

Stefan Kuhlins · Martin Schader

Die C++-Standardbibliothek

Einführung und Nachschlagewerk

Vierte, durchgesehene Auflage

Mit 77 Abbildungen und 37 Tabellen



Springer

Vorwort

Warum wird eine Standardbibliothek für C++ gebraucht? Programmentwickler benötigen immer wieder die gleichen Datenstrukturen wie z. B. dynamische Felder. Aufgrund verschiedener konkreter Anforderungen werden einzelne Implementierungen aber in der Regel sehr unterschiedlich ausfallen. Dadurch werden Programme anderer Entwickler schwer verständlich, und beim Wechsel der eingesetzten Bibliotheken entsteht erheblicher Lernaufwand. Durch eine Standardbibliothek lassen sich diese Probleme in den Griff bekommen. Um sich an die verschiedenen Anforderungen, unter denen eine Bibliothek zum Einsatz kommt, anpassen zu können, ist eine wichtige Voraussetzung für den Erfolg einer Standardbibliothek, dass sie flexibel und erweiterbar ist. Mit der *Standard Template Library* (STL), dem Kernstück der aktuellen C++-Standardbibliothek, wird dieses Ziel erreicht.

Darüber hinaus ist die STL äußerst effizient, so dass sich eine hervorragende Performance erreichen lässt. Allerdings setzt ein Gewinn bringender Einsatz der STL ein tiefes Verständnis für das Design der STL voraus. Um die STL zu verstehen, genügt es nicht, lediglich eine Funktionsreferenz zu lesen. Deshalb haben wir in diesem Buch besonderen Wert darauf gelegt, die Konzepte und Funktionsweisen der STL zu erläutern sowie auf typische Stolpersteine hinzuweisen. In diesem Zusammenhang ist vor allem Kapitel 2 hervorzuheben, das sich mit der Konzeption und Entwicklung der STL beschäftigt.

Bei der Beschreibung der einzelnen Komponenten der STL – Container, Iteratoren, Algorithmen, Funktionsobjekte usw. – gehen wir auch auf Implementierungsdetails ein, die vom Standard zwar nicht festgeschrieben sind, sich in der Regel aber aufgrund von Vorgaben für die Komplexität automatisch ergeben. Das Wissen um Implementierungsdetails unterstützt eine effiziente Benutzung der STL. Ferner demonstrieren zahlreiche kleine Anwendungsbeispiele den Gebrauch.

Im Schatten der STL existieren viele weitere interessante Komponenten der C++-Standardbibliothek, zu denen unter anderem *Streams*, *Strings*, *Auto-Pointer*, *Bitsets* und komplexe Zahlen gehören. Auch für diese Klassen liefern wir die für den praktischen Einsatz notwendigen Informationen.

Unsere Ausführungen basieren auf dem aktuellen C++-Standard, der unter der Bezeichnung ISO/IEC 14882:2003, *International Standard for the C++ Programming Language (Second Edition)*, erschienen ist. Das Dokument kann beim *American National Standards Institute* (ANSI) bezogen werden. Der Download einer elektronischen Version ist derzeit für \$18 von <http://www.ansi.org/> möglich. Eine ältere Version vom Dezember 1996 ist unter <http://www.dkuug.dk/jtc1/sc22/open/n2356/> frei zugänglich. Vom C++-Standardkomitee unter [v](http://www.open-</p></div><div data-bbox=)

std.org/jtc1/sc22/wg21/ diskutierte Korrekturen und Verbesserungsvorschläge haben wir weitgehend berücksichtigt.

Gegenüber der Urfassung der STL, die von *Alexander Stepanov* und *Meng Lee* bei *Hewlett Packard* als Technischer Report HPL-94-34 mit dem Titel „*The Standard Template Library*“ im April 1994 publiziert wurde, sind während der Standardisierung unzählige Änderungen vorgenommen worden. An Stellen, an denen es uns nützlich erscheint, weisen wir auf frühere Besonderheiten hin, damit sich unsere Programmbeispiele an ältere Versionen der STL anpassen lassen.

Da wir hier den C++-Standard beschreiben, sollten sich alle unsere Beispielprogramme mit jedem standardkonformen C++-Compiler erfolgreich übersetzen lassen. Zum Testen unserer Programmbeispiele haben wir die folgenden C++-Compiler eingesetzt (in alphabetischer Reihenfolge):

- Borland C++ unter Windows
- GNU g++ unter Linux und Solaris sowie Windows (cygwin)
- Kuck & Associates, Inc. KAI C++ unter Linux
- Microsoft Visual C++ unter Windows

Jedes Beispielprogramm lässt sich mit mindestens einem dieser Compiler erfolgreich übersetzen. Darüber hinaus decken die im Text angegebenen Definitionen für Klassen, Funktionen usw. einen Großteil der STL ab. Insbesondere älteren Implementierungen kann man damit auf die „Sprünge“ helfen, so dass man seine Programme gemäß dem aktuellen Standard schreiben kann und späterer Umstellungsaufwand entfällt.

Im Internet existieren frei verfügbare Implementierungen der STL und interessante Erweiterungen. Derzeit erscheinen uns insbesondere *STLport* und *Boost* (<http://www.stlport.org/> sowie <http://www.boost.org/>) erwähnenswert.


Die C++-Standardbibliothek reizt die Programmiersprache C++ voll aus. Das geht sogar so weit, dass die Sprache für die Bibliothek erweitert wurde; *Member-Templates* sind dafür ein Beispiel (siehe Seite 70). Um die in diesem Buch dargestellten Themen verstehen zu können, muss man über fundierte C++-Kenntnisse verfügen. Zum Erlernen der Sprache C++ und als Nachschlagewerk empfehlen wir gerne unser Buch „*Programmieren in C++*“, das ebenfalls im Springer-Verlag erschienen ist.

Wir gehen davon aus, dass das Buch in der vorgegebenen Reihenfolge durchgearbeitet wird. Um langweilige Wiederholungen zu vermeiden, werden z. B. Elementfunktionen von Containerklassen nur beim ersten Auftreten im Text ausführlich und mit Beispielen erläutert.

Jeweils am Kapitelende haben wir einige Aufgaben zusammengestellt. Lösungsvorschläge befinden sich in Kapitel 15. Die meisten Aufgaben beschäftigen sich mit Themen, die über das im Text Gesagte hinausgehen. Dementsprechend

ist der Schwierigkeitsgrad recht hoch, und ein Blick auf die Lösungen lohnt sich.

Für Informationen rund um das Buch haben wir im *World Wide Web* eine Homepage unter <http://www.wifo.uni-mannheim.de/veroeff/stl/> eingerichtet. Dort finden Sie:

- den Sourcecode aller mit dem Diskettensymbol  gekennzeichneten Programme und Lösungen;
- sämtliche Programmfragmente – Sie brauchen also nichts abzutippen;
- aktuelle Ergänzungen und Korrekturen.

Außerdem können Sie sich dort in unsere *Mailing-Liste* eintragen, damit wir Sie mit den jeweils neuesten Informationen zum Buch auf dem Laufenden halten können.

Über Anregungen unserer Leserinnen und Leser an unsere Postanschrift oder als E-Mail an stlbuch@wifo.uni-mannheim.de würden wir uns freuen. Wer uns einen Fehler (gleichgültig welcher Art) zuerst mitteilt, den erwähnen wir namentlich im *World Wide Web* auf unseren Seiten zum Buch.

Wir danken unseren Lesern – und hier insbesondere Rüdiger Dreier, Frank Fasse, Christian Hoffmann, Wolfgang Kaisers, Arnold Ludwig, Thomas Mehring, Olaf Raeke und Peter Schatte – für hilfreiche Hinweise zu den ersten drei Auflagen. Außerdem gilt unser Dank dem Springer-Verlag für die, wie immer, sehr gute Zusammenarbeit.

Stefan Kuhlins, Martin Schader

Inhaltsverzeichnis

1	Vorbemerkungen	1
1.1	namespace std	1
1.2	Header-Dateien	2
1.3	Eigenschaften	2
1.4	Kanonische Klassen	4
1.5	Komplexität und Aufwand.....	5
2	Konzeption und Entwicklung der STL	7
2.1	Eine Feldklasse	7
2.2	Eine Listenklasse	8
2.3	Folgerungen.....	9
2.4	Standardisierung der Suchfunktion.....	9
2.5	Die Schnittstelle eines Iterators.....	11
2.6	Ein Iterator für die Feldklasse	12
2.7	Ein Iterator für die Listenklasse.....	13
2.8	Zusammenfassung	13
2.9	const-Korrektheit.....	14
2.10	Flexible Suche mit Funktionsobjekten	16
2.11	Kleinste Bausteine	18
2.12	Programmübersetzungs- und -laufzeit	20
2.13	Der Rückgabetypp für eine Zählfunktion	22
2.14	Fehlerquellen	25
2.15	Ist die STL objektorientiert?	27
2.16	Einsatz der Standardbibliothek	29
2.17	Aufgaben.....	30
3	Funktionsobjekte	33
3.1	Basisklassen für Funktionsobjekte	33
3.2	Arithmetische, logische und Vergleichsoperationen	35
3.3	Projektionen	38
3.3.1	binder1st	38
3.3.2	bind1st	39
3.3.3	binder2nd	39
3.3.4	bind2nd	40
3.3.5	Beispiel.....	40
3.4	Negativierer.....	41
3.4.1	unary_negate	41
3.4.2	not1	42
3.4.3	binary_negate	42
3.4.4	not2	42
3.5	Adapter für Funktionszeiger	43
3.5.1	pointer_to_unary_function	44

3.5.2	ptr_fun für einstellige Funktionen.....	44
3.5.3	pointer_to_binary_function.....	44
3.5.4	ptr_fun für zweistellige Funktionen	45
3.6	Adapter für Zeiger auf Elementfunktionen.....	45
3.6.1	mem_fun_ref_t und const_mem_fun_ref_t	47
3.6.2	mem_fun_ref für Elementfunktionen ohne Argument.....	47
3.6.3	mem_fun1_ref_t und const_mem_fun1_ref_t.....	48
3.6.4	mem_fun_ref für Elementfunktionen mit Argument	49
3.6.5	mem_fun_t und const_mem_fun_t.....	49
3.6.6	mem_fun für Elementfunktionen ohne Argument	50
3.6.7	mem_fun1_t und const_mem_fun1_t	50
3.6.8	mem_fun für Elementfunktionen mit Argument.....	51
3.7	Funktionen zusammensetzen	51
3.8	Aufgaben	52
4	Hilfsmittel	55
4.1	Vergleichsoperatoren.....	55
4.2	pair	56
4.3	Aufgaben	57
5	Container	59
5.1	vector	60
5.2	Allgemeine Anforderungen an Container.....	62
5.3	Anforderungen an reversible Container	68
5.4	Anforderungen an sequenzielle Container	68
5.5	Optionale Anforderungen an sequenzielle Container	73
5.6	Weitere Funktionen.....	76
5.7	deque	79
5.8	list	84
5.9	Auswahl nach Aufwand.....	90
5.10	Aufgaben	91
6	Containeradapter	93
6.1	stack.....	93
6.2	queue	95
6.3	priority_queue	96
6.4	Aufgaben	99
7	Assoziative Container	101
7.1	map	102
7.2	Anforderungen an assoziative Container	104
7.3	Der Indexoperator der Klasse map	109
7.4	multimap	110
7.5	set.....	113
7.6	multiset	116
7.7	Elemente in set und multiset modifizieren.....	118
7.8	Übersicht der Container.....	120
7.9	Aufgaben	120

8	Iteratoren	123
8.1	Iteratoranforderungen	123
8.2	Input-Iteratoren	125
8.3	Output-Iteratoren	126
8.4	Forward-Iteratoren	127
8.5	Bidirectional-Iteratoren	129
8.6	Random-Access-Iteratoren	129
8.7	Übersicht über die Iteratorkategorien	131
8.8	Hilfsklassen und -funktionen für Iteratoren	132
8.8.1	iterator_traits	132
8.8.2	Die Basisklasse iterator	134
8.8.3	Iteratorfunktionen	135
8.8.3.1	advance	135
8.8.3.2	distance	136
8.9	Reverse-Iteratoren	137
8.10	Insert-Iteratoren	140
8.10.1	back_insert_iterator	141
8.10.2	front_insert_iterator	142
8.10.3	insert_iterator	143
8.11	Stream-Iteratoren	144
8.11.1	istream_iterator	145
8.11.2	ostream_iterator	147
8.12	Aufgaben	148
9	Algorithmen	151
9.1	Übersicht	153
9.2	Nichtmodifizierende Algorithmen	157
9.2.1	for_each	157
9.2.2	find und find_if	158
9.2.3	find_end	160
9.2.4	find_first_of	161
9.2.5	adjacent_find	163
9.2.6	count und count_if	164
9.2.7	mismatch	165
9.2.8	equal	167
9.2.9	search	168
9.2.10	search_n	169
9.3	Modifizierende Algorithmen	170
9.3.1	copy	170
9.3.2	copy_backward	172
9.3.3	swap	173
9.3.4	iter_swap	173
9.3.5	swap_ranges	174
9.3.6	transform	175
9.3.7	replace und replace_if	177
9.3.8	replace_copy und replace_copy_if	178
9.3.9	fill und fill_n	179

9.3.10	generate und generate_n	181
9.3.11	remove_copy und remove_copy_if.....	182
9.3.12	remove und remove_if.....	183
9.3.13	unique_copy	185
9.3.14	unique	187
9.3.15	reverse	188
9.3.16	reverse_copy	189
9.3.17	rotate	190
9.3.18	rotate_copy	191
9.3.19	random_shuffle	192
9.3.20	partition und stable_partition	194
9.4	Sortieren und ähnliche Operationen	195
9.4.1	sort	195
9.4.2	stable_sort.....	197
9.4.3	partial_sort.....	198
9.4.4	partial_sort_copy	199
9.4.5	nth_element	200
9.5	Binäre Suchalgorithmen	202
9.5.1	lower_bound	202
9.5.2	upper_bound.....	203
9.5.3	equal_range	204
9.5.4	binary_search.....	206
9.5.5	Schlüsselsuche mit Funktionsobjekt.....	207
9.6	Mischalgorithmen.....	208
9.6.1	merge.....	208
9.6.2	inplace_merge.....	209
9.7	Mengenalgorithmen für sortierte Bereiche	210
9.7.1	includes.....	211
9.7.2	set_union	212
9.7.3	set_intersection	213
9.7.4	set_difference	215
9.7.5	set_symmetric_difference.....	216
9.7.6	Mengenalgorithmen für multiset-Objekte	217
9.8	Heap-Algorithmen	218
9.8.1	make_heap	219
9.8.2	pop_heap.....	219
9.8.3	push_heap	219
9.8.4	sort_heap	220
9.8.5	Beispielprogramm	220
9.9	Minimum und Maximum	221
9.9.1	min	221
9.9.2	max.....	222
9.9.3	min_element.....	223
9.9.4	max_element	224
9.10	Permutationen	225
9.10.1	lexicographical_compare	225
9.10.2	next_permutation.....	226

9.10.3	prev_permutation	227
9.11	Numerische Algorithmen.....	229
9.11.1	accumulate.....	229
9.11.2	inner_product.....	230
9.11.3	adjacent_difference	231
9.11.4	partial_sum	231
9.12	Erweitern der Bibliothek mit eigenen Algorithmen.....	232
9.13	Präfix- versus Postfixoperatoren.....	234
9.14	Aufgaben.....	235
10	Allokatoren	239
10.1	Der Standardallokator.....	239
10.2	allocator<void>	243
10.3	Aufgaben.....	243
11	Strings	245
11.1	Containereigenschaften	246
11.2	basic_string.....	248
11.3	Implementierungsdetails.....	262
11.4	Sortieren von Strings.....	263
11.5	Aufgaben.....	264
12	Streams	267
12.1	Überblick	267
12.2	ios_base.....	270
12.2.1	Formatierung.....	272
12.2.2	Streamstatus.....	277
12.2.3	Initialisierung	278
12.2.4	Nationale Einstellungen.....	278
12.2.5	Synchronisation	280
12.3	basic_ios.....	280
12.3.1	Statusfunktionen	282
12.3.2	Ausnahmen	283
12.4	basic_ostream.....	284
12.4.1	sentry	285
12.4.2	Formatierte Ausgaben.....	287
12.4.3	Unformatierte Ausgaben.....	288
12.5	basic_istream.....	288
12.5.1	sentry	290
12.5.2	Formatierte Eingaben	290
12.5.3	Unformatierte Eingaben	291
12.6	basic_iostream.....	294
12.7	Ein- und Ausgabe von Objekten benutzerdefinierter Klassen	295
12.8	Namensdeklarationen.....	296
12.9	Manipulatoren.....	297
12.9.1	Manipulatoren ohne Parameter.....	297
12.9.2	Manipulatoren mit einem Parameter.....	300
12.10	Positionieren von Streams.....	301

12.11	Streams für Dateien	303
12.11.1	Modi zum Öffnen von Dateien	303
12.11.2	Die Header-Datei <fstream>	304
12.12	Streams für Strings	307
12.13	Aufgaben	309
13	Weitere Komponenten der C++-Standardbibliothek	311
13.1	auto_ptr.....	311
13.2	bitset.....	317
13.3	vector<bool>	323
13.4	complex	325
13.5	numeric_limits.....	331
13.6	valarray.....	337
13.6.1	slice und slice_array	343
13.6.2	gslice und gslice_array.....	346
13.6.3	mask_array	348
13.6.4	indirect_array.....	350
13.7	Aufgaben	351
14	Zeiger in Containern verwalten	353
14.1	Beispielklassen	353
14.2	Ein set-Objekt verwaltet Zeiger	356
14.3	Smart-Pointer	357
14.4	Ein set-Objekt verwaltet Smart-Pointer.....	358
14.5	Ein set-Objekt verwaltet Zeiger mittels Funktionsobjekten.....	359
14.6	Ein map-Objekt verwaltet Zeiger.....	361
14.7	Aufgaben	363
15	Lösungen	365
Anhang		399
A	Die Containerklasse list	399
B	Die Klasse LogAllocator	413
C	Literatur.....	415
Index		416

Index

Falls sich von mehreren Seitenzahlen eine durch **Fettschrift** hervorhebt, ist dort eine ausführliche Erklärung zu finden. Einträge, die *kursiv* gesetzt sind, gehören nicht zur Standardbibliothek, sondern sind Beispiele.

☞ vii, **30**

- A -

abs 329
Abstand 22
Abstandstyp 123
accumulate 229, 385
Adapter 38
 für Container 59, **93**
 für Funktionszeiger 43
 für Iteratoren 137, 140
 für Zeiger auf Elementfunktionen 45
address 240
adjacent_difference 231
adjacent_find 163
adjustfield 274
advance 135
Aktie 353
Algorithmen 14, **151**
allocate 241
allocator 239
allocator_type 62
allocator<void> 243
Allokatoren 239
Anleihe 353
ANSI v
any 320
app 303
append 252
apply 341
äquivalent 3
arg 329
argument_type 34
assert 411
assign
 basic_string 251
 vector, deque und list 76
assignable 2
assoziative Container 59, **101**
at
 basic_string **261**, 264
 vector und deque **74**, 92
ate 303
Aufgaben vi
Aufwand 5
Aufzählung 301
Ausgabe
 benutzerdefinierter Klassenobjekte 295
 formatiert 287
 unformatiert 288
Ausgaben

 umlenken 269
Ausgabeoperator 147
 basic_string 261
 bitset 322
 complex 328
 list 89
 map 376
 pair 372
 überladen 296
 vector 67
 virtuell 390
Ausnahmeklassen für Streams 283
auto_ptr 311
 Container 314
auto_ptr_ref 314
Auto-Pointer 312

- B -

back 74
 queue 95
back_insert_iterator 141
back_inserter 141
bad 283
badbit 277
basefield 274
basic_fstream 268, **305**
basic_ifstream 268, **305**
basic_ios 267, **280**
basic_iostream 267, **294**
basic_istream 267, **288**
basic_istringstream 268, **308**
basic_ofstream 268, **304**
basic_ostream 267, **284**
basic_ostringstream 268, **307**
basic_string 246, **248**
basic_stringstream 268, **308**
beg 301
begin 64
Benutzerdefinierte Klassenobjekte
 ein- und ausgeben 295
Bereiche 11, 125, 152
bidirectional_iterator_tag 132
Bidirectional-Iteratoren 129
Binärdarstellung 333
binary 303
binary_function 34
binary_negate 42
binary_search 206
BinaryOperation 151
BinaryPredicate 151
BinBaum 31, 120, 368

bind1st 39
 bind2nd 40
 binden 19, 38
 binder1st 38
 binder2nd 39
 Bitmaskentyp 272
 bitset 317
 boolalpha 272, 297, 299
 Boost vi, 358
 bounded_vector 374
 Bruch 335
 Buffer 269

- C -

C 279
 c_str 259
 C++-Standard v
 capacity 77
 cerr 269
 char_traits 246, 267
 char_type 280, 294
 cin 269
 clear **282**, 283
 assoziative Container 108
 basic_string 254
 sequenzielle Container 73
 clog 269
 close 306
 Codeduplizierung 370, 375, 396
 compare
 basic_string 256
 Compare 101, 151
 Compiler vi
 complex 325
 compose 52
 conj 328
 const_iterator 63
 const_mem_fun_ref_t 47
 const_mem_fun_t 49
 const_mem_fun1_ref_t 48
 const_mem_fun1_t 50
 const_pointer 62, 239
 const_reference 62, 239
 const_reverse_iterator 68
 const-Korrektheit 14
 construct 241
 Container 14, **59**
 assoziative 59, **101**
 auto_ptr 314
 reversible 68
 sequenzielle 59, **68**
 Zeiger 353
 container_type 94
 Containeradapter 59, **93**
 copy 170
 basic_string 260
 copy_backward 172
 copy_if 383
 copy-constructible 2
 copyfmt 281, 282
 Copy-Konstruktor 4
 allocator 240
 Container 63
 valarray 338

Copy-on-write 262
 count 164
 assoziative Container 109
 bitset 320
 count_if 164
 cout 269
 cshift 341
 C-Strings 245
 cur 302
 Cursor 12

- D -

data 260
 Dateien 303
 Größe 391
 in String einlesen 389
 Inhalt anzeigen 388
 kopieren 391
 Dateioffnungsmodus 303
 de_DE 279
 deallocate 241
 dec 272, 299
 deque 79
 destroy 242
 Destruktor 4
 allocator 240
 Container 64
 valarray 338
 virtuell 371
 Dezimalkomma 279
 difference_type 63, 132, 240
 Differenzmenge 215
 digits 333
 digits10 333
 Diskettensymbol vii, **30**
 distance 136, 377
 divides 35, 37
 drucken 388

- E -

Einfügen
 Containerelemente 70
 Eingabe
 benutzerdefinierter Klassenobjekte 295
 formatiert 290
 unformatiert 291
 Eingabeoperator 145
 basic_string 261
 bitset 321
 complex 328
 Einweg-Algorithmen 126
 Elementtyp 123
 E-Mail vii
 empty 65
 end 64, 302
 endl 300
 ends 300
 eof 283
 eofbit 277
 epsilon 333
 equal 167
 equal_range 204
 assoziative Container 109

equal_to 35, 37
 equality-comparable 3
 erase 72
 assoziative Container 107
 basic_string 254
 remove 185
 Ergebnistyp 34
 erreichbar 125
 Ersetzbarkeit 27
 event 271
 event_callback 271
 exception 284
 exceptions 284
 explicit 69, 326

- F -

fail 283
 failbit 277
 failure 284
 Fakultät 228
 Fehler vii
 Fehlerquellen 25
 Feld 7
 Feldbreite 274
 ff 388
 fill 179, 282
 fill_n 179
 find 158
 assoziative Container 108
 basic_string 257
 find_end 160, 233
 find_end_if 382
 find_first_not_of 162
 basic_string 259
 find_first_of 161
 basic_string 258
 find_if 158
 find_last_not_of
 basic_string 259
 find_last_of 383
 basic_string 258
 first 56
 first_argument_type 34
 first_type 56
 fixed 272, 274, 299
 flags 273
 Flags 272
 flip
 bitset 319
 vector<bool> 323
 float_denorm_style 331
 float_round_style 331
 floatfield 274
 flush 288, 300
 fmtflags 272
 for_each 157
 formatflags 273
 Formatierte Ausgaben 287
 Formatierte Eingaben 290
 Formatierung 272
 Formfeed 388
 forward_iterator_tag 132
 Forward-Iteratoren 127
 front 73

queue 95
 front_insert_iterator 142
 front_inserter 142
 fstream 268, 305
 Function 151
 Funktionsobjekte
 einstellige 33
 gleich 35
 größer 34, 35
 kleiner 35
 ungleich 35
 zweistellige 33
 Funktionsobjekte 17, 33
 Funktionsobjekte
 Bereich 34
 Funktionsobjekte
 IstPrimzahl 43
 Funktionsobjekte
 unary_compose 51
 Funktionsobjekte
 Rest 52
 Funktionsobjekte
 Quersumme 159
 Funktionsobjekte
 Zaehler 181
 Funktionsobjekte
 Zufall 193
 Funktionsobjekte
 ZeigerKleiner 359
 Funktionsobjekte
 ZeigerDeref 359
 Funktionsobjekte
 WKN 359
 Funktionsobjekte
 WPBez 359
 Funktionsobjekte
 vs. Funktionszeiger 367
 Funktionsobjekte
 Schaltjahr 371

- G -

gcount 291
 generate 181
 generate_n 181
 Generator 151
 Germany_Germany 279
 get 292
 get_allocator 64
 getline 262, 292
 getloc 279
 good 283
 goodbit 277
 greater 34, 35, 37
 greater_equal 35, 37
 Größe 65
 gslice 346
 gslice_array 346

- H -

Handle/Body 315
 Header-Dateien 2
 Heap 218
 hex 272, 299

Homepage vii

- I -

IEC v
 ifstream 268, 305
 ignore 294
 Illegal structure operation 26
 imag 329
 imbue 279, 281
 Implementierungsdetails v, 37
 in 303
 includes 211
 Indexoperator
 basic_string **261**, 264
 bitset 319
 map 109
 valarray 339
 vector und deque **74**, 92
 indirect_array 350
 init 281
 Init 278
 Initialisierung von Streamobjekten 278
 Inkrement 177
 inner_product 230
 inplace_merge 209
 input_iterator_tag 132
 Input-Iteratoren 125
 insert
 basic_string 253
 map und set 105
 multimap und multiset 106
 sequenzielle Container 70
 insert_iterator 143
 inserter 144
 Insert-Iteratoren 140
 assoziative Container 378
 int_type 280, 294
 internal 272, 299
 ios 268
 ios_base **270**
 iostate 277
 ostream 268, 295
 is_bounded 334
 is_exact 333
 is_integer 333
 is_modulo 334
 is_open 305
 is_signed 333
 is_specialized 332
 ISO v
 istream 268, 289
 istream_iterator 145
 istreambuf_iterator 145, 390
 istringstream 268, 308
 istrstream 268
 iter_swap 173
 iterator 63
 Basisklasse 134
 iterator_category 132
 iterator_traits 132
 iterator_traits<const T*> 133
 iterator_traits<T*> 133
 Iteratoradapter 137, 140
 Iteratoren 10, 14, **123**

Iteratorkategorien 124
 iword 271

- K -

kanonische Klassen 4
 Kapazität 77
 Kategorien 124
 key_comp 105
 key_compare 104
 key_type 104
 Klassen
 kanonische 4
 komplexe Zahlen 325
 Komplexität 5
 konstanter Aufwand 5
 Konstruktoren
 für sequenzielle Container 69
 Konvertieren
 Zahlen und Strings 309

- L -

left 272, 299, 387
 length 251
 less 35, 37
 less_equal 35, 37
 less-than-comparable 3
 lexicographical_compare 225
 lexikographisch 225
 linearer Aufwand 5
 list 84
 Liste 8
 locale 279
 log
 Aufwand 5
 LogAllocator **413**
 logarithmischer Aufwand 5
 logical_and 35, 38
 logical_not 35, 38
 logical_or 35, 38
 Löschen
 Containerelemente 72, 185
 Lösungen vii, 365
 lower 388
 lower_bound 202
 assoziative Container 109
 ltrim 388

- M -

main
 return 0; 1
 make_heap 219
 make_pair 57
 Manipulatoren 297
 mit einem Parameter 300
 ohne Parameter 297
 map 101, 102
 mapped_type 109
 mask_array 348
 max 222
 numeric_limits 332
 valarray 341
 max_element 224

max_exponent 334
 max_exponent10 334
 max_size 65
 allocator 241
 mem_fun 50, 51
 mem_fun_ref 47, 49
 mem_fun_ref_t 47
 mem_fun_t 49
 mem_fun1_ref_t 48
 mem_fun1_t 50
 Member-Template 70
 Mengen 210
 merge 208
 list 89
 min 221
 numeric_limits 332
 valarray 341
 min_element 223
 min_exponent 334
 min_exponent10 334
 minus 35, 36, 37
 mismatch 165
 modulus 35, 37
 multimap 101, 110
 multiplies 35, 37
 multiset 101, 116
 Elemente modifizieren 118

- N -

Nachkommastellen 274
 name 279
 Namensbereiche 1
 Namensdeklarationen 264, 296
 namespace std 1
 narrow 281
 NDEBUG 411
 negate 35, 37
 Negativierer 41
 next_permutation 226
 noboolalpha 299
 none 320
 norm 329
 noshowbase 299
 noshowpoint 299
 noshowpos 299
 noskipws 299
 not_equal_to 35, 37
 not1 42
 not2 42
 nunitbuf 299
 nouppercase 299
 npos 250
 nth_element 200
 numeric_limits 331

- O -

O(1) 5
 O(log(n)) 5
 O(n log(n)) 5
 O(n) 5
 O(n²) 5
 oct 272, 299
 off_type 280, 294

ofstream 268, 305
 O-Notation 5
 open 305
 openmode 303
 operator void* 283
 operator! 283
 operator!= 55
 bitset 320
 Container 66
 operator&
 bitset 321
 operator&=
 bitset 321
 operator[]
 basic_string **261**, 264
 bitset 319
 map 109
 valarray 339
 vector und deque **74**, 92
 operator^
 bitset 321
 operator^=
 bitset 321
 operator|
 bitset 321
 operator|=
 bitset 321
 operator~
 bitset 319
 operator+
 basic_string 253
 operator+=
 basic_string 252
 operator<
 Container 66
 operator<< 147, 287
 basic_string 261
 bitset 321, 322
 complex 328
 list 89
 map 376
 pair 372
 überladen 296
 vector 67
 operator<<=
 bitset 321
 operator<= 55
 Container 66
 operator= 4
 basic_string 251
 Container 64
 valarray 338
 operator==
 basic_string 256
 bitset 320
 Container 66
 operator> 55
 Container 66
 operator>= 55
 Container 66
 operator>> 145, 290
 basic_string 261
 bitset 321
 complex 328
 operator>>=

bitset 321
 Ordnungsrelation 3
 ostream 268, 285
 ostream_iterator 147
 ostreambuf_iterator 145
 ostringstream 268, 307
 ostrstream 268
 out 304
 output_iterator_tag 132
 Output-Iteratoren 126

- P -

pair 56
 partial specialization 25
 partial_sort 198
 partial_sort_copy 199
 partial_sum 231
 partition 194
 Past-the-end-value 125
 peek 294
 Permutationen 225
 Placement-new 241
 plus 35, 37
 pointer 62, 132, 239
 pointer_to_binary_function 44
 pointer_to_unary_function 44
 polar 329
 pop 94, 98
 priority_queue 97
 queue 95
 pop_back 74
 pop_front 75
 pop_heap 219
 pos_type 280, 294
 Positionieren von Streams 301
 pow 45
 complex 329
 Prädikate 36
 precision 274
 Predicate 152
 prev_permutation 227
 Primzahlen 43
 printf 280
 priority_queue 96
 Programme vii
 Programmfragmente vii
 Projektionen 38
 ptr_fun 44, 45
 ptrdiff_t 240
 Puffer 269
 push 94, 98
 priority_queue 97
 queue 95
 push_back 74
 push_front 75
 push_heap 219
 put 288
 putback 294
 pword 271

- Q -

quadratische Gleichungen 330
 queue 95

- R -

radix 333
 rand 182
 random_access_iterator_tag 132
 random_shuffle 192
 Random-Access-Iteratoren 129
 RandomNumberGenerator 152
 rationale Zahlen 335
 rbegin 68
 rdbuf 281, 305, 307, 391
 rdstate 282
 read 293
 readsome 293
 real 329
 reallocation 70
 rebind 242
 reference 62, 132, 239
 bitset 319
 vector<bool> 324
 Referenzenzählen 262
 register_callback 271
 rel_ops 55
 relationale Operatoren
 Container 66
 remove 183
 erase 185
 list 88
 remove_copy 182
 remove_copy_if 182
 remove_if 183, 382
 list 88
 rend 68
 replace 177
 basic_string 255
 replace_copy 178
 replace_copy_if 178
 replace_if 177
 replicate 301, 387
 reserve 78, 386
 basic_string 250
 reset
 bitset 318
 resetiosflags 300
 resize 77
 valarray 341
 result_type
 unary_function 34
 reverse 188
 list 89
 reverse_copy 189
 reverse_iterator 68, 137
 Reverse-Iteratoren 137
 reversible Container 68
 rfind
 basic_string 257
 right 272, 299, 387
 rotate 190
 rotate_copy 191
 round_error 334

- S -

scanf 280
 Schnittmenge 213

- scientific 272, 274, 299
- search 168
- search_n 169
- search_n_if 170
- second 56
- second_argument_type 34
- second_type 56
- seekdir 301
- seekg 302
- seekp 302
- sentry
 - basic_istream 290
 - basic_ostream 285
- sequenzielle Container 59, **68**
- set 101, 113
 - bitset 318
 - Elemente modifizieren 118
- set_difference 215
- set_intersection 213
- set_symmetric_difference 216
- set_union 212
- setbase 300
- setf 273
- setfill 300
- setiosflags 300
- setprecision 300
- setstate 283
- setw 300
- shared_ptr 358
- shift 341
- showbase 272, 299
- showpoint 272, 299
- showpos 272, 299
- shrink-to-fit 77, 250
- size 65
 - basic_string 251
 - bitset 320
 - gslice 347
 - slice 344
 - valarray 340
 - vector 65
- Size 152
- size_t 240
- size_type 63, 240
- Skalarprodukt 230, 352
- skipws 272, 276, 299
- slice 343
- slice_array 343
- slist 92, 99, 134, 149, 242, 243, **399**
- smart_ptr 357
- Smart-Pointer 357
- sort 195
 - list 89
- sort_heap 220
- sortieren
 - deutsch 263
 - Strings 263
- Sourcecode vii
- splice 87
- srand 182
- stable_partition 194
- stable_sort 197
- stack 93
- Standardallokator 239
- Standardkonstruktor 4
 - allocator 240
 - assoziative Container 105
 - bitset 317
 - Container 63
 - valarray 338
- Statusfunktionen 282
- std 1
- std::rel_ops 55
- STL v
- STLport vi
- str 307
- Streamende-Iteratoren 145
- Stream-Iteratoren 144
- streamoff 270
- Streams 267
 - Ausnahmeklassen 283
 - für Dateien 303
 - für Strings 307
 - positionieren 301
 - Status 277
 - Übersicht 269
 - Vererbungshierarchie 268
- streamsize 270
- string 246
- Strings
 - sortieren 263
- Strings 245
- Strings
 - Datei einlesen 389
- stringstream 268, 308
- Stringstreams 268, 307
- stringstream 268
- stuff 388
- Substitution 27
- substr 260
- Suchalgorithmen 14
- Suchfunktionen 14
- sum 341
- swap
 - Algorithmus 173
 - Elementfunktion 65
 - map 103
 - spezielle Version 79
- swap_ranges 174
- sync 289
- sync_with_stdio 280

- T -

- T 152
- tauschen 65
- Tausenderpunkte 279
- tellg 302
- tellp 302
- template<> 229
- test
 - bitset 320
- tie 281
- time 182
- times 35
- to_ulong 320
- top 94
 - priority_queue 97
- toupper 247
- traits_type 250, 280, 294

transform 175
 trim 388
 trunc 304
 typedef 104
 typename 24

- U -

Übersicht

- Adapter für Zeiger auf Elementfunktionen 47
- Algorithmen 153
- Container 120
- Containeriteratoren 131
- Funktionsobjekte 35
- Insert-Iteratoren 140
- Iteratorkategorien 124, 131
- Manipulatoren mit einem Parameter 300
- Manipulatoren ohne Parameter 299
- optionale Sequenzanforderungen 76
- sequenzielle Container 91
- Streams 269
- umlenken
 - von Ausgaben 269
- unary_compose 51
- unary_function 34
- unary_negate 41
- UnaryOperation 152
- uncaught_exception 287
- Unformatierte Ausgaben 288
- Unformatierte Eingaben 291
- unget 294
- unique 187
 - list 88
- unique_copy 185
- unitbuf 272, 299
- unsetf 273
- upper 388
- upper_bound 203
 - assoziative Container 109
- uppercase 272, 299
- using namespace std; 1

- V -

valarray 337
 value_comp 105

value_compare 104
 value_type 62, 132, 239

- map 103
- set und multiset 118

vector 60
 vector<bool> 323
 Vereinigungsmenge 212
 Vergleichsobjekte 101
 Vergleichsoperatoren 4, 55

- Container 66

vertauschen 65
 virtueller Destruktor 371

- W -

wchar_t 246
 Wertpapier 353
 what 284
 White-space 261
 widen 281
 width 274, 291
 Wrapper-Klasse 396
 write 288
 ws 300
 WWW vii

- X -

xalloc 271

- Z -

Zahlen

- einlesen 290

Zeichen

- einlesen 291

Zeichenketten 245

- einlesen 291

Zeiger 123, 133
 Zeiger und Container 353
 Zuweisungsoperator 4

- basic_string 251
- Container 64
- valarray 338

Buchrückentext

Die C++-Standardbibliothek, deren interessantester Teil die Standard Template Library (STL) ist, revolutioniert die C++-Programmierung. Der korrekte und effiziente Einsatz stellt eine große Herausforderung dar, die nur mit fundiertem Wissen zu meistern ist. In diesem Buch wird die Funktionsweise und Implementierung der einzelnen Komponenten (Container, Iteratoren, Algorithmen, Funktionsobjekte, Strings, Streams usw.) verständlich und detailliert erklärt, wobei typische Stolpersteine aufgezeigt werden. Zahlreiche, mit gängigen C++-Compilern getestete Programmbeispiele demonstrieren den praktischen Einsatz. Antworten auf Standardfragen sowie unzählige Tipps und Tricks machen das Buch zu einem nützlichen Ratgeber für den alltäglichen Gebrauch. Anhand von anspruchsvollen Aufgaben und deren Lösungen kann das erworbene Wissen überprüft und vertieft werden. Der komplette Programmcode ist im Internet verfügbar. Ein ausführlicher Index ermöglicht gezieltes Nachschlagen.