	<b>Grundlagen zu den Klassen .....</b>	<b>269</b>
<b>7.1</b>	<b>Prinzip von Klassen .....</b>	<b>271</b>
<b>7.2</b>	<b>Klassen erstellen .....</b>	<b>272</b>
<b>7.3</b>	<b>Objekte einer Klasse erzeugen .....</b>	<b>274</b>
<b>7.4</b>	<b>Klassen(-Eigenschaften) initialisieren .....</b>	<b>276</b>
7.4.1	Klassenelemente direkt initialisieren (C++11) .....	277
7.4.2	Konstruktoren .....	277
7.4.3	Initialisieren mit Methoden .....	293
<b>7.5</b>	<b>Objekte zerstören – Destruktoren .....</b>	<b>295</b>
7.5.1	Destruktor deklarieren .....	295

7.5.2	Destruktor definieren .....	296
7.5.3	Aufruf des Destruktors .....	296
<b>7.6</b>	<b>Exkurs: Zugriffskontrolle auf die Klassenmitglieder .....</b>	<b>298</b>
<b>7.7</b>	<b>Methoden – die Funktionen der Klasse .....</b>	<b>303</b>
7.7.1	Methoden deklarieren und definieren .....	303
7.7.2	Zugriffsmethoden – Setter und Getter .....	306
7.7.3	Zugriff auf die öffentlichen Mitglieder einer Klasse .....	311
7.7.4	const-Methoden (read-only) .....	314
7.7.5	Objekte als Methodenparameter .....	316
7.7.6	This-Zeiger .....	318
7.7.7	Objekte als Rückgabewert .....	321
7.7.8	Globale Hilfsfunktionen .....	322
7.7.9	Globale friend-Funktion .....	324
7.7.10	Methoden überladen .....	326
7.7.11	Die neue Funktionssyntax – decltype und auto (C++11) .....	327
<b>7.8</b>	<b>Spezielle Eigenschaften einer Klasse .....</b>	<b>329</b>
7.8.1	Konstante Elemente in einer Klasse .....	329
7.8.2	Roher Zeiger als Element in einer Klasse .....	332
7.8.3	Statische Eigenschaften in einer Klasse .....	338
7.8.4	Statische Methoden .....	341
7.8.5	Andere Klassen als Eigenschaft in einer Klasse .....	342
7.8.6	constexpr bei Klassen (Objekte zur Kompilierzeit) (C++11) .....	343
<b>7.9</b>	<b>Erzeugen von Methoden steuern (C++11) .....</b>	<b>345</b>
7.9.1	default .....	346
7.9.2	delete .....	348
<b>7.10</b>	<b>Klassen für das Verschieben schreiben (C++11) .....</b>	<b>350</b>
<b>7.11</b>	<b>Friend-Klassen .....</b>	<b>357</b>
<b>8</b>	<b>Operatoren überladen .....</b>	<b>359</b>
<b>8.1</b>	<b>Grundlegendes zum Überladen von Operatoren .....</b>	<b>361</b>
8.1.1	Binäre Operatoren .....	361
8.1.2	Binäre Operatoren als (friend-)Funktion überladen .....	364
8.1.3	Unäre Operatoren .....	366
8.1.4	Unäre Operatoren als (friend-)Funktion überladen .....	369
8.1.5	Unterschied zwischen Operatorüberladung und Methoden .....	371
8.1.6	Regeln für die Operatorüberladung .....	371
8.1.7	Überladbare Operatoren .....	372

<b>8.2</b>	<b>Zuweisungsoperator – operator=</b>	373
8.2.1	Zuweisungsoperator mehrfach überladen	376
8.2.2	Zuweisung verbieten (C++11)	378
<b>8.3</b>	<b>Die Operatoren im Schnelldurchlauf</b>	379
8.3.1	Der Zuweisungsoperator operator=	379
8.3.2	Binäre Operatoren – operator+, -, *, /, %	380
8.3.3	Erweiterte Schreibweise binärer Operatoren (+=, -=, *=, /=, %=)	380
8.3.4	Unäre Operatoren (+, -)	382
8.3.5	Bitweise Operatoren (& (binär),  , ^)	383
8.3.6	Erweiterte Schreibweise bitweiser Operatoren (&=,  =, ^=)	383
8.3.7	Logische Operatoren (==, !=)	383
8.3.8	Die logischen Verknüpfungen (&&,    und !)	386
8.3.9	Vergleichsoperatoren (<, <=, >=, >)	386
8.3.10	Inkrement- und Dekrement-Operator (++ , --)	389
8.3.11	Die Operatoren operator*, operator-> und operator ->*	389
8.3.12	Der Adressoperator &	389
8.3.13	Der Komplementoperator ~	389
8.3.14	Ein- und Ausgabeoperatoren – operator>> und operator<<	390
8.3.15	Der Konvertierungsoperator ()	392
8.3.16	Funktionsoperator () – Funktionsobjekte	395
8.3.17	Der Indexoperator [ ]	397
8.3.18	Die Operatoren new, new[], delete und delete []	398
<b>8.4</b>	<b>Übersicht der Operatoren in einer Tabelle</b>	400

## 9 Vererbung 407

---

<b>9.1</b>	<b>Grundlagen zur Vererbung</b>	407
<b>9.2</b>	<b>Abgeleitete Klassen implementieren</b>	410
9.2.1	Basisklasse erstellen	411
9.2.2	Von der Basisklasse ableiten	412
9.2.3	Erweiterte Klasse verwenden	414
9.2.4	Direkte und indirekte Basisklasse	415
9.2.5	Abgeleitete Klassen verbieten (C++11)	416
<b>9.3</b>	<b>Zugriffsschutz anpassen</b>	417
<b>9.4</b>	<b>Abgeleitete Klassen verwenden und erweitern</b>	423
9.4.1	Zugriff auf die erweiterten Mitglieder der abgeleiteten Klasse	423
9.4.2	Zugriff auf die Mitglieder der Basisklasse	424
9.4.3	Suche nach dem passenden Namen	426
9.4.4	Redefinition von Methoden	426

<b>9.5</b>	<b>Auf- und Abbau von abgeleiteten Klassen</b>	428
9.5.1	Aufbau von Objekten	428
9.5.2	Basisklasse initialisieren	429
9.5.3	Abbauen von Objekten	431
9.5.4	Konstruktoren erben (C++11)	432
<b>9.6</b>	<b>Implizite und explizite Typumwandlung in der Klassenhierarchie</b>	434
9.6.1	Abgeleitete Objekte an Basisklassenobjekte	434
9.6.2	Basisklassenobjekt an abgeleitete Objekte	435
<b>9.7</b>	<b>Polymorphie mithilfe von virtuellen Methoden</b>	436
<b>9.8</b>	<b>Virtueller Destruktor</b>	442
<b>9.9</b>	<b>Pure virtuelle Methoden und abstrakte Klassen</b>	445
9.9.1	Pure virtuelle Methoden	445
9.9.2	Abstrakte Klassen	446
9.9.3	Reine Interface-Klassen	450
9.9.4	Typinformationen zur Laufzeit	451
9.9.5	Überschreiben erzwingen mit override (C++11)	453
9.9.6	Nicht mehr überschreiben mit final (C++11)	455
<b>9.10</b>	<b>Mehrfachvererbung</b>	457
9.10.1	Mehrdeutigkeiten	461
9.10.2	Virtuelle Basisklassen (virtuelle Vererbung)	463

## **10 Ausnahmebehandlung (Exceptions)** 467

---

<b>10.1</b>	<b>Prinzip der Ausnahmebehandlung</b>	468
<b>10.2</b>	<b>Ausnahmebehandlung implementieren</b>	469
10.2.1	Ausnahme einleiten – try	470
10.2.2	Ausnahme werfen – throw	471
10.2.3	Ausnahme abfangen – catch	472
10.2.4	Mehrere Ausnahmen abfangen	474
10.2.5	Unbekannte oder alternative Ausnahmen abfangen	476
10.2.6	Aufräumarbeiten bei der Ausnahmebehandlung	477
10.2.7	Ausnahmen weiterleiten	478
10.2.8	try-catch verschachteln	479
10.2.9	Sinnvoller Einsatz von Ausnahmebehandlungen	480
<b>10.3</b>	<b>Ausnahmeklasse implementieren</b>	481
10.3.1	Ausnahmeklasse auslösen und abfangen	482
10.3.2	Ausnahmeklasse ableiten	485



<b>10.4</b>	<b>Standardausnahmen von C++</b>	<b>485</b>
10.4.1	Logische Fehler (logic_error)	487
10.4.2	Laufzeitfehler (runtime_error)	489
10.4.3	Weitere Standardfehlerklassen	493
10.4.4	Fehlermeldung mit what()	496
<b>10.5</b>	<b>Spezielle Fehlerbehandlungsfunktionen</b>	<b>496</b>
10.5.1	terminate() behandeln	497
10.5.2	std::uncaught_exception()	499
10.5.3	Ausnahme-Objekte zwischenspeichern (C++11)	499
10.5.4	noexcept (C++11)	501
10.5.5	Ausnahme-Spezifikation unexpected() behandeln (veraltet/deprecated)	503
<b>10.6</b>	<b>Gefahren bei der Ausnahmebehandlung</b>	<b>504</b>
10.6.1	Ausnahmen im Konstruktor	504
10.6.2	Ausnahmen im Destruktor	504
10.6.3	Aufräumprozess	504
10.6.4	Reservierter Speicher vom Heap	506

---

## 11 Template-Programmierung 507

---

<b>11.1</b>	<b>Funktions-Templates</b>	<b>508</b>
11.1.1	Funktions-Templates implementieren	509
11.1.2	Aufruf des Funktions-Templates	512
11.1.3	Funktions-Templates spezialisieren	515
11.1.4	Mehrere Template-Parameter verwenden	518
11.1.5	Explizite Template-Argumente	520
11.1.6	Methoden-Templates	522
<b>11.2</b>	<b>Klassen-Templates</b>	<b>523</b>
11.2.1	Klassen-Templates implementieren	524
11.2.2	Methoden von Klassen-Templates implementieren	525
11.2.3	Objekte aus Klassen-Templates erzeugen	527
11.2.4	Klassen-Templates mit mehreren formalen Datentypen	528
11.2.5	Klassen-Templates mit Non-Type-Parameter	530
11.2.6	Klassen-Templates mit Default-Wert	532
11.2.7	Klassen-Templates spezialisieren	535
11.2.8	Klassen-Template als Parameter an Funktion/Methode	538
<b>11.3</b>	<b>Templates mit variabler Argumentanzahl (C++11)</b>	<b>539</b>

## 12 Container, Iteratoren, Algorithmen und Hilfsmittel

545

<b>12.1</b>	<b>Grundlagen</b>	546
<b>12.2</b>	<b>Standardcontainer-Klassen</b>	550
12.2.1	Sequenzielle Containerklassen	550
12.2.2	Assoziative Containerklassen	560
12.2.3	Container für Bit-Manipulationen – bitset	571
12.2.4	Neue Möglichkeiten mit C++11	572
<b>12.3</b>	<b>Kleine Methodenübersicht aller Containerklassen</b>	578
12.3.1	Methodenübersicht von sequenzielle Containern	579
12.3.2	Methodenübersicht von assoziativen Containern	584
<b>12.4</b>	<b>Iteratoren</b>	590
12.4.1	Kategorien von Iteratoren	595
12.4.2	Iterator-Funktionen	597
12.4.3	Iterator-Adapter	600
<b>12.5</b>	<b>Algorithmen</b>	603
12.5.1	Bereich	604
12.5.2	Mehrere Bereiche	605
12.5.3	Algorithmen mit Prädikat	606
12.5.4	Algorithmen mit einfachen unären Funktionen	608
12.5.5	Funktionsobjekte	608
12.5.6	Lambda-Funktionen (C++11)	613
12.5.7	Übersicht zu den Algorithmen	619
<b>12.6</b>	<b>Hilfsmittel</b>	627
12.6.1	Template pair	627
12.6.2	Tupel, das bessere pair (C++11)	632
12.6.3	Vergleichsoperatoren für eigene Typen	638
12.6.4	std::bind (C++11)	639
12.6.5	Verschieben mit move() (C++11)	642
12.6.6	Smart Pointer (C++11)	642
12.6.7	Zeitbibliothek – <chrono> (C++11)	643

## 13 Die Stream-Ein-/Ausgabeklassen von C++

647

<b>13.1</b>	<b>Das Ein-/Ausgabe-Stream-Konzept von C++</b>	647
<b>13.2</b>	<b>Globale, vordefinierte Standard-Streams</b>	648

<b>13.3</b>	<b>Methoden für die Aus- und Eingabe von Streams</b>	<b>650</b>
13.3.1	Methoden für die unformatierte Ausgabe	651
13.3.2	Methoden für die (unformatierte) Eingabe	652
<b>13.4</b>	<b>Fehlerbehandlung bzw. Zustand von Streams</b>	<b>655</b>
<b>13.5</b>	<b>Streams manipulieren und formatieren</b>	<b>659</b>
13.5.1	Manipulatoren	660
13.5.2	Eigene Manipulatoren ohne Argumente erstellen	666
13.5.3	Eigene Manipulatoren mit Argumenten erstellen	667
13.5.4	Format-Flags direkt ändern	669
<b>13.6</b>	<b>Streams für die Datei-Ein-/Ausgabe</b>	<b>672</b>
13.6.1	Streams für die Datei-Ein-/Ausgabe	673
13.6.2	Verbindung zu einer Datei herstellen	673
13.6.3	Lesen und Schreiben	678
13.6.4	Wahlfreier Zugriff	686
<b>13.7</b>	<b>Streams für Strings</b>	<b>687</b>
<b>13.8</b>	<b>Stream-Puffer</b>	<b>692</b>

---

## **14 Reguläre Ausdrücke (C++11)** **695**

---

<b>14.1</b>	<b>Grammatik-Grundlagen zu den regulären Ausdrücken</b>	<b>696</b>
<b>14.2</b>	<b>Ein Objekt für reguläre Ausrücke erzeugen</b>	<b>699</b>
<b>14.3</b>	<b>Suchergebnis analysieren</b>	<b>702</b>
<b>14.4</b>	<b>Algorithmen für reguläre Ausdrücke</b>	<b>705</b>
14.4.1	Genaue Treffer mit <code>regex_match</code>	706
14.4.2	Erweiterte Suche mit <code>regex_search</code>	709
14.4.3	Ersetzen mit <code>regex_replace</code>	711
14.4.4	Suchen mit <code>regex_iterator</code> und <code>regex_token_iterator</code>	713

---

## **15 Multithreading (C++11)** **719**

---

<b>15.1</b>	<b>Die Grundlagen</b>	<b>719</b>
<b>15.2</b>	<b>Threads erzeugen</b>	<b>722</b>
15.2.1	Argumente für den Thread	723
15.2.2	Methoden für Threads	727
15.2.3	Funktionen für Threads	732
15.2.4	Wie viele Threads sollen es sein?	733

<b>15.3</b>	<b>Gemeinsame Daten synchronisieren</b>	734
15.3.1	Schutz der Daten über einen Mutex	736
15.3.2	Schutz der Daten über Locks	741
15.3.3	Daten sicher initialisieren	747
15.3.4	Statische Variablen	749
<b>15.4</b>	<b>Threadlokale Daten</b>	750
<b>15.5</b>	<b>Threads synchronisieren</b>	751
<b>15.6</b>	<b>Asynchrones Arbeiten (Future und Promise)</b>	755
15.6.1	Futures	757
15.6.2	Promise	760
15.6.3	Methoden für future und promise	766
<b>15.7</b>	<b>packaged_task</b>	767
<b>16</b>	<b>Weitere Neuerungen in C++11</b>	773
<hr/>		
<b>16.1</b>	<b>Move-Semantik und Perfect Forwarding</b>	773
16.1.1	Exkurs: LValue und RValue	774
16.1.2	RValue-Referenz und Move-Semantik	776
16.1.3	Perfect Forwarding	781
<b>16.2</b>	<b>Benutzerdefinierte Literale (C++11)</b>	784
<b>16.3</b>	<b>Smart Pointer (C++11)</b>	789
16.3.1	Shared Pointer	791
16.3.2	Weak Pointer	801
16.3.3	Unique Pointer	804
16.3.4	Weiteres zu den Smart Pointern	810
<b>16.4</b>	<b>Zeitbibliothek</b>	812
16.4.1	Zeitgeber (Clock)	812
16.4.2	Zeitpunkt (time_point) und Systemzeit (system_clock)	813
16.4.3	Zeitdauer (duration)	815
<b>16.5</b>	<b>Type-Traits</b>	820
<b>16.6</b>	<b>POD (Plain Old Data) (C++11)</b>	826
<b>16.7</b>	<b>std::initializer_list</b>	828

<b>17.1</b>	<b>Ein erstes Programmbeispiel schreiben</b>	836
17.1.1	Kurze Übersicht zur Oberfläche von Qt Creator	837
17.1.2	Ein einfaches Projekt erstellen	838
<b>17.2</b>	<b>Signale und Slots</b>	846
17.2.1	Verbindung zwischen Signal und Slot herstellen	847
17.2.2	Signal und Slot mithilfe der Qt-Referenz ermitteln	848
<b>17.3</b>	<b>Klassenhierarchie von Qt</b>	866
17.3.1	Basisklasse QObject	866
17.3.2	Klassenhierarchie	866
17.3.3	Speicherverwaltung	869
<b>17.4</b>	<b>Eigene Widgets erstellen</b>	870
<b>17.5</b>	<b>Eigene Widgets mit dem Qt Designer erstellen</b>	873
<b>17.6</b>	<b>Widgets anordnen</b>	880
17.6.1	Grundlegende Widgets für das Layout	881
<b>17.7</b>	<b>Dialoge erstellen mit QDialog</b>	888
<b>17.8</b>	<b>Vorgefertigte Dialoge von Qt</b>	896
17.8.1	QMessageBox – der klassische Nachrichtendialog	896
17.8.2	QFileDialog – Dialog zur Dateiauswahl	902
17.8.3	QInputDialog – Dialog zur Eingabe von Daten	907
17.8.4	Weitere Dialoge	909
<b>17.9</b>	<b>Eigenen Dialog mit dem Qt Designer erstellen</b>	909
<b>17.10</b>	<b>Grafische Bedienelemente von Qt (Qt-Widgets)</b>	930
17.10.1	Schaltflächen (Basisklasse QAbstractButton)	930
17.10.2	Container-Widgets (Behälter-Widgets)	940
17.10.3	Widgets zur Zustandsanzeige	950
17.10.4	Widgets zur Eingabe	953
17.10.5	Online-Hilfen	966
<b>17.11</b>	<b>Anwendungen in einem Hauptfenster</b>	969
17.11.1	Die Klasse für das Hauptfenster QMainWindow	969
17.11.2	Eine Menüleiste für das Hauptfenster (QMenu und QMenuBar)	971
17.11.3	Eine Statusleiste mit QStatusBar	980
17.11.4	Eine Werkzeugleiste mit der Klasse QToolBar	985
17.11.5	An- und abdockbare Widgets im Hauptfenster mit QDockWidget	987
17.11.6	Einstellungen sichern mit QSettings	990
17.11.7	Kompletter Quellcode des Texteditors	996

<b>17.12 Anwendung lokalisieren mit Qt Linguistic .....</b>	<b>998</b>
<b>17.13 Anwendungen in einem Hauptfenster mit dem Qt Designer .....</b>	<b>1006</b>
<b>17.14 Qt Designer vs. handgeschrieben .....</b>	<b>1018</b>
<b>17.15 Dinge, die man wissen sollte ... ..</b>	<b>1018</b>
17.15.1 QApplication und QCoreApplication .....	1019
17.15.2 Konsolenanwendungen mit Qt .....	1019
17.15.3 Wenn keine Icons angezeigt werden ... ..	1021
17.15.4 Das Ressourcen-System .....	1022
<b>17.16 Klassen und Typen zum Speichern von Daten .....</b>	<b>1027</b>
17.16.1 Qt-eigene Typdefinitionen .....	1027
17.16.2 QString .....	1028
17.16.3 QStringList .....	1030
17.16.4 QVariant .....	1031
17.16.5 Typen für Datum und Uhrzeit .....	1031
<b>17.17 Fazit .....</b>	<b>1032</b>

# Anhang

<b>A.1 Operatoren in C++ und deren Bedeutung (Übersicht) .....</b>	<b>1033</b>
<b>A.2 Vorrangtabelle der Operatoren .....</b>	<b>1035</b>
<b>A.3 Schlüsselwörter von C++ .....</b>	<b>1037</b>
<b>A.4 Informationsspeicherung .....</b>	<b>1037</b>
<b>A.5 Zeichensätze .....</b>	<b>1044</b>
<b>Index .....</b>	<b>1051</b>