Informatik & Praxis

Jürgen Dankert
C++ für C-Programmierer

Informatik & Praxis

Herausgegeben von

Prof. Dr. Helmut Eirund, Fachhochschule Harz

Prof. Dr. Herbert Kopp, Fachhochschule Regensburg

Prof. Dr. Axel Viereck, Hochschule Bremen

Anwendungsorientiertes Informatik-Wissen ist heute in vielen Arbeitszusammenhängen nötig, um in konkreten Problemstellungen Lösungsansätze erarbeiten und umsetzen zu können. In den Ausbildungsgängen an Universitäten und vor allem an Fachhochschulen wurde dieser Entwicklung durch eine Integration von Informatik-Inhalten in sozial-, wirtschafts- und ingenieurwissenschaftliche Studiengänge und durch Bildung neuer Studiengänge – z.B. Wirtschaftsinformatik, Ingenieurinformatik oder Medieninformatik – Rechnung getragen.

Die Bände der Reihe wenden sich insbesondere an die Studierenden in diesen Studiengängen, aber auch an Studierende der Informatik, und stellen Informatik-Themen didaktisch durchdacht, anschaulich und ohne zu großen „Theorie-Ballast“ vor.

Die Bände der Reihe richten sich aber gleichermaßen an den Praktiker im Betrieb und sollen ihn in die Lage versetzen, sich selbständig in ein in seinem Arbeitszusammenhang relevantes Informatik-Thema einzuarbeiten, grundlegende Konzepte zu verstehen, geeignete Methoden anzuwenden und Werkzeuge einzusetzen, um eine seiner Problemstellung angemessene Lösung zu erreichen.

C++ für C-Programmierer

Von Prof. Dr.-Ing. habil. Jürgen Dankert
Fachhochschule Hamburg

B.G.Teubner Stuttgart · Leipzig 1998

Prof. Dr.-Ing. habil. Jürgen Dankert

Geboren 1941, von 1961 bis 1966 Studium des Maschinenbaus an der Technischen Hochschule Magdeburg, 1971 Promotion zum Dr.-Ing., 1979 Habilitation. Von 1984 bis 1987 Mitarbeiter in der Firma Hewlett-Packard in Böblingen, von 1987 bis 1990 Professor für Technische Mechanik an der Fachhochschule Frankfurt/Main, seit 1990 Professor für Mathematik und Informatik an der Fachhochschule Hamburg.

Alle in diesem Buch erwähnten Firmen- und Produktnamen wie Borland, DOS, IBM, Java, Linux, Microsoft, MS-Visual-C++, Borland-C++, UNIX, Windows 3.1, Windows 95, Windows NT u.a. sind gesetzlich geschützt, ohne daß im einzelnen darauf hingewiesen wird.

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Dankert, Jürgen:

C++ für C-Programmierer / von Jürgen Dankert. – Stuttgart ; Leipzig : Teubner, 1998

(Informatik & Praxis)

ISBN 978-3-519-02641-9

ISBN 978-3-322-92730-9 (eBook)

DOI 10.1007/978-3-322-92730-9

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Das gilt besonders für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

© 1998 B.G.Teubner Stuttgart · Leipzig

Vorwort

Immer komplexere Probleme zwingen geradezu zu objektorientiertem Denken. Daß mit den objektorientierten Programmiersprachen genau die Hilfsmittel bereitgestellt werden, mit denen die reale Welt adäquat abgebildet werden kann, ist sicherlich die Basis für den Trend, neue Software-Projekte fast ausschließlich damit zu realisieren.

Die noch recht junge Programmiersprache C++ nimmt dabei zweifellos eine Sonderstellung ein, weil die sehr weit verbreitete Programmiersprache C in ihr komplett enthalten ist. Das erleichtert vielen Programmierern den Umstieg erheblich, und an diese Umsteiger wendet sich dieses Buch. Die vielfach zu hörende Skepsis, daß Umsteiger (wegen der Gefahr, "rückfällig zu werden") besondere Schwierigkeiten haben, die objektorientierte Denkweise konsequent zu verfolgen, wird von mir nicht geteilt, im Gegenteil: Gerade der (und nun nenne ich ihn) Aufsteiger zur objektorientierten Programmierung wird die Vorteile ganz besonders deutlich erkennen.

Nicht unterschätzt werden darf die Möglichkeit, bewährte Algorithmen der Programmiersprache C, die ohne Änderung auch von C++-Compilern übersetzt werden können, mit einer Verpackung zu versehen, die eine "objektorientierte Weiterverwendung" gestatten (die ab Kapitel 7 verwendeten "Microsoft foundation classes" sind geradezu ein klassisches Beispiel dafür). Dies legt einen "Aufstieg zu C++" ebenso nahe wie der nicht zu übersehende Trend, daß die Hersteller modernen Entwicklungsumgebungen für die Programmentwicklung konsequent die objektorientierten Sprachen unterstützen. Diese Aussage gilt in besonderem Maße für die Windows-Programmierung.

Das Erlernen der objektorientierten Programmierung gilt als schwierig. Ich kann aus eigener Erfahrung in der Lehre bestätigen, daß der Anfänger tatsächlich mehr Schwierigkeiten als mit anderen Programmiersprachen hat. Aber nach einer gewissen "Durststrecke" zahlen sich die Mühen aus. Voraussetzung ist allerdings, daß man nicht in erster Linie die syntaktischen Regeln der Programmiersprache erlernt, sondern die Strategie des objektorientierten Denkens in objektorientierte Programme umsetzt. Dieses Buch versucht, dies konsequent zu unterstützen (C++ gestattet - siehe oben - durchaus auch das Gegenteil).

Schließlich ist der Erfolg beim Erlernen einer Programmiersprache weitgehend auch vom Spaß abhängig, den man bei aller Mühe unbedingt haben sollte. Das schönste Ergebnis, ein funktionierendes Programm aus eigener Fertigung, sollte sich allerdings möglichst auch "so schön" präsentieren, wie es die professionell erzeugte Software tut. Dazu sind Kenntnisse der Windows-Programmierung heute unerlässlich.

Das vorliegende Buch ist aus Skripten entstanden, die ich meinen Studenten als Begleitmaterial für Vorlesung und Praktika zur Verfügung stelle. Das darin verfolgte Prinzip, nicht streng themengebunden vorzugehen, sondern an Programm-Beispielen nach und nach alle wichtigen Probleme abzuhandeln, hat sich auch für das Selbststudium bewährt. Für die gezielte Suche nach speziellen Themen ist deshalb das Sachverzeichnis manchmal hilfreicher als das Inhaltsverzeichnis, es ist deshalb besonders umfangreich.

Die Beispiel-Programme, die im Buch abgedruckt sind und unbedingt vom Lernenden "mit dem Computer nachempfunden" werden sollten, brauchen nicht abgetippt zu werden, sie sind über die im Abschnitt 1.2 angegebene Internet-Adresse verfügbar (oder über den auf Seite 332 beschriebenen Weg zu beziehen).

Das Programmieren kann man nur erlernen, indem man programmiert. Um möglichst keine Einstiegshürden aufzubauen, habe ich bewußt auf spezielle Hilfsmittel zur Sprachbeschreibung verzichtet. Es werden weder eine "Metasprache" noch die in den letzten Jahren entwickelte Notation objektorientierter Modelle benutzt (mit dem Ziel einer Standardisierung, Stichwort UML, "Unified modeling language"). Die Erfahrungen in der Lehre haben mich sogar ermutigt, manche Programm-Konstruktionen zunächst nur eingeschränkt zu erläutern (zugunsten der Verständlichkeit), um erst später dem fortgeschrittenen Leser die gesamte Information darüber zuzumuten.

Bedanken möchte ich mich bei Herrn Prof. Dr. Kopp von der FH Regensburg für fachliche Hinweise und Herrn Dr. Spuhler, der das Erscheinen des Buchs im Teubner-Verlag ermöglichte, und natürlich bei meiner Frau Helga, die mir allerhand störende Dinge vom Hals gehalten und für mein leibliches Wohl gesorgt hat. Das höchst undankbare Geschäft des Korrekturlesens, das sie für die vor einem Jahr erschienene "Praxis der C-Programmierung" erledigte, hat diesmal Herr Dr. Wolf Dorn übernommen, dem wir beide dafür herzlich danken.

Jesteburg, Juli 1998

Jürgen Dankert

e-mail: dankert@rzbt.fh-hamburg.de
oder 100430.3712@compuserve.com

Homepage:
<http://www.fh-hamburg.de/rzbt/dankert>

Inhalt

1	Programmiersprachen C und C++	11
1.1	Objektorientierte Programmierung	11
1.2	Hilfsmittel für die C++-Programmierung	13
1.3	Das erste C++-Programm, natürlich: "Hello, World"	14
1.4	C++-Erweiterungen, die nicht direkt "objektorientiert" sind	16
1.4.1	Kommentare, Variablen-Definitionen, "Casts"	16
1.4.2	"Call by reference", "Reference return values", inline-Funktionen	17
1.4.3	Dynamische Speicherplatz-Verwaltung	23
2	Klassen und Objekte	26
2.1	Daten und Member-Funktionen in der Klasse	27
2.2	Konstruktoren, Destruktoren, Default-Argumente	31
2.3	Objektorientiertes Programmieren erfordert objektorientiertes Denken	36
2.3.1	Objekte	36
2.3.2	Aktion auch bei leerer Funktion main	37
2.4	Eine String-Klasse	40
2.5	CString-Objekte "wachsen"	43
2.6	Kritik an der Klasse CString	45
3	Überladen	47
3.1	Überladen von Funktionen	47
3.2	Erweitern der Klasse CString	51
3.3	Konstruktoren sind immer überladen	54
3.3.1	Der Copy-Konstruktor	55
3.3.2	Ein Copy-Konstruktor für die Klasse CString	56
3.4	Überladen von Operatoren	58
3.4.1	Die Technik des Überladens von Operatoren	60
3.4.2	Der this-Pointer, das Überladen des Zuweisungsoperators	61
3.4.3	Typ-Konvertierung	64
3.4.3	Konvertierungsoperatoren	67
3.4.5	Noch einmal: Kritik an der Klasse CString	69
3.4.6	Überladen des Additionsoperators, friend-Funktionen	70
3.4.7	Eine "mathematische Klasse": CVector	75
3.4.8	"Prefix-Postfix"-Operatoren	79
3.4.9	Das Klassen-Objekt cout	81
4	Komposition und Vererbung	83
4.1	Zusammengesetzte ebene Flächen, Problemdiskussion	83
4.2	Komposition	86
4.3	Öffentliche Vererbung	91
4.4	Statische Daten-Elemente, statische Member-Funktionen	96
4.5	Vererbung und Konvertierung	101
5	Polymorphismus	106
5.1	Virtuelle Member-Funktionen, späte Bindung	107
5.2	Entwurf einer Klassen-Hierarchie	110
5.2.1	Redundanz in den Daten-Elementen einer Klasse	111

5.2.2	Rein virtuelle Member-Funktionen, abstrakte Klassen	113
5.2.3	Eine erste Version der Klassen-Hierarchie	115
5.3	Polymorphismus, Schlüssel zur objektorientierten Programmierung	120
5.3.1	Erzeugen einer Klassen-Bibliothek	120
5.3.2	Programm-Erweiterung: Reales Objekt "Dreieck" und Klasse CITriangle	121
5.4	Der Sinn eines virtuellen Destruktors	125
5.4.1	Erlaubte und unerlaubte Verwendung des Schlüsselworts virtual	125
5.4.2	Polygonflächen, die Klasse CIPolygon	126
5.4.3	Wie vorsichtig sollte man eigentlich sein?	130
5.5	Kritik an der Klassen-Hierarchie	131
5.5.1	Die reduzierte Basisklasse und die daraus abgeleiteten Klassen	132
5.5.2	Die geänderten Eingabe-Funktionen	133
5.5.3	Eine "Listen- und Stack-Klasse"	134
5.5.4	Eine Klasse für die Gesamtfläche	138
5.5.5	Programm sp8.cpp mit der überarbeiteten Klassen-Hierarchie	140
6	Ergänzende und spezielle Themen	143
6.1	Das Schlüsselwort const	143
6.1.1	Konstanten-Definitionen	143
6.1.2	Pointer und das Schlüsselwort const	147
6.1.3	Konstante Member-Funktionen	148
6.2	Templates	155
6.2.1	Funktions-Templates	155
6.2.2	Klassen-Templates	159
6.2.3	Verallgemeinerungen, allgemeine Betrachtungen zu Templates	163
6.3	Ausnahmebehandlung	167
6.3.1	Nichterfüllbare Speicherplatzanforderung	168
6.3.2	Ausnahmebehandlung mit try, catch und throw	170
6.4	Mehrfach-Vererbung	175
6.5	Private Vererbung	180
6.6	Arbeiten mit Dateien	182
6.6.1	Datei öffnen, Manövrieren in der Datei, Datei schließen	182
6.6.2	Lesen und Schreiben von Text-Dateien	184
6.6.3	Schreiben einer Text-Datei im Programm sp8asc.cpp	186
6.6.4	Lesen einer Text-Datei im Programm sp8asc.cpp	188
6.6.5	ASCII-Dateien oder Binär-Dateien?	193
6.6.6	Schreiben und Lesen einer Binär-Datei im Programm sp8bin.cpp	194
6.7	RTTI, Typ-Ermittlung zur Laufzeit	202
6.8	Namensbereiche	205
6.9	Objektorientierte Programmierung, eine kurze Zusammenfassung	207
7	Windows-Programmierung mit MFC	208
7.1	Besonderheiten der Windows-Programmierung	208
7.2	Windows-C++-Programmierung mit "Microsoft foundation classes"	210
7.3	Das minimale Programmgerüst eines MFC-Programms	211
7.4	Bearbeiten von Botschaften, Beispiel: WM_PAINT	214
7.4.1	"Message maps"	214
7.4.2	Der "Device context"	217
7.4.3	Farben, GDI-Objekte, zeichnende CDC-Funktionen	219

7.5	Koordinatensysteme	223
7.5.1	CIGI, eine Klasse für spezielle Koordinaten	225
7.5.2	Programm sp8draw.cpp, Zeichnen der ebenen Flächen	232
7.6	Ressourcen	237
7.6.1	Ein Menü für das Programm funct1.cpp	237
7.6.2	Bearbeiten der WM_COMMAND-Botschaft	242
7.6.3	"Command handlers" und "Update handlers"	244
7.6.4	Erzeugen und Einbinden einer Dialog-Box	247
7.7	Zusammenfassung, Ausblick	255
8	Arbeiten mit dem MFC-Anwendungsgerüst	259
8.1	Ein letztes Mal in diesem Buch: "Hello, World!"	259
8.2	Die "Document-View"-Architektur	265
8.3	Das Projekt Fmom	267
8.3.1	Die mit Fmom zu realisierende Funktionalität	267
8.3.2	Erzeugen des Projekts (Version Fmom1)	268
8.4	Die Klassen-Hierarchie des Projekts Fmom	270
8.4.1	Problembezogene Klassen-Hierarchie	270
8.4.2	Verankerung in der Dokumentklasse, die Klasse CObList	272
8.5	Eingabe der Daten	276
8.5.1	Bearbeiten des Menüs	276
8.5.2	Dialog-Boxen und Dialogklassen	277
8.5.3	Einbinden der Dialoge in das Programm	283
8.6	Bearbeiten der Ansichtsklasse, Ausgabe erster Ergebnisse	286
8.6.1	Vorbereitung der Ausgabe in der Ansichtsklasse	286
8.6.2	Textausgabe mit CDC::TextOut, die Struktur TEXTMETRIC	287
8.7	Verbesserung der Eingabe	291
8.7.1	Die Return-Taste muß Kompetenzen abgeben	291
8.7.2	"Toolbar-Buttons" löschen und hinzufügen	293
8.8	Das Dokument als Binär-Datei, "Serialization"	295
8.8.1	Eine Klasse für die "Serialization" vorbereiten	295
8.8.2	"Serialization" für die Klassen CCircle und CRectangle	298
8.8.3	Komplettierung der "Serialization", die Klasse CPoint	301
8.9	Eine zweite Ansicht für das Dokument, "Splitter-Windows"	304
8.9.1	"Splitter-Windows" erzeugen	304
8.9.2	Vorbereiten einer zweiten Ansicht	308
8.10	Graphische Darstellung der Flächen	309
8.10.1	Auslösen der Zeichenaktionen in CDrawView::OnDraw	309
8.10.2	Realisieren der Zeichenaktionen mit CFmomDoc-Funktionen	310
8.10.3	Ein "Marker" für den Schwerpunkt, Durchmesser: 0,1 "Logical Inches"	312
8.10.4	Der (oft vergebliche) Versuch, "pixelgenau" zu zeichnen	314
8.11	Listen, Ändern, Löschen	315
8.11.1	Anpassen des Menüs, "Accelerators"	315
8.11.2	"Alle löschen" und "Letzte löschen"	316
8.11.3	Dialog-Box mit Listenfeld	318
8.11.4	Initialisieren des Listenfeldes	321
8.11.5	Ändern bzw. Löschen einer ausgewählten Fläche	328
8.12	Zusammenfassung, Ausblick	330
	Literatur	332
	Sachverzeichnis	333

Verwendung von Bezeichnern, Schreibweise

Es erleichtert die Lesbarkeit von Programmen erheblich, wenn für die Bezeichner gewisse Regeln bei der Namensbildung eingehalten werden. Um bei der unvermeidlichen Vermischung von Bezeichnern, die in kommerziell vertriebenen Klassen-Bibliotheken verwendet werden, mit den selbst gewählten Namen die "Herkunft" erkennbar zu machen, werden in diesem Buch folgende Regeln eingehalten:

- ◆ Bezeichner für Klassen beginnen mit **Cl**, gefolgt von einem Groß-Buchstaben (Microsofts MFC-Klassen-Namen beginnen mit **C**, Borlands OWL-Klassen-Namen mit **T**, jeweils auch mit einem nachfolgenden Groß-Buchstaben).
- ◆ Namen von Pointern enden auf **_p** (Microsoft: Pointer-Namen beginnen mit **p**, Borland sieht keine Besonderheit für die Bezeichnung von Pointern vor). Namen von Member-Variablen beginnen (wie bei Microsoft) immer mit **m_**.
- ◆ "Sprechende" (und damit zwangsläufig längere) Namen von Funktionen werden bevorzugt mit Unterstrichen leserlich gemacht (z. B.: **insert_new_area**, bei Microsoft und Borland werden Groß-Buchstaben dafür verwendet, z. B.: **GetHorizontalExtent**).
- ◆ Der "Null-Pointer" hat in C++ (im Gegensatz zu C) tatsächlich den Wert **0**. Trotzdem wird die symbolische Konstante **NULL** verwendet, wodurch sich die Lesbarkeit der Programme sicher verbessert.

Zur Stellung des Dereferenzierungssymbols ***** gibt es keine einheitlichen Regeln. Die "C-Väter" Kernighan und Ritchie ordneten den Stern dem Namen zu (z. B.: **int *w1_p**, "weil ***w1_p** ein **int** ist, muß **w1_p** ein Pointer sein"). Es gibt ebenso gute (didaktische) Gründe, den Stern der Typ-Bezeichnung zuzuordnen (z. B.: **int* w1_p**, "**w1_p** ist eine Variable vom Typ **int***, also eine Pointer-Variable"). Stroustrup verwendet in [Stro94] diese Schreibweise. Entsprechende Aussagen gelten für Pointer-Return-Werte und für die Stellung des Symbols **&**, wenn es eine Referenz-Variable (oder einen Referenz-Return-Wert) kennzeichnet.

Dies wird deshalb erwähnt, weil die unterschiedlichen Schreibweisen nach den Erfahrungen des Autors bei Anfängern erhebliche Konfusion auslösen können, denn im Kapitel 8 dieses Buches muß zwangsläufig die bis dahin einheitlich gehandhabte Schreibweise verlassen werden. Also bitte aus den Leerzeichen vor und hinter dem Stern keine Information herauslesen ("Whitespace hat hier keine Bedeutung"), der Compiler tut es auch nicht.

Die genannte Inkonsequenz im Kapitel 8 wird von MS-Visual-C++ erzwungen. Während in der mit der Software gelieferten "Einführung in C++" konsequent der Stern beim Namen steht, wird er in den "Reference manuals" dem Typ zugeordnet. Der "Applikations-Assistent" hält sich an die Schreibweise in den Manuals, und der Klassen-Assistent kann sich für keine der beiden Varianten entscheiden und fügt (z. B.: **int * w1_p**) vor und nach dem Stern Leerzeichen ein (unabhängig davon, was der Programmierer ihm über den Dialog anbietet). In den von den Assistenten erzeugten Programmen wurde die Schreibweise nicht verändert.

In den Kapiteln 1 bis 6, in denen keine "echten Windows-Programme" erzeugt werden, wird zur Vereinfachung **void main** verwendet (entspricht der Praxis der Manuals von MS-Visual-C++), um das **return** zu sparen. Dies wird von den gängigen Compilern (z. B.: MS, Borland, GNU) akzeptiert, bei einer **return**-Anweisung ohne Argument erzeugt der GNU-C++-Compiler eine Warnung, die ignoriert werden kann (wenn sie als störend empfunden wird, muß man zu **int main** mit entsprechenden **return**-Anweisungen zurückkehren).