

# Objekttechnologie

---

*Reihenherausgeber*

Martin Schader, Mannheim

Springer-Verlag Berlin Heidelberg GmbH

# Objekttechnologie

---

Martin Schader und Michael Rundshagen  
Objektorientierte Systemanalyse  
2. Auflage  
IX, 241 Seiten. 1996

Martin Schader  
Objektorientierte Datenbanken  
Die C++-Anbindung des  
ODMG-Standards  
X, 219 Seiten. 1997

Günther Vinek  
Objektorientierte Softwareentwicklung  
mit Smalltalk  
XII, 440 Seiten. 1997

Martin Schader und Stefan Kuhllins  
Programmieren in C++  
Einführung in den Sprachstandard  
5. Auflage  
XII, 386 Seiten. 1998

Martin Schader und Lars Schmidt-Thieme  
Java  
Einführung in die objektorientierte  
Programmierung  
XVII, 544 Seiten. 1998

Stefan Kuhlins  
Martin Schader

# Die C++-Standard- bibliothek

Einführung und Nachschlagewerk

Mit 77 Abbildungen  
und 37 Tabellen



Springer

Dr. Stefan Kuhlins  
Prof. Dr. Martin Schader  
Universität Mannheim  
Lehrstuhl für Wirtschaftsinformatik III  
Schloß  
D-68131 Mannheim

ISBN 978-3-540-65052-2

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Kuhlins, Stefan: Die C++-Standardbibliothek: Einführung und Nachschlagewerk /  
Stefan Kuhlins; Martin Schader.

(Objekttechnologie)

ISBN 978-3-540-65052-2 ISBN 978-3-662-06640-9 (eBook)

DOI 10.1007/978-3-662-06640-9

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funksendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Vervielfältigung dieses Werkes oder von Teilen dieses Werkes ist auch im Einzelfall nur in den Grenzen der gesetzlichen Bestimmungen des Urheberrechtsgesetzes der Bundesrepublik Deutschland vom 9. September 1965 in der jeweils geltenden Fassung zulässig. Sie ist grundsätzlich vergütungspflichtig. Zuwiderhandlungen unterliegen den Strafbestimmungen des Urheberrechtsgesetzes.

© Springer-Verlag Berlin Heidelberg 1999

Ursprünglich erschienen bei Springer-Verlag Berlin Heidelberg New York 1999

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, daß solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Umschlaggestaltung: Struwe & Partner, Heidelberg

SPIN 10694940

43/2202-5 4 3 2 1 0 – Gedruckt auf säurefreiem Papier

# 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 Implementationen aber i.d.R. 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 gewinnbringender 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. In diesem Zusammenhang ist vor allem Kapitel 2 hervorzuheben, in dem wir einen möglichen Entwicklungsweg für die STL aufzeigen.

Bei der Beschreibung der einzelnen Komponenten der STL – Container, Iteratoren, Algorithmen, Funktionsobjekte usw. – gehen wir auch auf Implementationsdetails ein, die vom Standard zwar nicht festgeschrieben sind, sich i.d.R. aber aufgrund von Vorgaben für die Komplexität automatisch ergeben. Das Wissen um Implementationsdetails 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 u.a. *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 Entwurf für den C++-Standard. Die Arbeiten am C++-Standard, die vom *American National Standards Institute* (ANSI J16) und von der *International Standards Organization* (ISO WG21) vorangetrieben werden, sind gegenwärtig quasi abgeschlossen. Der aktuelle Stand wird durch den *Final Draft International Standard* (FDIS) vom November 1997 repräsentiert. Laut Zeitplan soll der endgültige Standard unter der Bezeichnung ISO/IEC 14882, *Standard for the C++ Programming Language*,

im Oktober 1998 erscheinen. Er soll sich vom FDIS lediglich durch die Korrektur von typographischen und Layoutfehlern unterscheiden.

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 erschienen ist, sind während der Standardisierung unzählige Änderungen vorgenommen worden. An Stellen, an denen es uns nützlich erschien, weisen wir auf frühere Besonderheiten hin, damit sich unsere Programmbeispiele an ältere Versionen der STL anpassen lassen. Es ist allerdings zu erwarten, daß die Compiler- und Bibliothekshersteller innerhalb kürzester Zeit den aktuellen C++-Standard umsetzen werden.

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 derzeit verfügbaren C++-Compiler eingesetzt (in alphabetischer Reihenfolge):


- Borland C++ unter Windows 95/NT
- EGCS unter Linux und Solaris
- GNU g++ unter Linux und Solaris
- Kuck & Associates, Inc. KAI C++ unter Linux und Solaris
- Microsoft Visual C++ unter Windows 95/NT

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 Implementationen kann man damit auf die „Sprünge“ helfen, so daß man seine Programme gemäß dem aktuellen Standard schreiben kann und späterer Umstellungsaufwand entfällt.

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 65). 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 langweilende 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.

Der Sourcecode aller mit dem Diskettensymbol  gekennzeichneten Programme und Lösungen ist mittels *Anonymous ftp* auf dem Server `ftp.wifo.uni-mannheim.de` im Unterverzeichnis `/pub/buecher/stlbuch` zugänglich. Alternativ ist er gegen Einsendung einer MS-DOS-formatierten 3½"-Diskette mit ausreichend frankiertem Rückumschlag an die auf den Umschlagseiten angegebene Anschrift erhältlich.

Über Anregungen unserer Leserinnen und Leser an unsere Postanschrift oder als *Email* 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 unter <http://www.wifo.uni-mannheim.de/stlbuch>, wo wir auch aktuelle Informationen zur C++-Standardbibliothek und Hinweise zu speziellen Compilern zur Verfügung stellen.

Wir danken Michael Gröschel und Rainer Lienhart für die Durchsicht des Manuskripts. Außerdem gilt unser Dank dem Springer-Verlag für die, wie immer, sehr gute Zusammenarbeit.

Mannheim, im August 1998

*Stefan Kuhlins, Martin Schader*

# Inhaltsverzeichnis

<b>1</b>	<b>Vorbemerkungen</b>	<b>1</b>
1.1	namespace std.....	1
1.2	Header-Dateien .....	2
1.3	Kanonische Klassen .....	2
1.4	Eigenschaften .....	3
1.5	Komplexität und Aufwand.....	5
<b>2</b>	<b>Konzeption und Entwicklung der STL</b>	<b>7</b>
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.....	21
2.14	Fehlerquellen.....	24
2.15	Ist die STL objektorientiert?.....	25
2.16	Einsatz der Standardbibliothek.....	28
2.17	Aufgaben .....	29
<b>3</b>	<b>Funktionsobjekte</b>	<b>31</b>
3.1	Basisklassen für Funktionsobjekte .....	31
3.2	Arithmetische, logische und Vergleichsoperationen.....	33
3.3	Projektionen.....	36
3.3.1	binder1st.....	36
3.3.2	bind1st.....	36
3.3.3	binder2nd.....	37
3.3.4	bind2nd.....	37
3.3.5	Beispiel .....	37
3.4	Negativierer .....	38
3.4.1	unary_negate.....	38
3.4.2	not1.....	39
3.4.3	binary_negate.....	39
3.4.4	not2.....	39
3.5	Adapter für Funktionszeiger .....	40
3.5.1	pointer_to_unary_function .....	41



3.5.2	ptr_fun für einstellige Funktionen .....	41
3.5.3	pointer_to_binary_function .....	41
3.5.4	ptr_fun für zweistellige Funktionen.....	42
3.6	Adapter für Zeiger auf Elementfunktionen .....	42
3.6.1	mem_fun_ref_t und const_mem_fun_ref_t.....	44
3.6.2	mem_fun_ref für Elementfunktionen ohne Argument .....	44
3.6.3	mem_fun1_ref_t und const_mem_fun1_ref_t .....	45
3.6.4	mem_fun_ref für Elementfunktionen mit Argument .....	46
3.6.5	mem_fun_t und const_mem_fun_t.....	46
3.6.6	mem_fun für Elementfunktionen ohne Argument.....	47
3.6.7	mem_fun1_t und const_mem_fun1_t.....	47
3.6.8	mem_fun für Elementfunktionen mit Argument .....	48
3.7	Funktionen zusammensetzen.....	48
3.8	Aufgaben .....	49
<b>4</b>	<b>Hilfsmittel</b> .....	<b>51</b>
4.1	Vergleichsoperatoren .....	51
4.2	pair.....	52
4.3	Aufgaben .....	53
<b>5</b>	<b>Container</b> .....	<b>55</b>
5.1	vector .....	56
5.2	Allgemeine Anforderungen an Container .....	57
5.3	Anforderungen an reversible Container .....	63
5.4	Anforderungen an sequentielle Container.....	63
5.5	Optionale Anforderungen an sequentielle Container .....	68
5.6	Weitere Funktionen .....	71
5.7	deque.....	74
5.8	list.....	78
5.9	Auswahl nach Aufwand .....	85
5.10	Aufgaben .....	86
<b>6</b>	<b>Containeradapter</b> .....	<b>89</b>
6.1	stack.....	89
6.2	queue.....	91
6.3	priority_queue .....	93
6.4	Aufgaben .....	95
<b>7</b>	<b>Assoziative Container</b> .....	<b>97</b>
7.1	map .....	98
7.2	Anforderungen an assoziative Container .....	100
7.3	Der Indexoperator der Klasse map .....	105
7.4	multimap.....	106
7.5	set.....	110
7.6	multiset .....	112
7.7	Übersicht der Container .....	115
7.8	Aufgaben .....	115

<b>8</b>	<b>Iteratoren</b>	<b>117</b>
8.1	Iteratoranforderungen .....	117
8.2	Input-Iteratoren .....	119
8.3	Output-Iteratoren.....	120
8.4	Forward-Iteratoren .....	121
8.5	Bidirectional-Iteratoren .....	122
8.6	Random-Access-Iteratoren.....	123
8.7	Übersicht über die Iteratorkategorien .....	124
8.8	Hilfsklassen und -funktionen für Iteratoren.....	125
8.8.1	iterator_traits .....	125
8.8.2	Die Basisklasse iterator .....	127
8.8.3	Iteratorfunktionen .....	128
8.8.3.1	advance.....	129
8.8.3.2	distance .....	130
8.9	Reverse-Iteratoren.....	130
8.10	Insert-Iteratoren.....	133
8.10.1	back_insert_iterator .....	134
8.10.2	front_insert_iterator .....	135
8.10.3	insert_iterator .....	136
8.11	Stream-Iteratoren .....	137
8.11.1	istream_iterator .....	138
8.11.2	ostream_iterator .....	140
8.12	Aufgaben .....	141
<b>9</b>	<b>Algorithmen</b>	<b>143</b>
9.1	Übersicht.....	145
9.2	Nichtmodifizierende Algorithmen .....	149
9.2.1	for_each .....	149
9.2.2	find und find_if .....	150
9.2.3	find_end.....	152
9.2.4	find_first_of .....	153
9.2.5	adjacent_find .....	154
9.2.6	count und count_if.....	156
9.2.7	mismatch.....	157
9.2.8	equal.....	158
9.2.9	search .....	159
9.2.10	search_n.....	160
9.3	Modifizierende Algorithmen .....	162
9.3.1	copy.....	162
9.3.2	copy_backward.....	163
9.3.3	swap .....	164
9.3.4	iter_swap .....	165
9.3.5	swap_ranges .....	166
9.3.6	transform .....	167
9.3.7	replace und replace_if .....	168
9.3.8	replace_copy und replace_copy_if .....	169
9.3.9	fill und fill_n .....	170

9.3.10	generate und generate_n.....	172
9.3.11	remove_copy und remove_copy_if.....	173
9.3.12	remove und remove_if.....	174
9.3.13	unique_copy.....	175
9.3.14	unique.....	177
9.3.15	reverse.....	178
9.3.16	reverse_copy.....	179
9.3.17	rotate.....	180
9.3.18	rotate_copy.....	180
9.3.19	random_shuffle.....	181
9.3.20	partition und stable_partition.....	183
9.4	Sortieren und ähnliche Operationen.....	184
9.4.1	sort.....	184
9.4.2	stable_sort.....	186
9.4.3	partial_sort.....	187
9.4.4	partial_sort_copy.....	188
9.4.5	nth_element.....	189
9.5	Binäre Suchalgorithmen.....	190
9.5.1	lower_bound.....	191
9.5.2	upper_bound.....	192
9.5.3	equal_range.....	193
9.5.4	binary_search.....	194
9.6	Mischalgorithmen.....	195
9.6.1	merge.....	195
9.6.2	inplace_merge.....	197
9.7	Mengenalgorithmen für sortierte Bereiche.....	198
9.7.1	includes.....	198
9.7.2	set_union.....	199
9.7.3	set_intersection.....	201
9.7.4	set_difference.....	202
9.7.5	set_symmetric_difference.....	203
9.7.6	Mengenalgorithmen für multiset-Objekte.....	204
9.8	Heap-Algorithmen.....	205
9.8.1	make_heap.....	206
9.8.2	pop_heap.....	206
9.8.3	push_heap.....	206
9.8.4	sort_heap.....	207
9.8.5	Beispielprogramm.....	207
9.9	Minimum und Maximum.....	209
9.9.1	min.....	209
9.9.2	max.....	209
9.9.3	min_element.....	210
9.9.4	max_element.....	211
9.10	Permutationen.....	212
9.10.1	lexicographical_compare.....	212
9.10.2	next_permutation.....	214
9.10.3	prev_permutation.....	214

---

9.11	Numerische Algorithmen .....	216
9.11.1	accumulate .....	216
9.11.2	inner_product .....	217
9.11.3	adjacent_difference .....	218
9.11.4	partial_sum .....	218
9.12	Erweitern der Bibliothek mit eigenen Algorithmen .....	219
9.13	Präfix- versus Postfixoperatoren .....	221
9.14	Aufgaben .....	222
<b>10</b>	<b>Allokatoren</b> .....	<b>225</b>
10.1	Der Standardallokator .....	225
10.2	allocator<void> .....	228
10.3	Aufgaben .....	228
<b>11</b>	<b>Strings</b> .....	<b>229</b>
11.1	Containereigenschaften .....	230
11.2	basic_string .....	232
11.3	Implementationsdetails .....	245
11.4	Aufgaben .....	246
<b>12</b>	<b>Streams</b> .....	<b>249</b>
12.1	Überblick .....	249
12.2	ios_base .....	251
12.2.1	Formatierung .....	253
12.2.2	Streamstatus .....	258
12.2.3	Initialisierung .....	259
12.3	basic_ios .....	260
12.3.1	Statusfunktionen .....	262
12.3.2	Ausnahmen .....	263
12.4	basic_ostream .....	263
12.4.1	sentry .....	264
12.4.2	Formatierte Ausgaben .....	266
12.4.3	Unformatierte Ausgaben .....	267
12.5	basic_istream .....	267
12.5.1	sentry .....	268
12.5.2	Formatierte Eingaben .....	269
12.5.3	Unformatierte Eingaben .....	270
12.6	basic_iostream .....	273
12.7	Ein- und Ausgabe von Objekten benutzerdefinierter Klassen .....	273
12.8	Namensdeklarationen .....	274
12.9	Manipulatoren .....	275
12.9.1	Manipulatoren ohne Parameter .....	276
12.9.2	Manipulatoren mit einem Parameter .....	278
12.10	Positionieren von Streams .....	279
12.11	Streams für Dateien .....	280
12.11.1	Modi zum Öffnen von Dateien .....	280
12.11.2	Die Header-Datei <fstream> .....	282
12.12	Streams für Strings .....	284

12.13	Aufgaben .....	287
<b>13</b>	<b>Weitere Komponenten der C++-Standardbibliothek</b>	<b>289</b>
13.1	auto_ptr .....	289
13.2	bitset .....	294
13.3	vector<bool> .....	299
13.4	complex .....	301
13.5	numeric_limits .....	307
13.6	valarray .....	312
13.6.1	slice und slice_array .....	318
13.6.2	gslice und gslice_array .....	321
13.6.3	mask_array .....	324
13.6.4	indirect_array .....	325
13.7	Aufgaben .....	327
<b>14</b>	<b>Zeiger in Containern verwalten</b>	<b>329</b>
14.1	Beispielklassen .....	329
14.2	Ein set-Objekt verwaltet Zeiger .....	332
14.3	Smart-Pointer .....	333
14.4	Ein set-Objekt verwaltet Smart-Pointer .....	334
14.5	Ein set-Objekt verwaltet Zeiger mittels Funktionsobjekten .....	335
14.6	Ein map-Objekt verwaltet Zeiger .....	337
14.7	Aufgaben .....	338
<b>15</b>	<b>Lösungen</b>	<b>339</b>
<b>Anhang</b>		<b>363</b>
A	Die Containerklasse list .....	363
B	Literatur .....	376
<b>Index</b>		<b>377</b>