

Inhaltsverzeichnis

Vorwort zur zweiten Auflage	1
Vorwort zur ersten Auflage	2
1 Über dieses Buch	3
1.1 Warum dieses Buch?	3
1.2 Voraussetzungen	4
1.3 Systematik	4
1.4 Wie liest man dieses Buch?	6
1.5 Zugriff auf die Quellen zu den Beispielen	6
1.6 Anregungen und Kritik	6
2 Einleitung: C++ und objektorientierte Programmierung	9
2.1 Die Sprache C++	9
2.1.1 Designkriterien	9
2.1.2 Sprachversionen	10
2.2 C++ als objektorientierte Programmiersprache	10
2.2.1 Objekte, Klassen und Instanzen	11
2.2.2 Klassen in C++	13
2.2.3 Datenkapselung	15
2.2.4 Vererbung	17
2.2.5 Polymorphie	18
2.3 Weitere Konzepte von C++	20
2.3.1 Ausnahmebehandlung	20
2.3.2 Templates	21
2.3.3 Namensbereiche	22
2.4 Terminologie	22

3	Grundkonzepte von C++-Programmen	25
3.1	Das erste Programm	26
3.1.1	„Hallo, Welt!“	26
3.1.2	Kommentare in C++	27
3.1.3	Hauptfunktion <code>main()</code>	28
3.1.4	Ein-/Ausgaben	29
3.1.5	Namensbereiche	30
3.1.6	Zusammenfassung	31
3.2	Datentypen, Operatoren, Kontrollstrukturen	33
3.2.1	Ein erstes Programm, das wirklich etwas berechnet	33
3.2.2	Fundamentale Datentypen	36
3.2.3	Operatoren	39
3.2.4	Kontrollstrukturen	45
3.2.5	Zusammenfassung	49
3.3	Funktionen und Module	50
3.3.1	Headerdateien	50
3.3.2	Quelldatei mit der Implementierung	52
3.3.3	Quelldatei mit dem Aufruf	52
3.3.4	Übersetzen und binden	54
3.3.5	Dateiendungen	54
3.3.6	Systemdateien	55
3.3.7	Präprozessor	55
3.3.8	Namensbereiche	59
3.3.9	Das Schlüsselwort <code>static</code>	60
3.3.10	Zusammenfassung	62
3.4	Strings	63
3.4.1	Ein erstes einfaches Beispielprogramm mit Strings	63
3.4.2	Ein weiteres Beispielprogramm mit Strings	66
3.4.3	String-Operationen im Überblick	71
3.4.4	Strings und C-Strings	72
3.4.5	Zusammenfassung	72
3.5	Verarbeitung von Mengen	74
3.5.1	Beispielprogramm mit Vektoren	74
3.5.2	Beispielprogramm mit Deques	76
3.5.3	Vektor versus Deque	77
3.5.4	Iteratoren	78
3.5.5	Beispielprogramm mit einer Liste	80
3.5.6	Beispielprogramme mit assoziativen Containern	81
3.5.7	Algorithmen	86
3.5.8	Algorithmen mit mehreren Bereichen	90

3.5.9	Stream-Iteratoren	94
3.5.10	Ausblick	96
3.5.11	Zusammenfassung	96
3.6	Ausnahmebehandlung	98
3.6.1	Motivation für das Konzept der Ausnahmebehandlung	98
3.6.2	Das Konzept der Ausnahmebehandlung	100
3.6.3	Standard-Ausnahmeklassen	101
3.6.4	Ausnahmebehandlung am Beispiel	101
3.6.5	Behandlung nicht abgefangener Ausnahmen	106
3.6.6	Hilfsfunktionen zur Behandlung von Ausnahmen	107
3.6.7	Zusammenfassung	108
3.7	Zeiger, Felder und C-Strings	110
3.7.1	Zeiger	110
3.7.2	Felder	112
3.7.3	C-Strings	115
3.7.4	Zusammenfassung	119
3.8	Freispeicherverwaltung mit <code>new</code> und <code>delete</code>	120
3.8.1	Der Operator <code>new</code>	121
3.8.2	Der Operator <code>delete</code>	121
3.8.3	Dynamische Speicherverwaltung für Felder	122
3.8.4	Fehlerbehandlung für <code>new</code>	124
3.8.5	Zusammenfassung	125
3.9	Kommunikation mit der Außenwelt	126
3.9.1	Argumente aus dem Programmaufruf	126
3.9.2	Zugriff auf Umgebungsvariablen	127
3.9.3	Abbruch von Programmen	128
3.9.4	Aufruf von weiteren Programmen	129
3.9.5	Zusammenfassung	129
4	Programmieren von Klassen	131
4.1	Die erste Klasse: Bruch	132
4.1.1	Vorüberlegungen zur Implementierung	132
4.1.2	Deklaration der Klasse Bruch	135
4.1.3	Die Klassenstruktur	136
4.1.4	Elementfunktionen	139
4.1.5	Konstruktoren	139
4.1.6	Überladen von Funktionen	141
4.1.7	Implementierung der Klasse Bruch	142
4.1.8	Anwendung der Klasse Bruch	147
4.1.9	Erzeugung temporärer Objekte	153
4.1.10	UML-Notation	153
4.1.11	Zusammenfassung	154

4.2	Operatoren für Klassen	156
4.2.1	Deklaration von Operatorfunktionen	156
4.2.2	Implementierung von Operatorfunktionen	159
4.2.3	Anwendung von Operatorfunktionen	166
4.2.4	Globale Operatorfunktionen	168
4.2.5	Grenzen bei der Definition eigener Operatoren	169
4.2.6	Besonderheiten spezieller Operatoren	170
4.2.7	Zusammenfassung	174
4.3	Laufzeit- und Codeoptimierungen	175
4.3.1	Die Klasse <code>Bruch</code> mit ersten Optimierungen	175
4.3.2	Default-Argumente	178
4.3.3	Inline-Funktionen	180
4.3.4	Optimierungen aus Anwendersicht	182
4.3.5	Using-Direktiven	183
4.3.6	Deklarationen zwischen Anweisungen	185
4.3.7	Copy-Konstruktoren	186
4.3.8	Zusammenfassung	187
4.4	Referenzen und Konstanten	189
4.4.1	Copy-Konstruktor und Parameterübergabe	189
4.4.2	Referenzen	190
4.4.3	Konstanten	193
4.4.4	Konstanten-Elementfunktionen	195
4.4.5	Die Klasse <code>Bruch</code> mit Referenzen	195
4.4.6	Zeiger auf Konstanten und Zeigerkonstanten	199
4.4.7	Zusammenfassung	201
4.5	Ein- und Ausgabe mit Streams	202
4.5.1	Streams	202
4.5.2	Umgang mit Streams	203
4.5.3	Zustand von Streams	210
4.5.4	I/O-Operatoren für eigene Datentypen	212
4.5.5	Zusammenfassung	223
4.6	Freunde und andere Typen	224
4.6.1	Automatische Typumwandlungen	224
4.6.2	Schlüsselwort <code>explicit</code>	226
4.6.3	Friend-Funktionen	227
4.6.4	Konvertierungsfunktionen	234
4.6.5	Probleme bei der automatischen Typumwandlung	235
4.6.6	Andere Anwendungen des Schlüsselworts <code>friend</code>	238
4.6.7	<code>friend</code> kontra objektorientierte Programmierung	239
4.6.8	Zusammenfassung	240

4.7	Ausnahmebehandlung für Klassen	241
4.7.1	Motivation für eine Ausnahmebehandlung in der Klasse Bruch	241
4.7.2	Ausnahmebehandlung am Beispiel der Klasse Bruch	242
4.7.3	Fehlerklassen	250
4.7.4	Weitergabe von Fehlern	251
4.7.5	Ausnahmen in Destruktoren	251
4.7.6	Ausnahmen in Schnittstellen-Deklarationen	252
4.7.7	Hierarchien von Fehlerklassen	253
4.7.8	Design von Fehlerklassen	257
4.7.9	Standard-Ausnahmen auslösen	258
4.7.10	Zusammenfassung	259
5	Vererbung und Polymorphie	261
5.1	Einfache Vererbung	263
5.1.1	Die Klasse Bruch als Basisklasse	263
5.1.2	Vorüberlegungen zur abgeleiteten Klasse KBruch	266
5.1.3	Deklaration der abgeleiteten Klasse KBruch	267
5.1.4	Vererbung und Konstruktoren	270
5.1.5	Implementierung von abgeleiteten Klassen	273
5.1.6	Anwendung von abgeleiteten Klassen	276
5.1.7	Konstruktoren für Objekte der Basisklasse	278
5.1.8	Zusammenfassung	279
5.2	Virtuelle Funktionen	280
5.2.1	Probleme beim Überschreiben von Funktionen der Basisklasse	280
5.2.2	Statisches und dynamisches Binden von Funktionen	283
5.2.3	Überladen kontra Überschreiben	288
5.2.4	Zugriff auf Parameter der Basisklasse	289
5.2.5	Virtuelle Destruktoren	290
5.2.6	Vererbung richtig angewendet	291
5.2.7	Weitere Fallen beim Überschreiben von Funktionen	297
5.2.8	Private Vererbung und reine Zugriffsdeklarationen	299
5.2.9	Zusammenfassung	303
5.3	Polymorphie	304
5.3.1	Was ist Polymorphie?	304
5.3.2	Polymorphie in C++	305
5.3.3	Polymorphie in C++ an einem Beispiel	306
5.3.4	Die abstrakte Basisklasse GeoObj	310
5.3.5	Anwendung von Polymorphie in Klassen	319
5.3.6	Polymorphie ist keine feste Fallunterscheidung	325
5.3.7	Rückumwandlung eines Objekts in seine Klasse	326
5.3.8	Design by Contract	330
5.3.9	Zusammenfassung	331

5.4	Mehrfachvererbung	333
5.4.1	Beispiel für Mehrfachvererbung	333
5.4.2	Virtuelle Basisklassen	339
5.4.3	Das Problem der Identität	343
5.4.4	Dieselbe Basisklasse mehrfach ableiten	346
5.4.5	Zusammenfassung	347
5.5	Design-Fallen bei der Vererbung	348
5.5.1	Vererbung kontra Verwendung	348
5.5.2	Design-Fehler: Einschränkende Vererbung	348
5.5.3	Design-Fehler: Wertverändernde Vererbung	350
5.5.4	Design-Fehler: Wertinterpretierende Vererbung	351
5.5.5	„Vermeide Vererbung!“	352
5.5.6	Zusammenfassung	353
6	Dynamische und statische Komponenten	355
6.1	Dynamische Komponenten	356
6.1.1	Implementierung der Klasse <code>String</code>	356
6.1.2	Konstruktoren bei dynamischen Komponenten	362
6.1.3	Implementierung eines Copy-Konstruktors	364
6.1.4	Destruktoren	364
6.1.5	Implementierung des Zuweisungsoperators	365
6.1.6	Weitere Operatoren	367
6.1.7	Einlesen eines Strings	370
6.1.8	Kommerzielle Implementierung von <code>String</code> -Klassen	372
6.1.9	Weitere Anwendungsmöglichkeiten dynamischer Komponenten	374
6.1.10	Zusammenfassung	376
6.2	Weitere Aspekte dynamischer Komponenten	377
6.2.1	Dynamische Komponenten bei konstanten Objekten	377
6.2.2	Konvertierungsfunktionen für dynamische Komponenten	380
6.2.3	Konvertierungsfunktionen für Bedingungen	382
6.2.4	Konstanten werden zu Variablen	385
6.2.5	Vordefinierte Funktionen verbieten	388
6.2.6	Proxy-Klassen	389
6.2.7	Ausnahmebehandlung mit Parametern	391
6.2.8	Zusammenfassung	395
6.3	Vererbung von Klassen mit dynamischen Komponenten	397
6.3.1	Die Klasse <code>Bsp::String</code> als Basisklasse	397
6.3.2	Die abgeleitete Klasse <code>FarbString</code>	399
6.3.3	Ableiten von Friend-Funktionen	402
6.3.4	Quelldatei der abgeleiteten Klasse <code>FarbString</code>	405
6.3.5	Anwendung der Klasse <code>FarbString</code>	406
6.3.6	Ableiten der Spezialfunktionen für dynamische Komponenten	407
6.3.7	Zusammenfassung	409

6.4	Klassen verwenden Klassen	410
6.4.1	Objekte als Komponenten anderer Klassen	410
6.4.2	Implementierung der Klasse <code>Person</code>	410
6.4.3	Zusammenfassung	417
6.5	Statische Komponenten und Hilfstypen	418
6.5.1	Statische Klassenkomponenten	418
6.5.2	Typdeklarationen innerhalb von Klassen	424
6.5.3	Aufzählungstypen als statische Klassenkonstanten	427
6.5.4	Eingebettete und lokale Klassen	428
6.5.5	Zusammenfassung	429
7	Templates	431
7.1	Motivation für Templates	431
7.1.1	Terminologie	432
7.2	Funktionstemplates	433
7.2.1	Definition von Funktionstemplates	433
7.2.2	Aufruf von Funktionstemplates	434
7.2.3	Praktische Hinweise zum Umgang mit Templates	435
7.2.4	Automatische Typumwandlung bei Templates	435
7.2.5	Überladen von Templates	436
7.2.6	Lokale Variablen	439
7.2.7	Zusammenfassung	439
7.3	Klassentemplates	440
7.3.1	Implementierung des Klassentemplate <code>Stack</code>	440
7.3.2	Anwendung des Klassentemplate <code>Stack</code>	444
7.3.3	Spezialisieren von Klassentemplates	445
7.3.4	Default Template-Parameter	448
7.3.5	Zusammenfassung	451
7.4	Werte als Template-Parameter	452
7.4.1	Beispiel für die Verwendung von Werte-Parametern	452
7.4.2	Einschränkungen bei Werte-Parametern	454
7.4.3	Zusammenfassung	455
7.5	Weitere Aspekte von Templates	456
7.5.1	Das Schlüsselwort <code>typename</code>	456
7.5.2	Komponenten als Templates	457
7.5.3	Statische Polymorphie mit Templates	460
7.5.4	Zusammenfassung	464
7.6	Templates in der Praxis	465
7.6.1	Übersetzen von Template-Code	465
7.6.2	Fehlerbehandlung	470
7.6.3	Zusammenfassung	472

8	Die Standard-I/O-Bibliothek im Detail	473
8.1	Die Standard-Stream-Klassen	474
8.1.1	Stream-Klassen und -Objekte	474
8.1.2	Fehlerzustände, Stream-Status	476
8.1.3	Standardoperatoren	479
8.1.4	Standardfunktionen	480
8.1.5	Manipulatoren	484
8.1.6	Formatdefinitionen	486
8.1.7	Setzen und Abfragen von Formatflags	486
8.1.8	Internationalisierung	496
8.1.9	Zusammenfassung	499
8.2	Dateizugriff	500
8.2.1	Stream-Klassen für Dateien	500
8.2.2	Beispiel für die Verwendung der Stream-Klassen für Dateien	500
8.2.3	Datei-Flags	503
8.2.4	Explizites Öffnen und Schließen	504
8.2.5	Wahlfreier Zugriff	506
8.2.6	Umleiten der Standardkanäle in Dateien	508
8.2.7	Zusammenfassung	510
8.3	Stream-Klassen für Strings	511
8.3.1	String-Stream-Klassen	511
8.3.2	Lexical-Cast-Operator	514
8.3.3	char*-Stream-Klassen	516
8.3.4	Zusammenfassung	518
9	Weitere Sprachmittel und Details	519
9.1	Weitere Details zur Standardbibliothek	520
9.1.1	Operationen von Vektoren	520
9.1.2	Gemeinsame Operationen aller STL-Container	526
9.1.3	Liste aller STL-Algorithmen	528
9.1.4	Numerische Limits	533
9.1.5	Zusammenfassung	538
9.2	Definition besonderer Operatoren	539
9.2.1	Smart-Pointer	539
9.2.2	Funktionsobjekte	543
9.2.3	Zusammenfassung	547
9.3	Weitere Aspekte von <code>new</code> und <code>delete</code>	548
9.3.1	Nothrow-Versionen von <code>new</code> und <code>delete</code>	548
9.3.2	Placement-New	548
9.3.3	New-Handler	549
9.3.4	Überladen von <code>new</code> und <code>delete</code>	554

9.3.5	Operator <code>new</code> mit zusätzlichen Parametern	557
9.3.6	Zusammenfassung	558
9.4	Funktions- und Komponentenzeiger	559
9.4.1	Funktionszeiger	559
9.4.2	Komponentenzeiger	560
9.4.3	Komponentenzeiger für Schnittstellen nach außen	563
9.4.4	Zusammenfassung	565
9.5	Kombination mit C-Code	566
9.5.1	Externe Bindung	566
9.5.2	Headerdateien für C und C++	567
9.5.3	Übersetzen von <code>main()</code>	567
9.5.4	Zusammenfassung	567
9.6	Weitere Schlüsselwörter	568
9.6.1	Varianten mit <code>union</code>	568
9.6.2	Aufzählungstypen mit <code>enum</code>	568
9.6.3	Das Schlüsselwort <code>volatile</code>	570
9.6.4	Zusammenfassung	570
10	Zusammenfassung	571
10.1	Hierarchie der C++-Operatoren	571
10.2	Klassenspezifische Eigenschaften von Operationen	574
10.3	Regeln zur automatischen Typumwandlung	575
10.4	Sinnvolle Programmierrichtlinien und Konventionen	575
	Literaturverzeichnis	579
	Glossar	583
	Index	589