

Jürgen Wolf

C++C



CD-ROM mit

- ✓ Openbooks
- ✓ Quellcode der Beispiele
- ✓ Entwicklungs-umgebungen

C++C++C++C++C++  
C++C++C++C++C++  
C++C++C++C++C++

C++

von A bis Z

Das umfassende Handbuch

C++C++  
C++C++C++C++C++  
C++C++C++C++C++  
C++C++C++C++C++  
C++C++C++C++C++  
C++C++C++C++

- Das Lehr- und Nachschlagewerk
- Für Einsteiger: ANSI C++ verstehen und anwenden
- Für Profis: UML, Netzwerkprogrammierung, GUI- und Multimedia-Bibliotheken

2., aktualisierte Auflage

Galileo Computing



Lehr- und  
Nachschlagewerk

von A bis Z

C++

Wolf



ISBN



# Auf einen Blick

<b>1</b>	<b>Grundlagen in C++</b> .....	25
<b>2</b>	<b>Höhere und fortgeschrittene Datentypen</b> .....	133
<b>3</b>	<b>Gültigkeitsbereiche, spezielle Deklarationen und Typumwandlungen</b> .....	225
<b>4</b>	<b>Objektorientierte Programmierung</b> .....	265
<b>5</b>	<b>Templates und STL</b> .....	477
<b>6</b>	<b>Exception-Handling</b> .....	661
<b>7</b>	<b>C++-Standardbibliothek</b> .....	695
<b>8</b>	<b>Weiteres zum C++-Guru</b> .....	821
<b>9</b>	<b>Netzwerkprogrammierung und Cross-Plattform-Entwicklung in C++</b> .....	917
<b>10</b>	<b>GUI- und Multimediaprogrammierung in C++</b> .....	993
<b>A</b>	<b>Anhang</b> .....	1207

# Inhalt

Vorwort .....	17
Vorwort des Fachgutachters .....	23
<b>1 Grundlagen in C++ .....</b>	<b>25</b>
1.1 Die Entstehung von C++ .....	25
1.1.1 Aufbau von C++ .....	28
1.2 Erste Schritte der C++-Programmierung .....	31
1.2.1 Ein Programm erzeugen mit einem Kommandozeilen-Compiler .....	32
1.2.2 Ausführen des Programms .....	34
1.2.3 Ein Programm erzeugen mit einer IDE .....	34
1.3 Symbole von C++ .....	35
1.3.1 Bezeichner .....	35
1.3.2 Schlüsselwörter .....	35
1.3.3 Literale .....	36
1.3.4 Einfache Begrenzer .....	37
1.4 Basisdatentypen .....	39
1.4.1 Deklaration und Definition .....	39
1.4.2 Was ist eine Variable? .....	40
1.4.3 Der Datentyp »bool« .....	40
1.4.4 Der Datentyp »char« .....	41
1.4.5 Die Datentypen »int« .....	44
1.4.6 Gleitkommazahlen »float«, »double« und »long double« .....	46
1.4.7 Limits für Ganzzahl- und Gleitpunkt-Datentypen .....	50
1.4.8 Die Größen der Basistypen .....	51
1.4.9 void .....	52
1.5 Konstanten .....	53
1.6 Standard Ein-/Ausgabe-Streams .....	54
1.6.1 Die neuen Streams – »cout«, »cin«, »cerr«, »clog« .....	54
1.6.2 Ausgabe mit »cout« .....	56
1.6.3 Ausgabe mit »cerr« .....	56
1.6.4 Eingabe mit »cin« .....	57
1.7 Operatoren .....	59
1.7.1 Arithmetische Operatoren .....	60
1.7.2 Inkrement- und Dekrementoperator .....	63
1.7.3 Bit-Operatoren .....	64
1.7.4 Weitere Operatoren .....	68

1.8	Kommentare .....	68
1.9	Kontrollstrukturen .....	69
1.9.1	Verzweigungen (Selektionen) .....	69
1.9.2	Schleifen (Iterationen) .....	88
1.9.3	Sprunganweisungen .....	96
1.10	Funktionen .....	99
1.10.1	Deklaration und Definition .....	99
1.10.2	Funktionsaufruf und Parameterübergabe .....	102
1.10.3	Lokale und globale Variablen .....	109
1.10.4	Standardparameter .....	110
1.10.5	Funktionen überladen .....	113
1.10.6	Inline-Funktionen .....	117
1.10.7	Rekursionen .....	120
1.10.8	Die »main«-Funktion .....	121
1.11	Präprozessor-Direktiven .....	122
1.11.1	Die »#define«-Direktive .....	123
1.11.2	Die »#undef«-Direktive .....	126
1.11.3	Die »#include«-Direktive .....	127
1.11.4	Die Direktiven »#error« und »#pragma« .....	128
1.11.5	Bedingte Kompilierung .....	129

## 2 Höhere und fortgeschrittene Datentypen ..... 133

2.1	Zeiger .....	133
2.1.1	Zeiger deklarieren .....	134
2.1.2	Adresse im Zeiger speichern .....	135
2.1.3	Zeiger dereferenzieren .....	137
2.1.4	Zeiger, die auf andere Zeiger verweisen .....	141
2.1.5	Dynamisch Speicherobjekte anlegen und zerstören – »new« und »delete« .....	143
2.1.6	void-Zeiger .....	148
2.1.7	Konstante Zeiger .....	148
2.2	Referenzen .....	149
2.3	Arrays .....	152
2.3.1	Arrays deklarieren .....	152
2.3.2	Arrays initialisieren .....	153
2.3.3	Bereichsüberschreitung von Arrays .....	155
2.3.4	Anzahl der Elemente eines Arrays ermitteln .....	156
2.3.5	Array-Wert von Tastatur einlesen .....	157
2.3.6	Mehrdimensionale Arrays .....	157
2.4	Zeichenketten (C-Strings) – char-Array .....	159
2.4.1	C-String deklarieren und initialisieren .....	160

2.4.2	C-String einlesen .....	161
2.4.3	C-Strings: Bibliotheksfunktionen .....	161
2.5	Arrays und Zeiger .....	166
2.5.1	C-Strings und Zeiger .....	171
2.5.2	Mehrfache Indirektion .....	171
2.5.3	C-String-Tabellen .....	173
2.5.4	Arrays im Heap (dynamisches Array) .....	176
2.6	Parameterübergabe mit Zeigern, Arrays und Referenzen .....	181
2.6.1	Call by value .....	181
2.6.2	Call by reference – Zeiger als Funktionsparameter .....	182
2.6.3	Call by reference mit Referenzen nachbilden .....	184
2.6.4	Arrays als Funktionsparameter .....	185
2.6.5	Mehrdimensionale Arrays an Funktionen übergeben ...	187
2.6.6	Argumente an die main-Funktion übergeben .....	188
2.7	Rückgabewerte von Zeigern, Arrays und Referenzen .....	190
2.7.1	Zeiger als Rückgabewert .....	190
2.7.2	Referenz als Rückgabewert .....	194
2.7.3	const-Zeiger als Rückgabewert .....	195
2.7.4	Array als Rückgabewert .....	195
2.7.5	Mehrere Rückgabewerte .....	196
2.8	Fortgeschrittene Typen .....	197
2.8.1	Strukturen .....	198
2.8.2	Unions .....	218
2.8.3	Aufzählungstypen .....	221
2.8.4	typedef .....	223
<b>3</b>	<b>Gültigkeitsbereiche, spezielle Deklarationen und Typumwandlungen .....</b>	<b>225</b>
3.1	Gültigkeitsbereiche (Scope) .....	225
3.1.1	Lokaler Gültigkeitsbereich (Local Scope) .....	226
3.1.2	Gültigkeitsbereich Funktionen .....	226
3.1.3	Gültigkeitsbereich Namensraum (Namespaces) .....	228
3.1.4	Gültigkeitsbereich Klassen (Class Scope) .....	228
3.2	Namensräume (Namespaces) .....	228
3.2.1	Neuen Namensbereich erzeugen (Definition) .....	228
3.2.2	Zugriff auf die Bezeichner im Namensraum .....	231
3.2.3	using – einzelne Bezeichner aus einem Namensraum importieren .....	233
3.2.4	using – alle Bezeichner aus einem Namensraum importieren .....	236
3.2.5	Namensauflösung .....	239

3.2.6	Alias-Namen für Namensbereiche .....	240
3.2.7	Anonyme (namenlose) Namensbereiche .....	241
3.2.8	Namensbereich und Header-Dateien .....	242
3.3	C-Funktionen bzw. -Bibliotheken in einem C++-Programm .....	244
3.3.1	C-Funktionen aus einer C-Bibliothek aufrufen .....	245
3.4	Speicherklassenattribute .....	249
3.4.1	Speicherklasse »auto« .....	250
3.4.2	Speicherklasse »register« .....	250
3.4.3	Speicherklasse »static« .....	250
3.4.4	Speicherklasse »extern« .....	251
3.4.5	Speicherklasse »mutable« .....	253
3.5	Typqualifikatoren .....	253
3.5.1	Qualifizierer »const« .....	254
3.5.2	Qualifizierer »volatile« .....	254
3.6	Funktionsattribute .....	255
3.7	Typumwandlung .....	255
3.7.1	Standard-Typumwandlung .....	256
3.7.2	Explizite Typumwandlung .....	260
<b>4</b>	<b>Objektorientierte Programmierung .....</b>	<b>265</b>
4.1	OOP-Konzept versus prozedurales Konzept .....	265
4.1.1	OOP-Paradigmen .....	266
4.2	Klassen (fortgeschrittene Typen) .....	267
4.2.1	Klassen deklarieren .....	268
4.2.2	Elementfunktion (Klassenmethode) definieren .....	269
4.2.3	Objekte deklarieren .....	271
4.2.4	Kurze Zusammenfassung .....	271
4.2.5	»private« und »public« – Zugriffsrechte in der Klasse ...	272
4.2.6	Zugriff auf die Elemente (Member) einer Klasse .....	274
4.2.7	Ein Programm organisieren .....	281
4.2.8	Konstruktoren .....	285
4.2.9	Destruktoren .....	293
4.3	Mehr zu den Klassenmethoden (Klassenfunktionen) .....	295
4.3.1	Inline-Methoden (explizit und implizit) .....	296
4.3.2	Zugriffsmethoden .....	299
4.3.3	Read-only-Methoden .....	303
4.3.4	this-Zeiger .....	305
4.4	Verwenden von Objekten .....	307
4.4.1	Read-only-Objekte .....	307
4.4.2	Objekte als Funktionsargumente .....	307
4.4.3	Objekte als Rückgabewert .....	314

4.4.4	Klassen-Array (Array von Objekten) .....	316
4.4.5	Dynamische Objekte .....	318
4.4.6	Dynamische Klassenelemente .....	324
4.4.7	Objekte kopieren (Kopierkonstruktor) .....	328
4.4.8	Dynamisch erzeugte Objekte kopieren (»operator=()«) .....	330
4.4.9	Standardmethoden (Überblick) .....	331
4.4.10	Objekte als Elemente (bzw. Eigenschaften) in anderen Klassen .....	332
4.4.11	Teilobjekte initialisieren .....	338
4.4.12	Klassen in Klassen verschachteln .....	340
4.4.13	Konstante Klasseneigenschaften (Datenelemente) .....	341
4.4.14	Statische Klasseneigenschaften (Datenelemente) .....	343
4.4.15	Statische Klassenmethoden .....	348
4.4.16	Friend-Funktionen bzw. friend-Klassen .....	349
4.4.17	Zeiger auf Eigenschaften einer Klasse .....	352
4.5	Operatoren überladen .....	358
4.5.1	Grundlegendes zur Operator-Überladung .....	358
4.5.2	Überladen von arithmetischen Operatoren .....	362
4.5.3	Überladen von unären Operatoren .....	371
4.5.4	Überladen von ++ und -- .....	374
4.5.5	Überladen des Zuweisungsoperators .....	376
4.5.6	Überladen des Indexoperators »[]« (Arrays überladen) .....	378
4.5.7	Shift-Operatoren überladen .....	381
4.5.8	()-Operator überladen .....	385
4.6	Typumwandlung für Klassen .....	388
4.6.1	Konvertierungskonstruktor .....	388
4.6.2	Konvertierungsfunktion .....	390
4.7	Vererbung (abgeleitete Klassen) .....	392
4.7.1	Anwendungsbeispiel (Vorbereitung) .....	394
4.7.2	Die Ableitung einer Klasse .....	397
4.7.3	Redefinition von Klassenelementen .....	401
4.7.4	Konstruktor .....	404
4.7.5	Destruktoren .....	407
4.7.6	Zugriffsrecht »protected« .....	407
4.7.7	Typumwandlung abgeleiteter Klassen .....	410
4.7.8	Klassenbibliotheken erweitern .....	413
4.8	Polymorphismus .....	414
4.8.1	Statische bzw. dynamische Bindung .....	415
4.8.2	Virtuelle Methoden .....	415

4.8.3	Virtuelle Methoden redefinieren .....	420
4.8.4	Arbeitsweise virtueller Methoden .....	426
4.8.5	Virtuelle Destruktoren bzw. Destruktoren abgeleiteter Klassen .....	431
4.8.6	Polymorphismus und der Zuweisungsoperator .....	433
4.8.7	Rein virtuelle Methoden und abstrakte Basisklassen ....	435
4.8.8	Probleme mit der Vererbung und der dynamic_cast-Operator .....	439
4.8.9	Fallbeispiel: Verkettete Listen .....	441
4.9	Mehrfachvererbung .....	463
4.9.1	Indirekte Basisklassen erben .....	467
4.9.2	Virtuelle indirekte Basisklassen erben .....	471

## 5 Templates und STL ..... 477

5.1	Funktions-Templates .....	477
5.1.1	Funktions-Templates definieren .....	479
5.1.2	Typübereinstimmung .....	482
5.1.3	Funktions-Templates über mehrere Module .....	483
5.1.4	Spezialisierung von Funktions-Templates .....	483
5.1.5	Verschiedene Parameter .....	487
5.1.6	Explizite Template-Argumente .....	488
5.2	Klassen-Templates .....	489
5.2.1	Definition .....	490
5.2.2	Methoden von Klassen-Templates definieren .....	491
5.2.3	Klassen-Template generieren (Instantiierung) .....	496
5.2.4	Weitere Template-Parameter .....	501
5.2.5	Standardargumente von Templates .....	504
5.2.6	Explizite Instantiierung .....	506
5.3	STL (Standard Template Library) .....	507
5.3.1	Konzept von STL .....	508
5.3.2	Hilfsmittel (Hilfsstrukturen) .....	512
5.3.3	Allokator .....	524
5.3.4	Iteratoren .....	525
5.3.5	Container .....	530
5.3.6	Algorithmen .....	581
5.3.7	Allokatoren .....	643

## 6 Exception-Handling ..... 661

6.1	Exception-Handling in C++ .....	662
6.2	Eine Exception auslösen .....	662



6.3	Eine Exception auffangen – Handle einrichten .....	663
6.3.1	Reihenfolge (Auflösung) der Ausnahmen .....	666
6.3.2	Alternatives »catch(...)« .....	666
6.3.3	Stack-Abwicklung (Stack-Unwinding) .....	668
6.3.4	try-Blöcke verschachteln .....	670
6.3.5	Exception weitergeben .....	673
6.4	Ausnahmeklassen (Fehlerklassen) .....	676
6.4.1	Klassenspezifische Exceptions .....	678
6.5	Standard-Exceptions .....	680
6.5.1	Virtuelle Methode »what()« .....	681
6.5.2	Anwenden der Standard-Exceptions .....	681
6.6	System-Exceptions .....	686
6.6.1	bad_alloc .....	687
6.6.2	bad_cast .....	687
6.6.3	bad_typeid .....	687
6.6.4	bad_exception .....	687
6.7	Exception-Spezifikation .....	688
6.7.1	Unerlaubte Exceptions .....	689
6.7.2	terminate-Handle einrichten .....	691

## 7 C++-Standardbibliothek ..... 695

7.1	Die String-Bibliothek (string-Klasse) .....	695
7.1.1	Exception-Handling .....	697
7.1.2	Datentypen .....	697
7.1.3	Strings erzeugen (Konstruktoren) .....	698
7.1.4	Zuweisungen .....	700
7.1.5	Elementzugriff .....	702
7.1.6	Länge und Kapazität ermitteln bzw. ändern .....	703
7.1.7	Konvertieren in einen C-String .....	706
7.1.8	Manipulation von Strings .....	707
7.1.9	Suchen in Strings .....	710
7.1.10	Strings vergleichen .....	717
7.1.11	Die (überladenen) Operatoren .....	719
7.1.12	Einlesen einer ganzen Zeile .....	721
7.2	Ein-/Ausgabe Klassenhierarchie (I/O-Streams) .....	722
7.2.1	Klassen für Ein- und Ausgabe-Streams .....	724
7.2.2	Klassen für Datei-Streams (File-Streams) .....	748
7.2.3	Klassen für String-Streams .....	763
7.2.4	Die Klasse »streambuf« .....	770
7.2.5	Die Klasse »filebuf« .....	774
7.2.6	Die Klasse »stringbuf« .....	775

7.3	Numerische Bibliothek(en) .....	776
7.3.1	Komplexe Zahlen (»complex«-Klasse) .....	776
7.3.2	valarray .....	779
7.3.3	Globale numerische Funktionen (»cmath« und »cstdlib«) .....	802
7.3.4	Grenzwerte von Zahlentypen .....	806
7.3.5	Halbnumerische Algorithmen .....	811
7.4	Typerkennung zur Laufzeit .....	814
<b>8</b>	<b>Weiteres zum C++-Guru .....</b>	<b>821</b>
8.1	Module .....	821
8.1.1	Aufteilung .....	822
8.1.2	Die öffentliche Schnittstelle (Header-Datei) .....	823
8.1.3	Die private Datei .....	824
8.1.4	Die Client-Datei .....	826
8.1.5	Speicherklassen »extern« und »static« .....	827
8.1.6	Werkzeuge .....	829
8.2	Von C zu C++ .....	830
8.2.1	Notizen .....	831
8.2.2	Kein C++ .....	831
8.2.3	Kein C .....	833
8.2.4	»malloc« und »free« oder »new« und »delete« .....	834
8.2.5	»setjmp« und »longjmp« oder »catch« und »throw« .....	835
8.3	»Altes« C++ .....	835
8.3.1	Header-Dateien mit und ohne Endung .....	835
8.3.2	Standardbibliothek nicht komplett oder veraltet .....	836
8.3.3	Namespaces (Namensbereiche) .....	836
8.3.4	Schleifenvariable von »for« .....	836
8.4	UML .....	837
8.4.1	Wozu UML? .....	837
8.4.2	UML-Komponenten .....	839
8.4.3	Diagramme erstellen .....	840
8.4.4	Klassendiagramme mit UML .....	840
8.5	Programmierstil .....	881
8.5.1	Kommentare .....	881
8.5.2	Code .....	883
8.5.3	Benennung .....	884
8.5.4	Codeformatierung .....	884
8.5.5	Zusammenfassung .....	885
8.6	Entwicklungsstufen von Software .....	886
8.6.1	Auftrag bzw. Idee .....	887

8.6.2	Spezifikation und Anforderung .....	888
8.6.3	Entwurf (Design) .....	889
8.6.4	Programmieren (Codieren) .....	890
8.6.5	Testen und Debuggen .....	890
8.6.6	Freigabe (Release) .....	891
8.6.7	Wartung .....	891
8.6.8	Aktualisierung (Update) .....	892
8.7	Boost .....	892
8.7.1	Boost.Regex (reguläre Ausdrücke) .....	894
8.7.2	Boost.lostreams .....	909
8.7.3	Boost.Filesystem .....	911

## 9 Netzwerkprogrammierung und Cross-Plattform-Entwicklung in C++ ..... 917

9.1	Begriffe zur Netzwerktechnik .....	918
9.1.1	IP-Nummern .....	918
9.1.2	Portnummern .....	919
9.1.3	Host- und Domainname .....	920
9.1.4	Nameserver .....	921
9.1.5	IP-Protokoll .....	921
9.1.6	TCP und UDP .....	921
9.1.7	Was sind Sockets? .....	922
9.2	Header-Dateien zur Socketprogrammierung .....	923
9.2.1	Linux/UNIX .....	923
9.2.2	Windows .....	923
9.3	Client-Server-Prinzip .....	926
9.3.1	Loopback-Interface .....	926
9.4	Erstellen einer Client-Anwendung .....	927
9.4.1	»socket()« – Erzeugen eines Kommunikations- endpunkts .....	927
9.4.2	»connect()« – Client stellt Verbindung zum Server her .....	929
9.4.3	Senden und Empfangen von Daten .....	934
9.4.4	»close()« und »closesocket()« .....	937
9.5	Erstellen einer Server-Anwendung .....	937
9.5.1	»bind()« – Festlegen einer Adresse aus dem Namensraum .....	938
9.5.2	»listen()« – Warteschlange für eingehende Verbindungen einrichten .....	939
9.5.3	»accept()« und die Server-Hauptschleife .....	940

9.6	Cross-Plattform-Development .....	943
9.6.1	Abstraction Layer .....	943
9.6.2	Header-Datei (»socket.h«) .....	943
9.6.3	Quelldatei (»socket.cpp«) .....	945
9.6.4	TCP-Echo-Server (Beispiel) .....	956
9.6.5	Exception-Handling integrieren .....	958
9.6.6	Server- und Client-Sockets erstellen (TCP) .....	964
9.6.7	Ein UDP-Beispiel .....	974
9.7	Mehrere Clients gleichzeitig behandeln .....	976
9.8	Weitere Anmerkungen zur Netzwerkprogrammierung .....	986
9.8.1	Das Datenformat .....	986
9.8.2	Der Puffer .....	987
9.8.3	Portabilität .....	988
9.8.4	Von IPv4 nach IPv6 .....	988
9.8.5	RFC-Dokumente (Request for Comments) .....	990
9.8.6	Sicherheit .....	990
9.8.7	Fertige Bibliotheken .....	991

## 10 GUI- und Multimediaprogrammierung in C++ ..... 993

10.1	GUI-Programmierung – Überblick .....	993
10.1.1	Low-Level .....	994
10.1.2	High-Level .....	994
10.1.3	Überblick über plattformunabhängige Bibliotheken ...	995
10.1.4	Überblick über plattformabhängige Bibliotheken .....	997
10.2	Multimedia- und Grafikprogrammierung – Überblick .....	998
10.2.1	Überblick über plattformunabhängige Bibliotheken ...	998
10.2.2	Überblick über plattformabhängige Bibliotheken .....	1000
10.3	GUI-Programmierung mit »wxWidgets« .....	1001
10.3.1	Warum »wxWidgets«? .....	1001
10.3.2	Das erste Programm – Hallo Welt .....	1002
10.3.3	Die grundlegende Struktur eines »wxWidgets«- Programms .....	1005
10.3.4	Event-Handle (Ereignisse behandeln) .....	1012
10.3.5	Die Fenster-Grundlagen .....	1019
10.3.6	Übersicht über die »wxWidgets«-(Fenster-)Klassen ...	1021
10.3.7	»wxWindow«, »wxControl« und »wxControlWithItems« – die Basisklassen .....	1023
10.3.8	Top-Level-Fenster .....	1026
10.3.9	Container-Fenster .....	1050
10.3.10	Nicht statische Kontrollelemente .....	1077
10.3.11	Statische Kontrollelemente .....	1135

10.3.12	Menüs .....	1140
10.3.13	Ein Beispiel – Text-Editor .....	1156
10.3.14	Standarddialoge .....	1171
10.3.15	Weitere Elemente und Techniken im Überblick .....	1195
<b>A Anhang .....</b>		<b>1207</b>
A.1	Operatoren in C++ und deren Bedeutung (Übersicht) .....	1207
A.2	Vorrangtabelle der Operatoren .....	1209
A.3	Schlüsselwörter von C++ .....	1210
A.4	Informationsspeicherung .....	1210
A.4.1	Zahlensysteme .....	1211
A.5	Zeichensätze .....	1218
A.5.1	ASCII-Zeichensatz .....	1219
A.5.2	ASCII-Erweiterungen .....	1220
A.5.3	Unicode .....	1222
Index .....		1225

*Dieses Kapitel geht auf die Grundlagen der C++-Programmierung bzw. auf die grundlegenden Themen der meisten Programmiersprachen überhaupt ein. Von Basisdatentypen, Standard-I/O-Streams, Konstanten, lexikalischen Elementen, Operatoren, Begrenzern, verschiedenen Kontrollstrukturen und Funktionen bis zum Präprozessor finden Sie hier vieles, was Ihnen in anderen Programmiersprachen recht ähnlich (oder fast gleich) begegnet. Hierbei soll auch kurz auf die Geschichte von C++ und auf die Frage eingegangen werden, wie man eigentlich ein Programm erstellt.*

# 1 Grundlagen in C++

## 1.1 Die Entstehung von C++

Ursprünglich wurde C++ 1979 von Dr. Bjarne Stroustrup entwickelt, um Simulationsprojekte mit geringem Speicher- und Zeitbedarf zu programmieren. Auch hier lieferte (wie schon bei C) kein geringeres Betriebssystem als UNIX die Ursprungsplattform dieser Sprache. Stroustrup musste (damals noch bei den Bell Labs beschäftigt) den UNIX-Betriebssystemkern auf verteilte Programmierung hin analysieren. Für größere Projekte verwendete Stroustrup bisher die Sprachen Simula – die allerdings in der Praxis recht langsam bei der Ausführung ist – und BCPL, die zwar sehr schnell ist, aber sich für große Projekte nicht eignete.

### Simula und BCPL

*Simula* gilt als Vorgänger von Smalltalk. Viele der mit Simula eingeführten Konzepte finden sich in modernen objektorientierten Programmiersprachen wieder.

*BCPL* (Kurzform für *Basic Combined Programming Language*) ist eine um 1967 von Martin Richards entwickelte kompilierte, systemnahe Programmiersprache, abgeleitet von der *Combined/Cambridge Programming Language* CPL. Es handelt sich um eine Sprache aus der ALGOL-Familie.

### Homepage von Stroustrup

Wer mehr über Bjarne Stroustrup erfahren möchte, findet unter der URL <http://public.research.att.com/~bs/homepage.html> seine Homepage.

Stroustrup erweiterte die Sprache C um ein Klassenkonzept, wofür er die Sprache Simula-67 (mit der Bildung von Klassen, Vererbung und dem Entwurf virtueller Funktionen) und später dann Algol68 (Überladen von Operatoren; Deklarationen im Quelltext frei platzierbar) sowie Ada (Entwicklung von Templates, Ausnahmebehandlung) als Vorlage nahm. Heraus kam ein »C mit Klassen« (woraus etwas später auch C++ wurde). C wurde als Ursprung verwendet, weil diese Sprache schnellen Code erzeugte, einfach auf andere Plattformen zu portieren ist und fester Bestandteil von UNIX ist. Vor allem ist C auch eine weitverbreitete Sprache, wenn nicht sogar die am weitesten verbreitete Sprache überhaupt. Natürlich wurde auch weiterhin auf die Kompatibilität von C geachtet, damit auch in C entwickelte Programme in C++-Programmen liefen. Insgesamt wurde »C mit Klassen« zunächst um folgende Sprachelemente erweitert, auf die später noch eingegangen wird:

- ▶ Klassen
- ▶ Vererbung (ohne Polymorphismus)
- ▶ Konstruktoren, Destruktoren
- ▶ Funktionen
- ▶ Friend-Deklaration
- ▶ Typüberprüfung

Einige Zeit später, 1982, wurde aus »C mit Klassen« die Programmiersprache C++. Der Inkrementoperator ++ am Ende des C sollte darauf hinweisen, dass die Programmiersprache C++ aus der Programmiersprache C entstanden ist und erweitert wurde. Folgende neue Features sind dann gegenüber »C mit Klassen« hinzugekommen:

- ▶ virtuelle Funktionen
- ▶ Überladen von Funktionen
- ▶ Überladen von Operatoren
- ▶ Referenzen
- ▶ Konstanten
- ▶ veränderbare Freispeicherverwaltung
- ▶ verbesserte Typüberprüfung
- ▶ Kommentare mit // am Zeilenende anfügen (von BCPL)

1985 erschien dann die Version 2.0 von C++, die wiederum folgende Neuerungen enthielt:

- ▶ Mehrfachvererbung
- ▶ abstrakte Klassen

- ▶ statische und konstante Elementfunktionen
- ▶ Erweiterung des Schutzmodells um das Schlüsselwort `protected`

Relativ spät, 1991, fand das erste Treffen der ISO(*International Organisation for Standardization*)-Workgroup statt, um C++ zu standardisieren, was 1995 zu einem *Draft Standard* führte.

#### Draft Standard

Draft Standard ist die Vorstufe zum Standard. Das Protokoll hat die Analyse- und Testphase bestanden, kann jedoch noch modifiziert werden.

Allerdings dauerte es wiederum drei weitere Jahre, bis 1998 C++ dann endlich von der ISO genormt wurde (ISO/IEC 14882:1998). Erweitert wurde C++ in dieser Zeit um folgende Features:

- ▶ Templates
- ▶ Ausnahmebehandlung
- ▶ Namensräume
- ▶ neuartige Typumwandlung
- ▶ boolesche Typen

Natürlich entstanden während der Zeit der Weiterentwicklung von C++ auch eine Menge Standardbibliotheken wie bspw. die Stream-I/O-Bibliothek, die die bis dahin traditionellen C-Funktionen `printf()` und `scanf()` abgelöst haben. Ebenfalls eine gewaltige Standardbibliothek wurde von HP mit STL (Standard Template Library) hinzugefügt.

Im Jahre 2003 wurde dann die erste überarbeitete Version von ISO/IEC 14882:1998 verabschiedet und ISO/IEC 14882:2003 eingeführt. Allerdings stellt diese Revision lediglich eine Verbesserung von ISO/IEC 14882:1998 dar und keine »neue« Version. Eine neue Version von C++ (auch bekannt unter C++0x bzw. C++200x) soll angeblich noch in diesem Jahrzehnt erscheinen – zumindest deutet der Name dies an.

Seit Ende 2006 wurde als Termin für die Fertigstellung das Jahr 2009 erwähnt. Somit wurde aus C++0x die Abkürzung C++09. Ob der Termin tatsächlich eingehalten wird, steht natürlich wieder auf einem anderen Blatt. Und neben dem Einhalten des Termins müssen auch die Compiler der Hersteller entsprechend erneuert werden. Man kann nur hoffen, dass die Compiler-Hersteller den neuen C++09-Standard konsequent umsetzen werden (beim g++, denke ich, wird dies recht schnell gehen), so dass dann bis 2010 bzw. spätestens 2011 alle Compiler



auf den neuen Standard aufbauen. Aber wie gesagt, das ist lediglich Wunschen des Autors.

Zu einem der Top-Features im neuen C++09-Standard gehört ganz klar die Unterstützung von Threads. Ein Thema, das früher vom Standardisierungskomitee nie beachtet wurde, muss jetzt im Zeitalter von Mehrprozessorumgebungen einfach standardisiert werden. Der C++09-Standard wird auf jeden Fall eine eigene Bibliothek zur Unterstützung von Threads enthalten.

Auch die Programmbibliothek wird u. a. um reguläre Ausdrücke, Zufallsbibliothek, intelligente Zeiger, neue untergeordnete assoziative Container, Tupel, Werkzeuge für C++-Metaprogrammierung usw. erweitert. Viele dieser Erweiterungen sind Teil der Boost-Bibliothek und werden mit minimaler Veränderung übernommen. Interessant ist auch, dass der C99-Standard in einer für C++ geänderten Form enthalten sein wird.

Auch der Sprachkern soll im C++09-Standard verbessert bzw. vereinfacht werden. Neu hinzukommen werden Konzepte (engl.: *concepts*) zur Spracherweiterung.



#### Hinweis

Weitere Informationen (Weblinks) zum künftigen C++09-Standard habe ich Ihnen auf der Buch-CD zusammengestellt.

### 1.1.1 Aufbau von C++

C++ ist im Gegensatz zu Sprachen wie Smalltalk oder Eiffel keine reine objektorientierte Sprache – genauer gesagt: Smalltalk bzw. Eiffel wurden ohne Wenn und Aber als objektorientierte Sprachen entwickelt. C++ hingegen entstand ja aus C – womit es sich hierbei um eine Programmiersprache mit objektorientierter Unterstützung handelt, was zum einen Nachteile, zum anderen aber auch Vorteile mit sich bringt (siehe »Stärken von C++« und »Schwächen von C++« weiter unten in diesem Abschnitt).



#### Hinweis

Sofern Sie bereits mit C vertraut sind, können Sie die folgenden Zeilen als »Update« betrachten, das Sie darüber informiert, was mit C++ alles neu auf Sie (als C-Umsteiger) zukommt.

## Erweiterungen von C

Neben der objektorientierten Unterstützung bietet C++ auch sprachliche Erweiterungen von C an. Zu den bekanntesten Erweiterungen gehören folgende Punkte:

- ▶ Inline-Funktionen
- ▶ Default-Argumente
- ▶ Referenztypen
- ▶ Überladen von Funktionen
- ▶ Überladen von Operatoren
- ▶ Templates
- ▶ Ausnahmebehandlung

## Objektorientierte Unterstützung

Die objektorientierte Unterstützung von C++ enthält folgende Möglichkeiten:

- ▶ Klassen bilden
- ▶ Zugriff auf Daten und Elementfunktionen mit `public`-, `private`- oder `protected`-Spezifikationen steuern
- ▶ Klassen vererben (auch mehrfach)
- ▶ polymorphe Klassen bilden

## Stärken von C++

Wie bereits erwähnt, hat C++, wie jede andere Sprache auch, einige »schwache« und »starke« Seiten. Zu den Stärken von C++ gehören folgende Punkte:

- ▶ maschinennahes Programmieren
- ▶ Erzeugung von hocheffizientem Code
- ▶ hohe Ausdrucksstärke und Flexibilität
- ▶ für umfangreiche Projekte geeignet
- ▶ sehr weite Verbreitung
- ▶ Keine Organisation (wie z. B. bei Java) hat hier ihre Finger mit im Spiel (die Standardisierung erfolgt durch die ISO).
- ▶ viele Möglichkeiten für die Metaprogrammierung
- ▶ C-kompatibel – dadurch kann das Programm, das in C erstellt wurde, weiterhin unverändert verwendet werden. Außerdem braucht sich ein C-Programmierer beim Umstieg nur mit den Erweiterungen und der objektorientierten Programmierung auseinanderzusetzen.

# Index

- (Operator) 60  
-- (Operator) 63  
!(Operator) 79  
!= (Operator) 77  
# (Präprozessor) 123  
#define 123  
#elif 130  
#else 130  
#endif 130  
#error 128  
#if 129  
#ifdef 129  
#ifndef 129  
#include 127  
#pragma 129  
#pragma pack 218  
#undef 126  
% (Operator) 60  
%= (Operator) 62  
& (Operator) 65  
& (Referenz) 149  
&& (Operator) 79  
\* (Indirektionsoperator) 137  
\* (Operator) 60  
\*= (Operator) 62  
+ (Operator) 60  
++ (Operator) 63  
+= (Operator) 62  
. (Punktoperator) 275  
.\* (Operator) 353  
/ (Operator) 60  
/\* \*/ (Kommentar) 68  
// (Kommentar) 68  
/= (Operator) 62  
< (Operator) 77  
<< (Operator) 67  
<< (Shift-Operator) 56  
    *Überladen* 381  
<= (Operator) 77  
<algorithm> 581  
<boost/regex.hpp> 899  
<float> 50, 811  
<limits> 50, 811  
<cmath> 802  
<complex> 776  
    <cstdlib> 147, 802  
    <cstring> 162, 164  
    <deque> 551  
    <exception> 687, 689  
    <float.h> 50  
    <fstream> 748, 751  
    <funktional> 520  
    <iomanip> 730  
    <iostream> 54, 722  
    <limits.h> 50  
    <limits> 806  
    <list> 540  
    <memory> 645  
    <new> 323, 687  
    <numeric> 811  
    <queue> 557, 561  
    <set> 565  
    <sstream> 764  
    <stack> 554  
    <stdexcept> 681, 689  
    <stdlib.h> 147  
    <string.h> 162  
    <string> 695  
    <stringstream> 764  
    <typeinfo> 687, 814  
    <utility> 512  
    <valarray> 779  
    <vector> 534  
-= (Operator) 62  
== (Operator) 77  
> (Operator) 77  
-> (Pfeiloperator) 277  
->\* (Operator) 353  
>= (Operator) 77  
>> (Operator) 67  
>> (Shift-Operator) 57  
    *Überladen* 381  
?: (Operator) 78  
[] (Index) 152  
[] (Indexoperator) 378  
[] Operator 702  
^ (Operator) 66  
\_\_attribut\_\_ 217  
\_\_cplusplus 246  
\_Bool 40

| (Operator) 66  
 || (Operator) 79  
 ~ (Operator) 67

## A

---

Abgeleitete Klasse 392  
   *Basisinitialisierer* 405  
   *Destruktor* 407, 431  
   *Elementinitialisierer* 406  
   *Explizite Typumwandlung* 412  
   *Klassenbibliotheken erweitern* 413  
   *Konstruktor* 404  
   *protected* 407  
   *public* 399  
   *Redefinition* 401  
   *Standardkonstruktor* 405  
   *Typumwandlung* 410  
   *Zugriff auf Basisklasse* 400  
   *Zugriffsrechte* 399, 407  
 abs() 777, 787  
 Absolutwert 803  
 Abstrakte Basisklasse 435  
 Abstrakte Klasse  
   *Konstruktor* 438  
   *Kopierkonstruktor* 439  
 accept() 940  
 accumulate() 811  
 acos() 787, 802  
 Ada 26  
 Adapter-Klassen 553  
 address() 647  
 adjacent\_difference() 812  
 adjacent\_find() 586  
 advance() 529  
 AF\_BLUETOOTH 928  
 AF\_INET 928  
 AF\_INET6 928  
 AF\_IRDA 928  
 AF\_UNIX 928  
 Aktualparameter 104  
 Algol68 26  
 Algorithmen  
   *binäre Suche* 624  
   *halbnumerische* 811  
   *Heap-* 635  
   *Laufzeit* 533  
   *lexikografischer Vergleich* 639  
   *Maximum* 638  
 Algorithmen (Forts.)  
   *Mengenoperationen* 630  
   *Minimum* 638  
   *Mischen* 627  
   *mit Prädikat (STL)* 581  
   *nicht verändernde Sequenz-* 581  
   *Permutation (STL)* 641  
   *sortieren* 620  
   *STL* 509, 581  
 Alias-Name 149  
 Allegro 999  
 allocate() 646  
 Allokator 524, 643  
   *Adresse ermitteln* 647  
   *Datentypen* 646  
   *Elemente konstruieren* 647  
   *Elemente zerstören* 647  
   *Maximale Größe* 647  
   *selbst definieren* 648  
   *Speicher freigeben* 646  
   *Speicher reservieren* 646  
   *Standard-* 645  
   *Vergleichsoperatoren* 648  
   *void* 648  
 ANSI 44  
 append() 708  
 apply() 782  
 Arcuscosinus 802  
 Arcussinus 802  
 Arcustangens 802  
 arg() 777  
 Array 152  
   *Anzahl Elemente ermitteln* 156  
   *assoziatives* 380  
   *Bereichsüberschreitung* 155  
   *Buffer Overflow* 155  
   *call by reference* 185  
   *char* 159  
   *deklarieren* 152  
   *delete* 176  
   *dynamisch* 176  
   *Funktionsparameter* 185  
   *Funktionsparameter (mehrdim.)* 187  
   *Heap* 176  
   *initialisieren* 153  
   *Klasse* 316  
   *mehrdimensional* 157  
   *new* 176  
   *Objekt* 316

- Array (Forts.)
    - Rückgabewert aus Funktionen* 195
    - struct* 204
    - Strukturen* 204
    - Teilvektoren* 788
    - überladen* 378
    - Werte eingeben* 157
    - Zeiger* 166
    - zweidimensional (dynamisch)* 179
  - ASCII-Code 44
  - ASCII-Erweiterungen 1220
  - ASCII-Zeichensatz 41, 1219
  - asin() 787, 802
  - assign() 536, 542, 553, 701
  - at() 534, 551, 702
  - atan() 787, 802
  - atan2() 788, 802
  - Aufzählungstyp
    - enum* 221
  - Ausgabe
    - blockweise* 758
    - Datei* 748
    - Flag* 742
    - Status* 742
    - synchronisieren* 741
    - unformatierte* 736
    - zeichenweise* 754
    - zeilenweise* 756
  - auto 250
  - average case 533
- B**
- 
- back() 534, 540, 551, 558
  - back\_inserter 527
  - bad() 743, 763
  - bad\_alloc 687
  - bad\_cast 687
  - bad\_exception 687
  - bad\_typeid 687, 818
  - basic\_string 696
  - Basisdatentyp 39
    - Definition* 39
    - Deklarationen* 39
  - Basisinitialisierer 405
  - Basisklasse
    - abstrakt* 435, 436
  - BCD 49
  - BCPL 25
  - Bedingte Kompilierung 129
  - Bedingungsoperator 78
  - before() 818
  - begin() 529
  - BEGIN\_EVENT\_TABLE() 1010
  - Begrenzer 37
    - geschweifte Klammern* 38
    - Gleichheitszeichen (=)* 38
    - Komma (,)* 38
    - Semikolon (;)* 37
  - Bereichsoperator 226
  - best case 533
  - Bezeichner 35
    - importieren* 233
    - Namensraum* 233, 236
  - bidirectional\_iterator 526
  - Binärsystem 1212
  - binary\_search() 624
  - bind() 938
  - bind1st 523
  - bind2nd 523
  - Bit-Felder 216
  - Bit-Operatoren 64
  - Blockweise Ausgabe 758
  - Blockweise Eingabe 758
  - bool 40
    - false* 40
    - true* 40
  - Bool-Typumwandlung 259
  - Boost 892
    - Regex* 894
    - Regex (Konstruktor)* 898
  - braces 38
  - break 84, 96
  - Buffer Overflow 164
  - Bytes 1218
- C**
- 
- C
- ++ 26
  - \_Bool* 40
  - Casts* 260
  - Erweiterungen* 29
  - extern* 245
  - free()* 147
  - Header-Dateien* 244
  - in C++ verwenden* 244
  - kein C* 833

- C (Forts.)
  - malloc()* 147
  - mit Klassen* 26
  - Umschreiben nach C++* 830
- C++
  - Aufbau* 28
  - C* 830
  - Entstehung* 25
  - Geschichte* 25
  - kein C++* 831
  - Operatoren* 1207
  - Schlüsselwörter* 1210
  - Schwächen* 30
  - Standard* 27
  - Stärken* 29
  - Stroustrup* 25
  - veraltetes* 835
- C++09 27
- c\_str()* 706
- call by reference 182, 185, 201, 309
  - Referenzen* 184
- call by value 103, 181, 201, 308
- capacity()* 535, 704
- case-Marke 83
- catch
  - setjmp* 835
- catch()* 663
- catch(...)* 666
- ceil()* 803
- cerr* 55, 56
- cfloat* 50, 811
- char* 41, 696
  - Array* 159
  - signed* 41
  - unsigned* 41
- CHAR\_BIT 41, 50
- CHAR\_MAX 50
- CHAR\_MIN 50
- cin* 55, 57
  - get()* 58
  - getline()* 59, 161
- class* 267
  - struct* 267
- clear()* 535, 541, 552, 567, 575, 704, 743, 762
- climits* 41, 44, 50, 811
- clog* 55
- close()* 749, 774, 937
- closesocket()* 937
- cmath* 802
- Code* 883
  - Formatierung* 884
- Codespeicher* 143
- compare()* 717
- Compiler* 31
  - Borlands Free Command Line Tools* 34
  - GNU g++* 33
  - Microsoft Visual C++* 33
- complex* 776
- conj()* 777
- Connect()* 1017
- connect()* 929
- const* 53, 125, 148, 195, 221, 254
  - define* 125
  - Klasseneigenschaft* 341
- const\_cast<>* 262
- const\_iterator* 527
- construct()* 647
- Container* 508, 530
  - abstrakte* 553
  - assoziative* 564
  - bit\_vector* 579
  - bitset* 580
  - Datentypen* 530
  - deque* 551
  - eigener Allokator* 648
  - entwickeln* 656
  - hash\_* 580
  - Laufzeitklassen* 532
  - list* 540
  - map* 572
  - Methoden* 531
  - multimap* 572
  - multiset* 564
  - Operatoren* 531
  - priority\_queue* 560
  - queue* 557
  - sequentielle* 532
  - set* 564
  - slist* 579
  - stack* 554
  - string* 580
  - vector* 533
  - wstring* 580
- continue* 97
- copy()* 597, 706
- copy\_backward()* 597
- copy\_n()* 598

cos() 777, 787, 803  
 cosh() 777, 787, 803  
 Cosinus 803  
 count() 568, 576, 596  
 count\_if() 596  
 cout 55, 56, 725  
 Create() 1021  
 Cross-Plattform-Entwicklung 943  
   *Abstraction Layer* 943  
   *Client-Sockets* 964  
   *Exception-Handling* 958  
   *Hauptprogramm* 956  
   *Header-Datei* 943  
   *Mehrere Clients gleichzeitig* 976  
   *Quelldatei(en)* 945  
   *Server-Sockets* 964  
   *TCP-Echo-Server* 956  
   *UDP-Client* 975  
   *UDP-Server* 974  
 cshift() 782  
 cstdlib 147, 802  
 C-String 159  
   *Bibliotheksfunktionen* 161  
   *Buffer Overflow* 164  
   *deklarieren* 160  
   *eingeben* 161  
   *initialisieren* 160  
   *Probleme* 164  
   *string* 706  
   *Tabellen* 173  
   *Terminierungszeichen* 160  
   *Zeiger* 171  
 cstring 162, 164  
 curly brackets 38

## D

---

data() 706  
 Datei  
   *Ausgabe* 748, 751  
   *blockweise ausgeben* 758  
   *blockweise einlesen* 758  
   *Eingabe* 748, 751  
   *Exception* 763  
   *Fehlerbehandlung* 762  
   *Positionierung* 760  
   *zeichenweise ausgeben* 754  
   *zeichenweise einlesen* 754  
   *zeilenweise ausgeben* 756

Datei (Forts.)  
   *zeilenweise einlesen* 756  
   *Zugriff (wahlfrei)* 760  
 Datenabstraktion 266  
 Datenkapselung 266  
 Datenspeicher 143  
 Datenstrom 753  
 Datentyp 39  
   *fortgeschritten* 133  
   *Grenzwerte* 806  
   *Information zur Laufzeit* 814  
 DBL\_DIG 51  
 DBL\_EPSILON 51  
 DBL\_MANT\_DIG 50  
 DBL\_MAX 51  
 DBL\_MAX\_10\_EXP 51  
 DBL\_MAX\_EXP 51  
 DBL\_MIN 51  
 DBL\_MIN\_10\_EXP 51  
 DBL\_MIN\_EXP 51  
 deallocate() 647  
 Debuggen 890  
 dec 726  
 DECLARE\_EVENT\_TABLE() 1010  
 default 84  
 define-Direktive 123  
 Definition 39, 99  
 Deklaration 39, 99  
 Dekrementoperator 63  
 delete 146, 322  
   *Array* 176  
   *free()* 834  
 denorm\_min() 809  
 deque (STL-Container) 551  
   *Datentypen* 551  
   *Methoden* 551  
 Design 889  
 Destroy() 1021  
 destroy() 647  
 Destruktor 293  
   *abgeleitete Klasse* 407, 431  
   *aufrufen (implizit)* 295  
   *definieren* 293  
   *deklarieren* 293  
   *inline* 298  
   *virtueller* 431  
 Dezimalkomma 47  
 Dezimalsystem 1212  
 DirectX 1000

Disconnect() 1017  
 distance() 529  
 divides<> 522  
 do...while 91  
 domain\_error 685  
 Domainname 920  
 double 47  
 Draft Standard 27  
 DSP 41  
 Dualsystem 1212  
 dynamic\_cast<> 264, 439, 819  
   *bad\_cast* 687  
 Dynamisch Speicher anlegen 143  
 Dynamische Datenstrukturen 210

## E

---

EBCDIC-Code 44  
 Eingabe  
   *blockweise* 758  
   *Datei* 748  
   *Flag* 742  
   *Status* 742  
   *synchronisieren* 741  
   *unformatierte* 738  
   *zeichenweise* 754  
   *zeilenweise* 756  
 Einlesen  
   *String* 721  
 Elementfunktion 269  
 Elementinitialisierer 338, 406  
 Elementzeiger 352  
 elif-Direktive 130  
 Ellipse 116  
 else if-Verzweigung 73  
 else-Direktive 130  
 else-Verzweigung 69  
 empty() 704  
 END\_EVENT\_TABLE() 1010  
 endif-Direktive 130  
 endl 725  
 ends 725  
 Entwicklungsumgebung 31  
   *IDE* 31  
 Entwurf 889  
 enum 221  
 EOF 58  
 eof() 743, 762  
 epsilon() 809  
 equal() 594  
 equal\_range() 567, 576, 625  
 equal\_to<> 522  
 erase() 535, 541, 552, 566, 575, 708  
 error-Direktive 128  
 Escape-Zeichen 42  
 EVT\_BUTTON() 1010  
 Exception 661  
   *alternatives catch()* 666  
   *auffangen* 663  
   *auflösen* 666  
   *auslösen* 662  
   *bad\_alloc* 687  
   *bad\_cast* 687  
   *bad\_exception* 687  
   *bad\_typeid* 687  
   *catch()* 663  
   *domain\_error* 685  
   *invalid\_argument* 684  
   *ios\_base::failure* 685  
   *Klasse* 676  
   *klassenspezifisch* 678  
   *Laufzeitfehler* 681  
   *length\_error* 684  
   *logische Fehler* 681  
   *new* 144  
   *out\_of\_range* 682  
   *overflow\_error* 686  
   *range\_error* 686  
   *set\_terminate()* 691  
   *set\_unexpected()* 689  
   *Spezifikation* 688  
   *Stack-Abwicklung* 668  
   *Standard-* 680  
   *System-* 686  
   *terminate()-Handle* 691  
   *throw* 662  
   *try* 663  
   *try-Blöcke verschachteln* 670  
   *underflow\_error* 686  
   *unerlaubte* 689  
   *unexpected()* 689  
   *weitergeben* 673  
   *what()* 681  
 exception 680  
 Exception-Handling 661  
 Exception-Klasse 676  
 exp() 777, 787, 803  
 Explizite Typumwandlung 260



Exponent 46  
 Extern 245, 251, 827

## F

---

fabs() 803  
 fail() 743, 763  
 false 40  
 FD\_CLR() 979  
 FD\_ISSET() 979  
 FD\_SET() 979  
 FD\_ZERO() 979  
 Fehlerbehandlung  
   *Datei* 762  
 Fehlerklasse 676  
 Felder 152  
 filebuf 774  
 fill() 607, 734  
 fill\_n() 607  
 find() 567, 575, 582, 711  
 find\_end() 593  
 find\_first\_not\_of() 714  
 find\_first\_of() 585, 713  
 find\_if() 582  
 find\_last\_not\_of() 715  
 find\_last\_of() 714  
 fixed 726  
 flags() 746  
 Fließkommadatentypen 46  
 Fließkommazahlen 36  
 float 46  
 float.h 50  
 floor() 803  
 FLT\_DIG 51  
 FLT\_EPSILON 51  
 FLT\_MANT\_DIG 50  
 FLT\_MAX 51  
 FLT\_MAX\_10\_EXP 51  
 FLT\_MAX\_EXP 51  
 FLT\_MIN 51  
 FLT\_MIN\_10\_EXP 51  
 FLT\_MIN\_EXP 51  
 FLT\_RADIX 50  
 FLTK (GUI-Bibliothek) 995  
 Fluchtzeichen 42  
 flush 56, 725  
 flush() 736  
 fmod() 803  
 for 93

for\_each() 582  
 Formalparameter 104  
 Fortgeschrittene Typen 197  
   *Aufzählungstypen (enum)* 221  
   *Bit-Felder* 216  
   *Klasse* 267  
   *Strukturen (struct)* 198  
   *Unions* 218  
   *verkettete Liste* 210  
 forward\_iterator 526  
 Fox (GUI-Bibliothek) 996  
 free() 147, 834  
 frexp() 803  
 friend 349  
   *Operator-Überladung* 365  
 front() 534, 540, 551, 558  
 front\_inserter 527  
 fstream 748  
 funktional 520  
 Funktionen 99  
   *Array als Rückgabewert* 195  
   *Arrays als Parameter* 185  
   *Attribute* 255  
   *Aufruf* 102  
   *Bezeichner* 100  
   *call by reference* 182  
   *call by value* 103, 181  
   *const-Zeiger als Rückgabewert* 195  
   *Definition* 99  
   *Deklaration* 99  
   *Exception* 688  
   *friend* 349  
   *globale Variablen* 109  
   *Gültigkeitsbereich* 226  
   *inline* 117  
   *lokale Variablen* 109  
   *main()* 121, 188  
   *mathematische* 802  
   *mehrdimensionale Arrays* 187  
   *mehrere Rückgabewerte* 196  
   *Objekt als Argument* 307  
   *pair* 516  
   *Parameter* 100  
   *Parameterübergabe* 103, 181  
   *Polymorphie* 113  
   *Prototyp* 100  
   *Referenz als Rückgabewert* 194  
   *Rekursion* 120  
   *Rückgabebetyp* 100, 105

## Funktionen (Forts.)

- Rückgabewert* 105, 190
  - Rücksprungadresse* 117
  - Scope-Operator* 226
  - Spezifizierer* 100
  - Stack* 117
  - Standardparameter* 110
  - Strukturen als Parameter* 201
  - Template* 477
  - überladen* 113
  - Zeiger als Parameter* 182
  - Zeiger als Rückgabewert* 190
  - zwei Werte zurückgeben* 516
- Funktionsadapter 522
- bind1st* 523
  - bind2nd* 523
  - not1* 523
  - not2* 523
  - ptr\_fun* 523
- Funktionsobjekte 520
- divides<>* 522
  - equal\_to<>* 522
  - greater<>* 522
  - greater\_equal<>* 522
  - less<>* 522
  - less\_equal<>* 522
  - logical\_and<>* 522
  - logical\_not<>* 522
  - logical\_or<>* 522
  - minus<>* 522
  - modulus<>* 522
  - multiplies<>* 522
  - negate<>* 522
  - not\_equal\_to<>* 522
  - plus<>* 522
- Funktions-Template 477
- Argumente* 482
  - definieren* 479
  - Gültigkeitsbereich* 483
  - Spezialisierung* 483
  - Typ explizit festlegen* 488
  - Typübereinstimmung* 482
  - unterschiedliche Parameter* 487

**G**

- 
- Ganzzahlen 36, 44
  - Garbage Collection 146
  - gcount() 739

- generate() 607
- generate\_n() 607
- get() 58, 738, 754
- gethostbyname() 932
- getline() 59, 161, 721, 739, 756
- getservbyname() 931
- Gleitkomma-Promotion 257
- Gleitkomma-Typumwandlung 258
- Gleitkommazahlen 46
  - Genauigkeit* 47
  - Rechenfehler* 48
- GLU 998
- GLUT 999
- good() 743, 762
- goto 96
- Grafikprogrammierung 993
- greater<> 522
- greater\_equal<> 522
- Grenzwerte 806
- gslice 792
- gtkmm (GUI-Bibliothek) 996
- GUI-Programmierung 993
  - FLTK* 995
  - Fox* 996
  - gtkmm (gtk-)* 996
  - High-Level* 994
  - Low-Level* 994
  - MFC* 997
  - OWL* 997
  - Qt* 996
  - RAD-Tools* 995
  - VCL* 997
  - wxWidgets* 1001
- Gültigkeitsbereich 225
  - Funktionen* 226
  - Klassen* 228
  - Lokal* 226
  - Namensraum* 228

**H**

- 
- Halbnumerische Algorithmen (STL) 811
  - Header-Datei
    - ANSI-C* 244
    - mit Endung* 835
    - Namensraum* 242
    - ohne Endung* 835
  - Heap
    - Algorithmen (STL)* 635

- Heap (Forts.)
    - Array* 176
  - Heap-Speicher 143
  - hex 726
  - Hexadezimalsystem 1214
  - Hostname 920
  - htonl() 931
  - htons() 931
- I**
- 
- ifdef-Direktive 129
  - if-Direktive 129
  - ifndef-Direktive 129
  - ifstream 751
  - if-Verzweigung 69
  - ignore() 739
  - imag() 777
  - IMPLEMENT\_APP() 1004
  - in\_avail() 771
  - INADDR\_ANY 938
  - include-Direktive 127
  - includes() 630
  - indirect\_array 800
  - Indirektionsoperator 137
  - Informationsspeicherung 1210
  - Inkrementoperator 63
  - inline 118, 126
    - define* 126
    - Destruktor* 298
    - Konstruktor* 298
    - Methode* 296
  - inner\_product() 811
  - inplace\_merge() 628
  - input\_iterator 526
  - insert() 535, 541, 552, 567, 575, 707
  - inserter 527
  - int 44
  - INT\_MAX 44, 50
  - INT\_MIN 44, 50
  - Integral-Gleitkomma-Typumwandlung 258
  - Integral-Promotion 256
  - Integral-Typumwandlung 257
  - internal 734
  - invalid\_argument 684
  - iomanip 730
  - ios 723
    - ios::adjustfield 745, 746
    - ios::app 749, 765
    - ios::ate 749, 765
    - ios::badbit 685, 742, 763
    - ios::basefield 745
    - ios::beg 761
    - ios::binary 749, 765
    - ios::cur 761
    - ios::dec 745
    - ios::end 761
    - ios::eofbit 685, 742, 763
    - ios::failbit 685, 742, 763
    - ios::fixed 745
    - ios::floatfield 745
    - ios::goodbit 742
    - ios::hex 745
    - ios::in 749, 765
    - ios::internal 734, 746
    - ios::left 734, 746
    - ios::oct 745
    - ios::out 749, 765
    - ios::right 734, 746
    - ios::scientific 745
    - ios::seekdir 760
    - ios::showbase 746
    - ios::showpoint 746
    - ios::showpos 746
    - ios::trunc 749, 765
    - ios::uppercase 746
    - ios\_base::failure 685
  - iostream 722
  - IP-Nummer 918
    - dynamisch* 918
    - statisch* 918
  - IP-Protokoll 921
  - IPv4 919
  - IPv6 919
  - is\_heap() 636
  - is\_open() 748, 774
  - is\_sorted() 622
  - istream 736
  - istringstream 764
  - iter\_swap() 600
  - Iterationen 88
  - Iterator 508, 525, 527
    - advance()* 529
    - bidirektionaler* 526
    - distance()* 529
    - Distanzen* 529
    - Einfüge-* 527

## Iterator (Forts.)

- Forward-* 526
- Funktionen* 529
- Input-* 526
- Kategorien* 526
- konstanter* 527
- Output-* 526
- Random-Access-* 526, 624
- reverser* 527
- Stream-* 528
- Zustand* 525

**K**

key\_comp() 568, 576

## Klasse 267

- abgeleitete* 392
- abstrakt* 435, 436
- als Eigenschaft* 332
- Array* 316
- Bereichsoperator* 270
- call by reference* 309
- call by value* 308
- Container erstellen* 656
- deklarieren* 268
- Destruktor* 293
- Direkter Zugriff* 275
- dynamische Eigenschaften* 324
- Eigenschaften* 268
- eingebettet* 332
- Elementzeiger* 352
- enum-Eigenschaften* 278
- Exception* 676
- Fehler-* 676
- friend* 349
- indirekter Zugriff* 277
- konstante Klasseeigenschaft* 341
- Konstruktor* 285
- Konvertierungsfunktion* 390
- Konvertierungskonstruktor* 388
- Mehrfachvererbung* 463
- Member* 268
- Methode* 268, 269, 295
- mit UML visualisieren* 840
- Objekte deklarieren* 271
- Objekte verwenden* 307
- Operator-Überladung* 358
- Polymorphie* 414
- Polymorphismus* 414

## Klasse (Forts.)

- private* 272, 407
  - Programm organisieren* 281
  - protected* 407
  - public* 272, 407
  - Read-only-Methoden* 303
  - Referenzen* 310
  - Scope-Operator* 270
  - statische Klasseeigenschaft* 343
  - Template* 489
  - this-Zeiger* 305
  - Typumwandlung* 388
  - Vererbung* 392
  - verkettete Liste* 441
  - verschachteln* 340
  - virtual* 433
  - Zeiger auf Eigenschaften* 352
  - Zugriff auf Elemente* 274
  - Zugriffsmethoden* 299
  - Zugriffsrechte* 272
- Klassen-Array 316
- deklarieren* 316
  - dynamisch* 318
  - initialisieren* 316
  - Zugriff* 317
- Klassenbibliotheken erweitern 413
- Klassenelemente
- dynamisch* 324
- Klassenmethoden 295
- Klassen-Template 489
- Argumente* 501
  - Definition* 490
  - explizite Instanziierung* 506
  - generieren* 496
  - Instanziierung* 496
  - Methode* 491
  - Parameter* 501
  - Standardargumente* 504
  - verkettete Liste* 490
- Klassen-Typumwandlung 259
- Kommentare 68, 881
- Komplexe Zahlen 776
- Konstanten 53
- Konstruktor 285
- abgeleitete Klasse* 404
  - abstrakte Klassen* 438
  - aufrufen* 289
  - definieren* 287
  - deklarieren* 286

Konstruktor (Forts.)  
*inline* 298  
*Konvertierungs-* 388  
*Kopier-* 328  
*Standard-* 292  
*virtueller* 433  
 Kontrollstrukturen 69  
*Verzweigungen* 69  
 Konvertierungsfunktion 390  
 Konvertierungskonstruktor 388  
 Kopierkonstruktor  
*Abstrakte Klasse* 439

## L

---

Laufzeitklassen 532  
*exponentiell* 533  
*konstant* 532  
*linear* 532  
*logarithmisch* 532  
*N\*log N* 532  
*quadratisch* 532  
 LDBL\_DIG 51  
 LDBL\_EPSILON 51  
 LDBL\_MANT\_DIG 51  
 LDBL\_MAX 51  
 LDBL\_MAX\_10\_EXP 51  
 LDBL\_MAX\_EXP 51  
 LDBL\_MIN 51  
 LDBL\_MIN\_10\_EXP 51  
 LDBL\_MIN\_EXP 51  
 ldiv\_t 806  
 left 734  
 length() 704  
 length\_error 684, 697  
 less\_equal<> 522  
 lexicographical\_compare() 639  
 Limits  
*Ganzzahlen* 50  
*Gleitkommazahlen* 50  
 limits 806  
 limits.h 41, 44, 50  
 Linker 31  
 Linksassoziativität 60  
 list (STL-Container) 540  
*Datentypen* 540  
*Methoden* 540  
 listen() 939  
 Literale 36

locale 47  
 log() 777, 787, 803  
 log10() 777, 787, 803  
 Logarithmus 803  
 logic\_error 681  
 logical\_and<> 522  
 logical\_not<> 522  
 logical\_or<> 522  
 Logische Operatoren 79  
 long 45  
 long double 47  
 long int 45  
 long long 44, 46  
 LONG\_MAX 50  
 LONG\_MIN 50  
 longjmp 835  
 Loopback-Interface 926  
 lower\_bound() 567, 576, 625

## M

---

main() 121  
*Argumente* 122, 188  
*Funktionsparameter* 188  
*Parameter* 122  
*Rückgabewert* 122  
 make\_heap() 636  
 MAKEWORD() 925  
 Makros 123  
 malloc() 147, 834  
 Manipulatoren  
*eigene* 728  
*eigene mit Parameter* 731  
*istream* 738  
*mit Parameter* 730  
*ostream* 725  
 Mantisse 47  
 map (STL-Container) 572  
*Datentypen* 574  
*Methoden* 574  
 Maschinensprache  
*Maschinencode* 31  
 Maschinenwörter 1218  
 mask\_array 797  
 math.h 802  
 Mathematische Funktionen 802, 803  
 Matrix 789  
 max() 639, 782, 808  
 max\_element() 639

max\_size() 648, 704  
 MB\_LEN\_MAX 50  
 Mehrfachvererbung 463  
     *Erzeugung der Objekte (Reihenfolge)* 474  
     *indirekte Basisklasse erben* 467  
     *virtuelle indirekte Basisklasse erben* 471  
 Member 268  
 memchr() 165  
 memcmp() 165  
 memcpy() 165  
 memmove() 165  
 memset() 165  
 Mengenoperationen  
     *Algorithmen (STL)* 630  
 merge() 542, 628  
 MesaGL 999  
 Methode 269  
     *definieren* 269  
     *inline* 296  
     *inline (explizit)* 297  
     *inline (implizit)* 296  
     *Klassen-Template* 491  
     *Objekt als Rückgabewert* 314  
     *Objekte als Argument* 310  
     *Operator-Überladung* 363  
     *Read-only-* 303  
     *rein virtuell* 435  
     *Rückgabewert (virtuell)* 424  
     *Standard-* 331  
     *static* 348  
     *this* 305  
     *virtual* 417, 433  
     *virtuelle* 415  
     *virtuelle Methoden redefinieren* 420  
     *Zugriffs-* 299  
     *Zugriffsrechte (virtuell)* 425  
 MFC  
     *wxWidgets* 1206  
 MFC (GUI-Bibliothek) 997  
 min() 638, 782, 808  
 min\_element() 639  
 minus<> 522  
 Mischen  
     *Algorithmen (STL)* 627  
 mismatch() 590  
 modf() 803  
 Modularisierung 281

Module 31, 821  
     *aufteilen* 822  
     *Client-Datei* 826  
     *extern (Speicherklasse)* 827  
     *Hauptprogramm* 826  
     *Header-Datei(en)* 823  
     *main()* 826  
     *namenloser Namensraum* 828  
     *öffentliche Schnittstelle* 823  
     *organisieren* 822  
     *private Datei* 824  
     *static (Speicherklasse)* 827  
 modulus<> 522  
 multimap (STL-Container) 572  
 Multimediaprogrammierung 993  
     *Allegro* 999  
     *DirectX* 1000  
     *GLU* 998  
     *GLUT* 999  
     *MesaGL* 999  
     *OpenGL* 998  
     *SDL* 1000  
 multiplies<> 522  
 multiset (STL-Container) 564  
 mutable 253

## N

---

Name Lookup 239  
 name() 814  
 Namensauflösung 239  
 Namensbereich 836  
 Namensraum 228  
     *Alias-Name* 240  
     *anonym* 241, 828  
     *Erzeugen* 228  
     *Header-Datei* 242  
     *importieren* 233, 236  
     *namenlos* 241  
     *static* 242  
     *using* 233, 236  
     *Zugriff auf Bezeichner* 231  
 Nameserver 921  
 namespace 236  
 Namespaces 228  
 negate<> 522  
 Netzwerkprogrammierung 917  
     *Client-Server-Prinzip* 926  
     *Linux* 923

- Netzwerkprogrammierung (Forts.)  
*Windows* 923
- Netzwerktechnik 918  
*Domainname* 920  
*Hostname* 920  
*IP-Nummer* 918  
*IP-Protokoll* 921  
*IPv4* 919  
*IPv6* 919  
*Loopback-Interface* 926  
*Nameserver* 921  
*Portnummer* 919  
*Sockets* 922  
*TCP* 921  
*UDP* 921
- new 144, 322  
*Array* 176  
*bad\_alloc* 687  
*Exception* 323, 687  
*Exceptions* 144  
*Fehler-Handle* 323  
*malloc()* 834  
*nothrow* 145  
*Rückgabewert* 144  
*set\_new\_handler()* 323
- next\_permutation() 642
- norm() 777
- noshowbase 726
- noshowpoint 726
- noshowpos 726
- not\_equal\_to<> 522
- not1 523
- not2 523
- nothrow 145
- nouppercase 726
- nth\_element() 618
- ntohl() 931
- ntohs() 931
- NULL 141
- numeric 811
- numeric\_limits 806
- O**
- 
- Objekte  
*als Rückgabewert* 314  
*Array* 316  
*call by reference* 309  
*call by value* 308
- Objekte (Forts.)  
*deklarieren* 271  
*dynamisch anlegen* 318  
*Elementinitialisierer* 338  
*freigeben* 320  
*Funktionsargument* 307  
*kopieren* 328  
*kopieren (dynamisch)* 330  
*Read-only-* 307  
*Referenzen* 310  
*Teil-* 334  
*Teilobjekte initialisieren* 338  
*verwenden* 307  
*Zuweisung verhindern* 377
- Objektorientiert 265
- oct 726
- ofstream 751
- Oktalsystem 1213
- OnInit() 1004
- open() 748, 751, 752, 774
- OpenGL 998
- openmode 765
- operator 358
- operator=() 330
- Operatoren 59  
*arithmetische* 60  
*Bedingung* 78  
*Bit-* 64  
*Dekrement* 63  
*Inkrement* 63  
*logische* 79  
*Typumwandlung* 261  
*überladen* 358  
*Übersicht* 1207  
*Vergleichs-* 76  
*Vorrangtabelle* 1209
- Operator-Überladung  
*friend-Funktion* 365  
*Klassenmethode* 363
- ostream 724  
*Manipulatoren* 725
- ostreamstream 764
- out\_of\_range 682, 697, 702
- output\_iterator 526
- overflow\_error 686
- OWL (GUI-Bibliothek) 997

**P**

---

pair 512  
 pair<> 567  
 partial\_sort() 621  
 partial\_sort\_copy() 621  
 partial\_sum() 812  
 partition() 618  
 PDP 44  
 Permutation 641  
 plus<> 522  
 Pointer 133  
 polar() 777  
 Polymorphie 266, 414  
 Polymorphismus 414  
   *dynamic\_cast<>* 439  
   *dynamische Bindung* 415  
   *Methoden redefinieren* 420  
   *Rückgabewert* 424  
   *Signatur* 422  
   *statische Bindung* 415  
   *virtual* 417  
   *Zugriffsrechte* 425  
   *Zuweisungsoperator* 433  
 pop() 555, 558, 562  
 pop\_back() 535, 541, 552  
 pop\_front() 541, 552  
 pop\_heap() 635  
 PopupMenu 1154  
 Portnummer 919  
 pow() 778, 788, 803  
 Prädikat (STL) 581  
 pragma-Direktive 129  
 Präprozessor  
   *bedingte Kompilierung* 129  
   *define* 123  
   *Direktiven* 122  
   *elif* 130  
   *else* 130  
   *endif* 130  
   *error* 128  
   *if* 129  
   *ifdef* 129  
   *ifndef* 129  
   *include* 127  
   *pragma* 129  
   *undef* 126  
 precision() 726  
 prev\_permutation() 641

priority\_queue (STL-Container) 560  
   *Methoden* 561  
 private 272, 407  
 ProcessEvent() 1017  
 Programm organisieren 281  
 Programmierstil 881  
   *Benennungen* 884  
   *Code* 883  
   *Formatierung* 884  
   *Kommentare* 881  
 protected 407  
 Prototyp 100  
 ptr\_fun 523  
 public 272, 407  
 pubseekoff() 771  
 pubseekpos() 771  
 pubsetbuf() 771  
 pubsync() 771  
 Puffer  
   *filebuf* 774  
   *streambuf* 770  
   *stringbuf* 775  
 push() 555, 558, 562  
 push\_back() 535, 541, 552  
 push\_front() 541, 552  
 push\_heap() 635  
 put() 736, 754  
 putback() 739

**Q**

---

Qt (GUI-Bibliothek) 996  
 Quadratwurzel 803  
 Qualifizierer  
   *const* 254  
   *volatile* 254  
 Quantifizierer 897  
 queue (STL-Container) 557  
   *Datentypen* 558  
   *Methoden* 558  
 quiet\_NaN() 809

**R**

---

RAD-Tools 995  
 random\_access\_iterator 526  
 random\_shuffle() 616, 617  
 range\_error 686  
 rbegin() 527



rdbuf() 749  
 rdstate() 743  
 read() 739, 758  
 readsome() 739  
 real() 777  
 Rechtsassoziativität 60  
 recv() 934  
 recvfrom() 936  
 Redefinition 401  
 reelle Zahlen 46  
 Referenzen 149
 

- call by reference nachbilden* 184
- Klasse* 310
- Rückgabewert aus Funktion* 194
- Zeiger* 150

 Regex
 

- Boost* 894

 regex\_match() 899  
 regex\_replace() 906  
 regex\_search() 902  
 register 250  
 Reguläre Ausdrücke 894, 895
 

- \$ 898
- \* 898
- + 898
- . 896
- < 898
- > 898
- ? 898
- [] 896
- ^ 898
- Alternativen* 897
- B* 898
- b* 898
- beliebiges Zeichen* 896
- D* 898
- d* 898
- Einführung* 895
- Elemente (POSIX)* 895
- ersetzen* 906
- Gruppierung* 897
- Komplettsuche* 899
- Quantifizierer* 897
- S* 898
- s* 898
- Sonderzeichen* 898
- Teilsuche* 902
- Zeichenauswahl* 896
- Zeichenklassen* 896

Reguläre Ausdrücke (Forts.)
 

- Zeichen-Literale* 896

 reinterpret\_cast<> 263  
 Rekursionen 120  
 remove() 542, 609  
 remove\_copy() 610  
 remove\_copy\_if() 610  
 remove\_if() 542, 609  
 rend() 527  
 replace() 604, 708  
 replace\_copy() 605  
 replace\_copy\_if() 605  
 replace\_if() 605  
 reserve() 535, 704  
 resetiosflags() 730  
 resize() 535, 541, 552, 704, 782  
 return 106  
 reverse() 543, 600  
 reverse\_copy() 598  
 reverse\_iterator 527  
 rfind() 711  
 right 734  
 rotate() 614  
 rotate\_copy() 615  
 round\_error() 809  
 RTTI 814  
 runtime\_error 681

## S

---

sbumpc() 771  
 SCHAR\_MAX 50  
 SCHAR\_MIN 50  
 Schleifen 88
 

- do...while* 91
- for* 93
- while* 88

 Schlüsselwörter 35
 

- C++ 833, 1210

 scientific 726  
 Scope 225
 

- Klasse* 228
- lokal* 226
- Operator* 226

 SDL 1000  
 search() 593  
 search\_n() 586  
 seekg() 760  
 seekp() 760

- select() 977
  - FD\_CLR() 979
  - FD\_ISSET() 979
  - FD\_SET() 979
  - FD\_ZERO() 979
- Selektionen 69
- send() 934
- sendto() 936
- sequentielle Container 532
- set (STL-Container) 564
  - Datentypen 565
  - Methoden 566
- set\_difference() 631
- set\_intersection() 631
- set\_new\_handler() 323
- set\_symmetric\_difference() 632
- set\_terminate() 691
- set\_unexpected() 689
- set\_union() 630
- setbase() 730
- setf() 727, 746
- setfill() 730
- setiosflags() 730
- setjmp 835
- setprecision() 730
- setstate() 743
- setw() 730
- sgetc() 772
- sgetn() 772
- shift() 782
- Shift-Operator
  - << 56
  - >> 57
- short 45
- short int 45
- Show() 1020
- showbase 726
- showpoint 726
- showpos 726
- SHRT\_MAX 50
- SHRT\_MIN 50
- signaling\_NaN() 809
- Signatur 114, 422
- signed 46
- Simula 25
- sin() 778, 787, 803
- sinh() 778, 787, 803
- Sinus 803
- size() 704, 782
- sizeof 52
- Skip() 1017
- slice 789
- snnextc() 771
- socket() 927
- Socketprogrammierung 917, 923
  - accept() 940
  - Adresse festlegen (binden) 938
  - bind() 938
  - Client-Anwendung 927
  - Client-Server-Prinzip 926
  - close() 937
  - closesocket() 937
  - connect() 929
  - Daten empfangen 934
  - Daten senden 934
  - Datenformat 986
  - gethostbyname() 932
  - hostent 932
  - htonl() 931
  - htons() 931
  - IPv4 nach IPv6 ändern 988
  - Linux 923
  - listen() 939
  - ntohl() 931
  - ntohs() 931
  - Portabilität 988
  - Puffer 987
  - recv() 934
  - recvfrom() 936
  - RFC-Dokumente 990
  - select() 977
  - send() 934
  - sendto() 936
  - servent 931
  - Server-Anwendung 937
  - Server-Hauptschleife 940
  - Sicherheit 990
  - sockaddr\_in 930
  - Socket anlegen 927
  - Socket freigeben 937
  - socket() 927
  - Verbindung annehmen 940
  - Verbindung herstellen 929
  - Warteschlange einrichten 939
  - Windows 923
- Sockets 922
- sort() 543, 620
- sort\_heap() 636

- sortieren
  - Algorithmen (STL)* 620
- Speicher freigeben 146
- Speicher reservieren 143
- Speicherbereiche
  - Code* 143
  - Daten* 143
  - Heap* 143
  - Stack* 143
- Speicherklassenattribute 249
  - auto* 250
  - extern* 251
  - mutable* 253
  - register* 250
  - static* 250
- Spezifikation 888
- splICE() 542
- Sprunganweisungen 96
- sputback() 772
- sputc() 772
- sputn() 772
- sqrt() 778, 787, 803
- sstream 764
- stable\_partition() 618
- stable\_sort() 621
- Stack 117
  - Overflow* 120
- stack (STL-Container) 554
  - Datentypen* 554
  - Methoden* 555
- Stack-Speicher 143
- Stack-Unwinding 668
- Standard Template Library 507
- Standardausgabe 55, 56
- Standardbibliothek 695
- Standardeingabe 55
- Standard-Exception 680
- Standardfehlerausgabe 55
- Standardklasse
  - complex* 776
  - filebuf* 774
  - fstream* 748
  - gslice* 792
  - ifstream* 751
  - indirect\_array* 800
  - ios* 723
  - iostream* 722
  - istream* 736
  - istringstream* 764
- Standardklasse (Forts.)
  - mask\_array* 797
  - ofstream* 751
  - ostream* 724
  - ostringstream* 764
  - slice* 789
  - sstream* 764
  - streambuf* 770
  - string* 695
  - stringbuf* 775
  - stringstream* 764
  - strstream* 764
  - valarray* 779
- Standardparameter 110
- Standard-Streams 54
- static 242, 250, 827
  - Klasseneigenschaft* 343
  - Klassenmethoden* 348
- static\_cast<> 263
- Statische Klassenmethoden 348
- std 55
- stderr 55
- stdin 55
- stdlib.h 147, 802
- stdout 55
- Steuerzeichen 42
- STL 507
  - abstrakte Container* 553
  - Algorithmen* 509, 581
  - Algorithmen für Mengenoperationen* 630
  - Algorithmen zum lexikografischen Vergleich* 639
  - Algorithmen zum Mischen* 627
  - Algorithmen zum Sortieren* 620
  - Algorithmen zum Suchen* 624
  - Allokator* 524, 643
  - assoziative Container* 564
  - bit\_vector (Container)* 579
  - bitset (Container)* 580
  - Container* 508, 530
  - deque (Container)* 551
  - Funktionsobjekte* 520
  - halbnumerische Algorithmen* 811
  - hash\_ (Container)* 580
  - Heap-Algorithmen* 635
  - Hilfsmittel* 512
  - Iterator* 508, 525
  - Konzept* 508

- STL (Forts.)
  - list (Container)* 540
  - map (Container)* 572
  - multimap (Container)* 572
  - multiset (Container)* 564
  - pair* 512
  - Permutation (Algorithmen)* 641
  - priority\_queue (Container)* 560
  - queue (Container)* 557
  - selbstdefinierter Allokator* 648
  - set (Container)* 564
  - slist (Container)* 579
  - stack (Container)* 554
  - string (Container)* 580
  - vector (Container)* 533
  - Vergleichsoperatoren* 519
  - verkettete Liste* 543
  - wstring (Container)* 580
- str()* 766, 775
- strcat()* 162
- strchr()* 164
- strcpy()* 161
- Stream
  - Ausgabe-* 724
  - Datei-* 748
  - Daten-* 753
  - Eingabe-* 736
  - File-* 748
  - Flag* 742
  - Puffer* 770
  - Status* 742
  - String-* 763
  - synchronisieren* 741
  - verbinden* 741
- streambuf* 764, 770
- Streams 54
  - Standard-* 722
- String 159
  - Bibliothek-* 695
  - C-String* 706
  - Datentypen* 697
  - einlesen* 721
  - Elementzugriff* 702
  - erzeugen* 698
  - Exception-Handling* 697
  - Kapazität ändern* 703
  - Konstruktoren* 698
  - konvertieren* 706
  - Länge ermitteln* 703
- String (Forts.)
  - Manipulation* 707
  - Streams* 763
  - suchen in* 710
  - Überladene Operatoren* 719
  - vergleichen* 717
  - zeilenweise Einlesen* 758
  - Zuweisung* 700
- string
  - Klasse* 695
- string.h* 162
- stringbuf* 764, 775
- String-Streams 763, 764
- strlen()* 163
- strncat()* 162
- strncmp()* 165
- strncpy()* 161
- Stroustrup 25
- strrchr()* 164
- strstr()* 165
- strstream* 764
- strtok()* 165
- struct 198
  - class* 267
- Strukturen 198
  - Array* 204
  - Bit-Felder* 216
  - deklarieren* 198
  - dynamisch* 210
  - eingebettet* 208
  - gepackt* 216
  - Parameterübergabe an Funktionen* 201
  - union* 218
  - Variante* 218
  - vergleichen* 203
  - verkettete Liste* 210
  - Zugriff* 199
- substr()* 709
- Suche
  - Algorithmen (STL)* 624
  - String* 710
- sum()* 782
- sungetc()* 772
- swap()* 536, 542, 553, 568, 576, 600, 709
- swap\_ranges()* 600
- switch 83
  - case* 83
  - default* 84

## T

---

tan() 778, 787, 803  
 Tangens 803  
 tanh() 778, 803  
 TCP 921  
 Teilvektoren 788  
 tellg() 760  
 tellp() 760  
 Template  
     *Funktions-* 477  
     *Klasse* 489  
     *STL* 507  
     *verkettete Liste* 490  
 terminate() 663, 691  
 Testen 890  
 this 305  
     *\*this* 306, 312  
     *this->* 306, 312  
 throw 662  
     *longjmp* 835  
 tie() 741  
 top() 555, 562  
 transform() 602  
 true 40  
 try 663  
     *verschachteln* 670  
 Typdefinition  
     *typedef* 223  
 type\_info 814  
 typeid() 815  
 typeinfo 814  
 Typerkennung zur Laufzeit 814  
 Typqualifikatoren 253  
     *const* 254  
     *volatile* 254  
 Typumwandlung 255  
     *abgeleitete Klasse* 410  
     *Bool-* 259  
     *C-Casts* 260  
     *const\_cast<>* 262  
     *dynamic\_cast<>* 264  
     *explizit* 260  
     *Gleitkomma-* 258  
     *Gleitkomma-Promotion* 257  
     *implizit* 256  
     *Integral-* 257  
     *Integral-Gleitkomma-* 258  
     *Integral-Promotion* 256

Typumwandlung (Forts.)  
     *Klassen* 259, 388  
     *Konvertierungsfunktion* 390  
     *Konvertierungskonstruktor* 388  
     *Operatoren* 261  
     *reinterpret\_cast<>* 263  
     *Standard* 256  
     *static\_cast<>* 263  
     *Zeiger* 259

## U

---

Überladen  
     - (*unärer Operator*) 371  
     -- (*unärer Operator*) 374  
     ! (*unärer Operator*) 372  
     () (*Funktionsoperator*) 385  
     ++ (*unärer Operator*) 374  
     += (*Operator*) 362  
     << (*Shift-Operator*) 381  
     = (*Zuweisungsoperator*) 376  
     >> (*Shift-Operator*) 381  
     *arithmetische Operatoren* 362  
     *Funktionen* 113  
     *Operatoren* 358  
     *unäre Operatoren* 371  
 UCHAR\_MAX 50  
 UDP 921  
 UINT\_MAX 50  
 ULONG\_MAX 50  
 UML 837  
     *Abhängigkeiten* 867  
     *abstrakte Klassen* 852  
     *Aggregationen* 859  
     *Assoziationen* 853  
     *Diagramme erstellen* 840  
     *Eigenschaften einer Klasse* 841  
     *Kardinalität* 856  
     *Klasse* 840  
     *Klassenattribut* 846  
     *Klassendiagramme* 840  
     *Komponenten* 839  
     *Komposita* 861  
     *Kontexte* 862  
     *Mehrfachvererbung* 851  
     *Methoden einer Klasse* 844  
     *Notizen machen* 848  
     *Objekt einer Klasse* 843  
     *Pakete* 863

- UML (Forts.)
  - qualifizierte Assoziation* 858
  - reflexive Assoziation* 858
  - Schnittstellen* 865
  - Sichtbarkeit* 845
  - spezielle Assoziationen* 858
  - Vererbung* 849
  - Wozu UML?* 837
- undef-Direktive 126
- underflow\_error 686
- unexpected() 689
- Unformatierte Ausgabe 736
  - flush()* 736
  - put()* 736
  - write()* 736
- Unformatierte Eingabe 738
  - gcount()* 739
  - get()* 738
  - getline()* 739
  - ignore()* 739
  - putback()* 739
  - read()* 739
  - readsome()* 739
  - unget()* 739
- unget() 739
- Unicode 44, 1222
- Unions 218
- unique() 542, 612
- unique\_copy() 612
- unsetf() 746
- unsigned 46
- upper\_bound() 567, 576, 625
- uppercase 726
- USHRT\_MAX 50
- using 233, 236, 241
  - namespace* 236
- utility 512
- value\_comp() 568, 576
- Variable 40
- Variablen
  - global* 109
  - lokal* 109
- Variante 218
- VCL (GUI-Bibliothek) 997
- vector (STL-Container) 533
  - Datentypen* 534
  - Methoden* 534
- Vektoren 152
- Vererbung 266, 392
  - dynamic\_cast<>* 439
  - public* 399
  - Signatur* 422
  - Zugriffsrechte* 399
- Vergleichsoperatoren 76
- Verkettete Liste 210, 441, 490, 543
  - Anker* 211
  - doppelt verkettet* 215
  - Ende* 212
  - Knoten* 211
- Verschmelzen
  - Algorithmen (STL)* 627
- Verzweigungen 69
  - case* 83
  - else* 69
  - else if* 73
  - if* 69
  - switch* 83
- virtual 417, 433
- Virtual Method Table (VMT) 426
- void 52
  - Zeiger* 148
- void\* 141
- volatile 254
- Vorrangtabelle 1209

## V

---

- valarray 779
  - gslice* 792
  - indirect\_array* 800
  - mask\_array* 797
  - mathematische Funktionen* 787
  - Methoden* 781
  - Operatoren* 784
  - slice* 789
  - Teilvektoren* 788

## W

---

- Wahlfreier Dateizugriff 760
- Wahrheitswert 40
- wchar\_t 43, 696
- what() 681
- while 88
- width() 734
- Win32-API 994
- Winsock 923
  - initialisieren* 925

- worst case 533
- write() 736, 758
- ws 738
- WSACleanup() 925
- WSAStartup() 925
- wstring 696
- wxApp 1003
- wxArray 1201
- wxBitmap 1196
- wxBitmapButton 1088
  - Ereignis-Handle einrichten* 1089
  - Style* 1089
- wxBookCtrlBase 1058
- wxBoxSizer 1047
- wxBusyInfo 1173
- wxButton 1081
  - Ereignis-Handle einrichten* 1083
  - Methoden* 1082
  - Stock-Button* 1084
  - Style* 1082
- wxCheckBox 1103
  - Ereignis-Handle einrichten* 1105
  - Methoden* 1104
  - Style* 1104
- wxCheckListBox 1107
  - Ereignis-Handle einrichten* 1110
  - Methoden* 1109, 1110
  - Style* 1108
- wxChoice 1091
  - Ereignis-Handle einrichten* 1094
  - Methoden* 1092
- wxChoicebook 1059
- wxClipboard 1198
- wxColourDialog 1182
- wxComboBox 1097
  - Ereignis-Handle einrichten* 1099
  - Methoden* 1098
  - Style* 1098
- wxCommandEvent 1010
- wxContextMenuEvent 1146
  - Ereignis-Handle einrichten* 1147
- wxControl 1026
- wxControlWithItems 1026
- wxCursor 1197
- wxDateSpan 1203
- wxDateTime 1202
- wxDC 1195
- wxDialog 1035
  - Methoden* 1036
- wxDialog (Forts.)
  - Style* 1036
- wxDir 1203
- wxDirDialog 1181
  - Style* 1182
- wxEVT\_DESTROY 1021
- wxEvtHandle 1012
- wxFFile 1203
- wxFile 1203
- wxFileDialog 1178
  - Methoden* 1180
  - Style* 1179
- wxFileName 1204
- wxFindReplaceDialog 1194
- wxFlexGridSizer 1049
- wxFontDialog 1184
- wxFrame 1003, 1005, 1027
  - Ereignis-Handle einrichten* 1028
  - Methoden* 1028
  - Style* 1006
- wxGauge 1136
- wxGrid 1199
- wxGridSizer 1048
- wxHashMap 1202
- wxHtmlWindow 1198
- wxIcon 1197
- wxImage 1197
- wxInputStream 1204
- wxList 1202
- wxListbook 1058
- wxListBox 1107
  - Ereignis-Handle einrichten* 1110
  - Methoden* 1109
  - Style* 1108
- wxListCtrl 1198
- wxLongLong 1203
- wxMDIChildFrame 1034
- wxMDIParentFrame 1034
- wxMenu 1140
  - Ereignis-Handle einrichten* 1146
  - Methoden* 1141
  - Tastenkombinationen* 1145
- wxMenuBar 1147
  - Methoden* 1148
- wxMessageDialog 1035, 1040
  - Style* 1040
- wxMultiChoiceDialog 1187
- wxNotebook 1051
  - Methoden* 1052

- wxNotebook (Forts.)
  - Style* 1052
- wxNotebookSizer 1049
- wxNumberEntryDialog 1191
- wxObjArray 1202
- wxObject 1203
- wxOutputStream 1204
- wxPageSetupDialog 1194
- wxPanel 1007, 1050
- wxPasswordEntryDialog 1194
- wxPoint 1203
- wxPopupWindow 1050
- wxPrintDialog 1194
- wxProgressDialog 1171
  - Methoden* 1173
  - Style* 1172
- wxRadioButton 1113
  - Ereignis-Handle einrichten* 1115
  - Methoden* 1114
  - Style* 1114
- wxRadioButton 1113
  - Ereignis-Handle einrichten* 1117
  - Methoden* 1116
  - Style* 1116
- wxRect 1203
- wxRegex 1201
- wxScrolledWindow 1060
  - Ereignis-Handle einrichten* 1061
  - Methoden* 1061
- wxShowTip 1174
- wxSingleChoiceDialog 1186
  - Style* 1187
- wxSize 1203
- wxSizer 1045
  - Flags für Ausrichtung* 1046
  - Flags für Rahmen* 1046
- wxSlider 1119
  - Ereignis-Handle einrichten* 1123
  - Methoden* 1122
  - Style* 1122
- wxSocket 1205
- wxSortedArray 1202
- wxSpinCtrl 1119
  - Ereignis-Handle einrichten* 1121
  - Methoden* 1120
  - Style* 1120
- wxSplitterWindow 1066
  - Ereignis-Handle einrichten* 1068
  - Style* 1067
- wxStaticBitmap 1137
  - Methoden* 1138
- wxStaticBox 1139
- wxStaticBoxSizer 1049
- wxStaticLine 1138
- wxStaticText 1136
  - Methoden* 1137
  - Style* 1137
- wxStreamBase 1204
- wxString 1201
- wxStringTokenizer 1201
- wxTaskBarIcon 1199
- wxTempFile 1203
- wxTextCtrl 1125
  - Ereignis-Handle einrichten* 1131
  - Methoden* 1127
  - Style* 1126
- wxTextEntryDialog 1192
- wxTextFile 1203
- wxThread 1205
- wxTimeSpan 1203
- wxToggleButton 1133
  - Ereignis-Handle einrichten* 1134
  - Methoden* 1133
- wxToolBar 1149
  - Ereignis-Handle einrichten* 1154
  - Methoden* 1151
- wxToolBar
  - Style* 1150
- wxToolbook 1060
- wxTopLevelWindow
  - Methoden* 1029
- wxTreebook 1060
- wxTreeCtrl 1198
- wxUpdateUIEvent 1146
  - Ereignis-Handle einrichten* 1146
- wxWidgets 1001
  - Container-Fenster* 1050
  - Dateiklassen* 1203
  - Datenstrukturen* 1201
  - Device-Context* 1195
  - Dialoge* 1171
  - Dialoge für Dateien* 1178
  - Dialoge für Verzeichnisse* 1178
  - Dynamischer Event-Handle* 1017
  - Ereignis-Handle* 1012
  - Ereignisse behandeln* 1009
  - Ereignisse überspringen* 1017
  - Fenster* 1019



wxWidgets (Forts.)  
*Hallo Welt* 1002  
*Kontrollbalken* 1147  
*Kontrollelemente* 1077  
*Kontrollelemente (statisch)* 1135  
*Maus-Eingabe* 1195  
*Menüs* 1140  
*MFC* 1206  
*Stream-Klassen* 1204  
*Tastatur-Eingabe* 1195  
*Text-Editor (Beispiel)* 1156  
*Top-Level-Fenster* 1020  
*Übersicht* 1001  
*Verzeichnisklassen* 1203  
 wxWindows 1002  
*Zwischenablage* 1197  
 wxWindow 1004, 1005, 1020, 1023  
*Ereignis-Handle einrichten* 1023  
*Methoden* 1048, 1087  
*PopupMenu* 1154  
 wxWindows 1002  
 wxWizard 1198

## X

---

Xlib 994

## Z

---

Zahlen  
*Grenzwerte* 806  
*komplexe* 776  
 Zahlensysteme 1211  
*dezimal nach dual* 1215  
*dezimal nach hexadezimal* 1216  
*Dezimalsystem* 1212  
*dual nach dezimal* 1216  
*Dualsystem* 1212  
*hexadezimal nach dezimal* 1217  
*hexadezimal nach dual* 1217  
*Hexadezimalsystem* 1214  
*oktal nach dual* 1217  
*Oktalsystem* 1213  
*umrechnen* 1215  
 Zeichen 37  
*Einlesen* 58

Zeichenauswahl 896  
 Zeichencodierung 44  
 Zeichenketten 37, 159  
*einlesen* 58  
 Zeichenklassen 896  
 Zeichen-Literale 896  
 Zeichensätze 44, 1218  
*ASCII* 1219  
*ASCII-Erweiterungen* 1220  
*Unicode* 1222  
 Zeichenweise Ausgabe 754  
 Zeichenweise Eingabe 754  
 Zeiger 133  
*Adressen speichern* 135  
*Adresszuweisung* 135  
*Arithmetik* 140  
*Array* 166  
*auf andere Zeiger verweisen* 141  
*auf Klasseigenschaften* 352  
*const* 148  
*C-String* 171  
*deklarieren* 134  
*delete* 146  
*dereferenzieren* 137  
*dynamisch Speicher anlegen* 143  
*Elementzeiger* 352  
*Funktionsparameter* 182  
*konstante* 148  
*mehrfache Indirektion* 171  
*new* 144  
*Referenzen* 150  
*Rückgabewert* 190  
*Typumwandlung* 259  
*verkettete Liste* 210  
*virtueller Methoden-* 426  
*void* 148  
 Zeigerarithmetik 140  
 Zeiger-Typumwandlung 259  
 Zeilenweise Ausgabe 756  
 Zeilenweise Eingabe 756  
 Zugriffsmethoden 299  
 Zugriffsrechte 407  
*abgeleitete Klasse* 407  
*Klasse* 272