

Xpert.press

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Richard Kaiser

C++ mit dem Borland C++Builder 2007

Einführung in den C++-Standard
und die objektorientierte
Windows-Programmierung

2. überarbeitete Auflage
Mit CD-ROM

 Springer

Prof. Richard Kaiser

Schwärzlocher Str. 53
72070 Tübingen
www.rkaiser.de

ISBN 978-3-540-69575-2

e-ISBN 978-3-540-69773-2

DOI 10.1007/978-3-540-69773-2

ISSN 1439-5428

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen
Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über
<http://dnb.d-nb.de> abrufbar.

© 2002, 2008 Springer-Verlag Berlin Heidelberg

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.

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. Text und Abbildungen wurden mit größter Sorgfalt erarbeitet. Verlag und Autor können jedoch für eventuell verbliebene fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Springer ist nicht Urheber der Daten und Programme. Weder Springer noch der Autor übernehmen die Haftung für die CD-ROM und das Buch, einschließlich ihrer Qualität, Handels- und Anwendungseignung. In keinem Fall übernehmen Springer oder der Autor Haftung für direkte, indirekte, zufällige oder Folgeschäden, die sich aus der Nutzung der CD-ROM oder des Buches ergeben.

Einbandgestaltung: KünkelLopka Werbeagentur, Heidelberg

Gedruckt auf säurefreiem Papier

9 8 7 6 5 4 3 2 1

springer.com

Für Ruth

Geleitwort

Das Programmieren unter C++ gilt als die Königsklasse der objektorientierten Applikations-Entwicklung: Anwender nutzen C++, um universell einsetzbare, modulare Programme zu erstellen. Wer diese Sprache beherrscht, profitiert von einem beispiellosen Funktionsumfang und von der Option, plattformunabhängig zu arbeiten. Das war anfangs nur hochgradig versierten Profis vorbehalten. Sie allein waren in der Lage, der Komplexität des C++-Quellcodes Herr zu werden.

Längst aber stehen die Vorzüge von C++ auch all jenen zur Verfügung, die nur gelegentlich oder schlicht und ergreifend aus Freude am Tüfteln Applikationen erstellen. Einen wesentlichen Beitrag zur „Demokratisierung“ der objektorientierten Programmierung leisten integrierte RAD-Systeme (Rapid Application Development) wie der C++Builder von Borland.

Ganz gleich ob Profi oder Einsteiger: Die C++-Version der erfolgreichen Object Pascal-Lösung *Borland Delphi* bietet Programmierern eine visuelle Entwicklungsumgebung, mit der sie einfach und rasch objektorientierte Windows-Applikationen schreiben können. Der C++Builder verfügt über eine umfangreiche Palette an fertigen Komponenten und erleichtert seit der Version 5 auch die Entwicklung von Web-Applikationen. Wer grafische Benutzeroberflächen bauen will, stellt diese einfach mit wenigen Handgriffen per Maus zusammen. Das ist die Basis für ein schnelles, effizientes und komfortables Arbeiten. Kurzum: Mit dem C++Builder wird die Applikations-Entwicklung von der langwierigen Fleißaufgabe zur zielorientierten Kopfarbeit.

Das vorliegende Buch ist eine systematische Einführung in die Arbeit mit C++ und dem Borland C++Builder. Ausführlich und praxisnah schildert Richard Kaiser die Konzepte und Elemente der Programmiersprache und der Entwicklungsumgebung. Mit zahlreichen Beispielen und Übungsaufgaben erschließt er auch Lesern ohne Vorkenntnisse die Logik objektorientierten Programmierens.

Borland wünscht allen Nutzern dieses hervorragenden Lehrbuchs und Nachschlagewerks viel Spaß und Erfolg bei der Arbeit mit dem C++Builder.

Jason Vokes

European Product Line Manager – RAD Products and InterBase

Vorwort zur 2. Auflage

Nach nunmehr fünf Jahren liegt jetzt die zweite Auflage des „Builder-Buches“ vor. In dieser Zeit habe ich zahlreiche Vorlesungen und Industrieseminare auf der Basis der ersten Auflage gehalten. Dabei ergaben sich immer wieder Ansatzpunkte für Verbesserungen, Fehlerkorrekturen, Präzisierungen, Erweiterungen und Straffungen des Textes. Das Ergebnis ist eine komplette Überarbeitung der ersten Auflage.

Folgende Themen wurden für die zweite Auflage zusätzlich aufgenommen bzw. grundlegend erweitert:

- Änderungen gegenüber dem C++Builder 5
- systematische Tests und Unit Tests (Abschnitt 3.5)
- Programmierlogik und die Programmverifikation (Abschnitt 3.7)
- Objektorientierte Analyse und Design (Kapitel 6)

Die meisten Ausführungen gelten für den **C++Builder 2007** ebenso wie für den C++Builder 2006 oder noch ältere Versionen (C++Builder 5 und 6). Dabei ist es auch unerheblich, dass **Borland** den C++Builder inzwischen in eine eigene Firma mit dem Namen **CodeGear** ausgelagert hat. Da sich auch die meisten Screenshots und Verweise (z.B. unter *Projekt/Optionen*) bei den verschiedenen Versionen nur wenig unterscheiden, wurden nur die wichtigsten auf den kurz vor der Fertigstellung des Buches erschienenen C++Builder 2007 angepasst.

Tübingen, im August 2007

Richard Kaiser

Vorwort zur 1. Auflage

Dieses Buch entstand ursprünglich aus dem Wunsch, in meinen Vorlesungen über C++ nicht nur Textfensterprogramme, sondern Programme für eine grafische Benutzeroberfläche zu entwickeln. Mit dem C++Builder von Borland stand 1997 erstmals ein Entwicklungssystem zur Verfügung, das so einfach zu bedienen war, dass man es auch in Anfängervorlesungen einsetzen kann, ohne dabei Gefahr zu laufen, dass die Studenten nur noch mit dem Entwicklungssystem kämpfen und gar nicht mehr zum Programmieren kommen.

Angesichts der damals anstehenden Verabschiedung des ANSI/ISO-Standards von C++ lag es nahe, in diesem einführenden Lehrbuch auch gleich den gesamten Sprachumfang des Standards umfassend darzustellen. Mir war allerdings nicht klar, auf welche Arbeit ich mich damit eingelassen hatte. Ich hatte weder vor, vier Jahre an diesem Buch zu schreiben, noch einen Wälzer mit 1100 Seiten zu produzieren.

Als ich dann die Möglichkeit bekam, Kurse für erfahrene Praktiker aus der Industrie zu halten, wurde ich mit einer Fülle von Anregungen aus ihrer täglichen Arbeit konfrontiert. Diese gaben dem Buch enorme Impulse.

Die Programmiersprache C++ wurde als Obermenge der Programmiersprache C entworfen. Dieser Entscheidung verdankt C++ sicher seine weite Verbreitung. Sie hat aber auch dazu geführt, dass oft weiterhin wie in C programmiert wird und lediglich ein C++-Compiler anstelle eines C-Compilers verwendet wird. Dabei werden viele Vorteile von C++ verschenkt. Um nur einige zu nennen:

- In C++ werden die fehleranfälligen Zeiger viel seltener als in C benötigt.
- Die Stringklassen lassen sich wesentlich einfacher und risikoloser als die nullterminierten Strings von C verwenden.
- Die Containerklassen der C++-Standardbibliothek haben viele Vorteile gegenüber Arrays, selbstdefinierten verketteten Listen oder Bäumen.
- Exception-Handling bietet eine einfache Möglichkeit, auf Fehler zu reagieren.
- Objektorientierte Programmierung ermöglicht übersichtlichere Programme.
- Templates sind die Basis für eine außerordentlich vielseitige Standardbibliothek.

Ich habe versucht, bei allen Konzepten nicht nur die Sprachelemente und ihre Syntax zu beschreiben, sondern auch Kriterien dafür anzugeben, wann und wie man sie sinnvoll einsetzen kann. Deshalb wurde z.B. mit der objektorientierten Programmierung eine Einführung in die objektorientierte Analyse und das objektorientierte Design verbunden. Ohne die Beachtung von Design-Regeln schreibt man leicht Klassen, die der Compiler zwar übersetzen kann, die aber kaum hilfreich sind.

Man hört immer wieder die Meinung, dass C++ viel zu schwierig ist, um es als einführende Programmiersprache einzusetzen. Dieses Buch soll ein in mehreren Jahren erprobtes Gegenargument zu dieser Meinung sein. Damit will ich aber die Komplexität von C++ überhaupt nicht abstreiten.

Zahlreiche Übungsaufgaben geben dem Leser die Möglichkeit, die Inhalte praktisch anzuwenden und so zu vertiefen. Da man Programmieren nur lernt, indem man es tut, möchte ich ausdrücklich dazu ermuntern, zumindest einen Teil der Aufgaben zu lösen und sich dann selbst neue Aufgaben zu stellen. Der Schwierigkeitsgrad der Aufgaben reicht von einfachen Wiederholungen des Textes bis zu kleinen Projektchen, die ein gewisses Maß an selbständiger Arbeit erfordern. Die Lösungen der meisten Aufgaben findet man auf der beiliegenden CD und auf meiner Internetseite <http://www.rkaiser.de>.

Anregungen, Korrekturhinweise und Verbesserungsvorschläge sind willkommen. Meine EMail-Adresse finden Sie auf meiner Internetseite.

Bei allen meinen Schulungskunden und ganz besonders bei Herrn Welsner und der Alcatel University der Alcatel SEL AG Stuttgart bedanke ich mich für die Möglichkeit, dieses Manuskript in zahlreichen Kursen mit erfahrenen Praktikern weiterzuentwickeln. Ohne die vielen Anregungen aus diesen Kursen hätte es weder diesen Praxisbezug noch diesen Umfang erreicht. Peter Schwalm hat große Teile des Manuskripts gelesen und es in vielen Diskussionen über diffizile Fragen mitgestaltet. Mein Sohn Alexander hat als perfekter Systembetreuer dafür gesorgt, dass die Rechner immer liefen und optimal installiert waren.

Die Unterstützung von Dr. Hans Wössner und seinem Team vom Springer-Verlag hätte nicht besser sein können. Seine Hilfsbereitschaft und seine überragende fachliche Kompetenz waren immer wieder beeindruckend. „Meiner“ Lektorin Ruth Abraham verdankt dieses Buch eine in sich geschlossene Form, die ich allein nicht geschafft hätte. Die technische Herstellung war bei Gabi Fischer in erfahrenen guten Händen. Herrn Engesser danke ich für die gute Zusammenarbeit beim Abschluss des Projekts.

Tübingen, im Oktober 2001

Richard Kaiser

Inhalt

1 Die Entwicklungsumgebung.....	1
1.1 Visuelle Programmierung: Ein erstes kleines Programm	1
1.2 Erste Schritte in C++.....	5
1.3 Der Quelltexteditor	7
1.4 Kontextmenüs und Symbolleisten (Toolbars).....	11
1.5 Projekte, Projektdateien und Projektoptionen.....	13
1.6 Einige Tipps zur Arbeit mit Projekten	16
1.7 Die Online-Hilfe	20
1.8 Projektgruppen und die Projektverwaltung Θ	22
1.9 Hilfsmittel zur Gestaltung von Formularen Θ	24
1.10 Packages und eigenständig ausführbare Programme Θ	25
1.11 Win32-API und Konsolen-Anwendungen Θ	27
1.11.1 Konsolen-Anwendungen Θ	27
1.11.2 Der Start des Compilers von der Kommandozeile Θ	28
1.11.3 Win32-API Anwendungen Θ	28
1.12 Windows-Programme und Units Θ	29
2 Komponenten für die Benutzeroberfläche	31
2.1 Die Online-Hilfe zu den Komponenten.....	31
2.2 Namen.....	35
2.3 Labels, Datentypen und Compiler-Fehlermeldungen.....	38
2.4 Funktionen, Methoden und die Komponente <i>TEdit</i>	43
2.5 Memos, ListBoxen, ComboBoxen und die Klasse <i>TStrings</i>	47
2.6 Buttons und Ereignisse.....	53
2.6.1 Parameter der Ereignisbehandlungsroutinen	54
2.6.2 Der Fokus und die Tabulatorreihenfolge	55
2.6.3 BitButtons und einige weitere Eigenschaften von Buttons	56
2.7 CheckBoxen, RadioButtons und einfache <i>if</i> -Anweisungen.....	58
2.8 Die Container GroupBox, Panel und PageControl.....	60
2.9 Hauptmenüs und Kontextmenüs.....	63

2.9.1	Hauptmenü und der Menüdesigner	64
2.9.2	Kontextmenü	65
2.9.3	Die Verwaltung von Bildern mit ImageList Θ	66
2.9.4	Menüvorlagen speichern und laden Θ	67
2.10	Standarddialoge	67
2.10.1	Einfache Meldungen mit <i>ShowMessage</i>	70

3 Elementare Datentypen und Anweisungen..... 73

3.1	Syntaxregeln	73
3.2	Variablen und Bezeichner	76
3.3	Ganzzahldatentypen	80
3.3.1	Die interne Darstellung von Ganzzahlwerten	83
3.3.2	Ganzzahl-literale und ihr Datentyp	86
3.3.3	Zuweisungen und Standardkonversionen bei Ganzzahlausdrücken	88
3.3.4	Operatoren und die „üblichen arithmetischen Konversionen“	91
3.3.5	Der Datentyp <i>bool</i>	97
3.3.6	Die <i>char</i> -Datentypen und der ASCII- und ANSI-Zeichensatz	101
3.3.7	Der Datentyp <i>__int64</i>	108
3.3.8	C++0x-Erweiterungen für Ganzzahldatentypen Θ	108
3.4	Kontrollstrukturen und Funktionen	108
3.4.1	Die <i>if</i> - und die Verbundanweisung	109
3.4.2	Wiederholungsanweisungen	113
3.4.3	Funktionen und der Datentyp <i>void</i>	116
3.4.4	Werte- und Referenzparameter	120
3.4.5	Die Verwendung von Bibliotheken und Namensbereichen	121
3.4.6	Zufallszahlen	122
3.5	Tests und der integrierte Debugger	127
3.5.1	Systematisches Testen	127
3.5.2	Testprotokolle und Testfunktionen für automatisierte Tests	132
3.5.3	Tests mit DUnit im C++Builder 2007	136
3.5.4	Der integrierte Debugger	138
3.6	Gleitkommatypen	142
3.6.1	Die interne Darstellung von Gleitkommawerten	143
3.6.2	Der Datentyp von Gleitkommaliteralen	147
3.6.3	Standardkonversionen	148
3.6.4	Mathematische Funktionen	153
3.6.5	Datentypen für exakte und kaufmännische Rechnungen	155
3.6.6	Ein Kriterium für annähernd gleiche Gleitkommazahlen	162
3.7	Ablaufprotokolle und Programmierlogik	165
3.7.1	Ablaufprotokolle	166
3.7.2	Schleifeninvarianten mit Ablaufprotokollen erkennen	169
3.7.3	Symbolische Ablaufprotokolle	173
3.7.4	Schleifeninvarianten, Ablaufprotokolle, vollständige Induktion Θ	180
3.7.5	Verifikationen, Tests und Bedingungen zur Laufzeit prüfen	187
3.7.6	Funktionsaufrufe und Programmierstil für Funktionen	191

3.7.7	Einfache logische Regeln und Wahrheitstabellen Θ	198
3.7.8	Bedingungen in und nach <i>if</i> -Anweisungen und Schleifen Θ	200
3.8	Konstanten	209
3.9	Syntaxregeln für Deklarationen und Initialisierungen Θ	212
3.10	Arrays und Container	214
3.10.1	Einfache <i>typedef</i> -Deklarationen.....	215
3.10.2	Eindimensionale Arrays.....	215
3.10.3	Die Initialisierung von Arrays bei ihrer Definition.....	223
3.10.4	Arrays als Container	225
3.10.5	Mehrdimensionale Arrays.....	232
3.10.6	Dynamische Programmierung.....	236
3.10.7	Array-Eigenschaften der VCL Θ	237
3.11	Strukturen und Klassen	238
3.11.1	Mit <i>struct</i> definierte Klassen	238
3.11.2	Mit <i>union</i> definierte Klassen Θ	245
3.11.3	Die Datentypen <i>TVarRec</i> und <i>Variant</i> Θ	248
3.11.4	Bitfelder Θ	250
3.12	Zeiger, Strings und dynamisch erzeugte Variablen.....	252
3.12.1	Die Definition von Zeigervariablen	254
3.12.2	Der Adressoperator, Zuweisungen und generische Zeiger	257
3.12.3	Ablaufprotokolle für Zeigervariable.....	261
3.12.4	Dynamisch erzeugte Variablen: <i>new</i> und <i>delete</i>	262
3.12.5	Garbage Collection mit der Smart Pointer Klasse <i>shared_ptr</i>	272
3.12.6	Dynamische erzeugte eindimensionale Arrays	274
3.12.7	Arrays, Zeiger und Zeigerarithmetik	276
3.12.8	Arrays als Funktionsparameter Θ	280
3.12.9	Konstante Zeiger	283
3.12.10	Stringlitterale, nullterminierte Strings und <i>char*</i> -Zeiger.....	285
3.12.11	Verkettete Listen	292
3.12.12	Binärbäume	303
3.12.13	Zeiger als Parameter und Win32 API Funktionen	306
3.12.14	Bibliotheksfunktionen für nullterminierte Strings Θ	310
3.12.15	Die Erkennung von „Memory leaks“ mit CodeGuard Θ	314
3.12.16	Zeiger auf Zeiger auf Zeiger auf ... Θ	315
3.12.17	Dynamisch erzeugte mehrdimensionale Arrays Θ	316
3.13	Die Stringklasse <i>AnsiString</i>	320
3.13.1	Die Definition von Variablen eines Klassentyps	321
3.13.2	Funktionen der Klasse <i>AnsiString</i>	323
3.13.3	Globale <i>AnsiString</i> -Funktionen	326
3.14	Deklarationen mit <i>typedef</i> und <i>typeid</i> -Ausdrücke	333
3.15	Aufzählungstypen	336
3.15.1	<i>enum</i> Konstanten und Konversionen Θ	339
3.16	Kommentare und interne Programmdokumentation.....	340
3.17	Globale, lokale und dynamische Variablen.....	344
3.17.1	Die Deklarationsanweisung	344
3.17.2	Die Verbundanweisung und der lokale Gültigkeitsbereich.....	345
3.17.3	Statische lokale Variablen	348

3.17.4	Lebensdauer von Variablen und Speicherklassenspezifizierer Θ ...	349
3.18	Referenztypen, Werte- und Referenzparameter	352
3.18.1	Werteparameter	353
3.18.2	Referenzparameter	354
3.18.3	Konstante Referenzparameter	356
3.19	Weitere Anweisungen	358
3.19.1	Die Ausdrucksanweisung	358
3.19.2	Exception Handling: <i>try</i> und <i>throw</i>	360
3.19.3	Die <i>switch</i> -Anweisung Θ	365
3.19.4	Die <i>do</i> -Anweisung Θ	368
3.19.5	Die <i>for</i> -Anweisung Θ	369
3.19.6	Die Sprunganweisungen <i>goto</i> , <i>break</i> und <i>continue</i> Θ	372
3.19.7	Assembler-Anweisungen Θ	375
3.20	Ausdrücke	376
3.20.1	Primäre Ausdrücke Θ	377
3.20.2	Postfix-Ausdrücke Θ	379
3.20.3	Unäre Ausdrücke Θ	380
3.20.4	Typkonversionen in Typecast-Schreibweise Θ	383
3.20.5	Zeiger auf Klasselemente Θ	383
3.20.6	Multiplikative Operatoren Θ	383
3.20.7	Additive Operatoren Θ	384
3.20.8	Shift-Operatoren Θ	384
3.20.9	Vergleichsoperatoren Θ	385
3.20.10	Gleichheitsoperatoren Θ	386
3.20.11	Bitweise Operatoren Θ	387
3.20.12	Logische Operatoren Θ	388
3.20.13	Der Bedingungsoperator Θ	388
3.20.14	Konstante Ausdrücke Θ	390
3.20.15	Zuweisungsoperatoren	390
3.20.16	Der Komma-Operator Θ	392
3.20.17	L-Werte und R-Werte Θ	393
3.20.18	Die Priorität und Assoziativität der Operatoren Θ	393
3.20.19	Alternative Zeichenfolgen Θ	396
3.20.20	Explizite Typkonversionen Θ	398
3.21	Namensbereiche	405
3.21.1	Die Definition von benannten Namensbereichen	406
3.21.2	Die Verwendung von Namen aus Namensbereichen	409
3.21.3	Aliasnamen für Namensbereiche	412
3.21.4	Unbenannte Namensbereiche	413
3.22	Präprozessoranweisungen	416
3.22.1	Die <i>#include</i> -Anweisung	417
3.22.2	Makros Θ	418
3.22.3	Bedingte Kompilation Θ	423
3.22.4	Pragmas Θ	428
3.23	Separate Kompilation und statische Bibliotheken	431
3.23.1	C++-Dateien, Header-Dateien und Object-Dateien	432
3.23.2	Bindung Θ	433

- 3.23.3 Deklarationen und Definitionen Θ435
- 3.23.4 Die „One Definition Rule“ Θ437
- 3.23.5 Die Elemente von Header-Dateien und C++-Dateien Θ439
- 3.23.6 Object-Dateien und Statische Bibliotheken linken Θ 441
- 3.23.7 Der Aufruf von in C geschriebenen Funktionen Θ 441
- 3.24 Dynamic Link Libraries (DLLs)443
 - 3.24.1 DLLs erzeugen Θ444
 - 3.24.2 Implizit geladene DLLs Θ 446
 - 3.24.3 Explizit geladene DLLs Θ 448
 - 3.24.4 Hilfsprogramme zur Identifizierung von Funktionen in DLLs Θ ...449
 - 3.24.5 DLLs mit VCL Komponenten Θ 451
 - 3.24.6 Die Verwendung von MS Visual C++ DLLs im C++Builder Θ452

4 Einige Klassen der Standardbibliothek 457

- 4.1 Die Stringklassen *string* und *wstring*458
 - 4.1.1 *AnsiString* und *string*: Gemeinsamkeiten und Unterschiede.....458
 - 4.1.2 Einige Elementfunktionen der Klasse *string*.....461
 - 4.1.3 Stringstreams464
- 4.2 Sequenzielle Container der Standardbibliothek469
 - 4.2.1 Die Container-Klasse *vector*.....469
 - 4.2.2 Iteratoren473
 - 4.2.3 Algorithmen der Standardbibliothek476
 - 4.2.4 Die Speicherverwaltung bei Vektoren Θ 482
 - 4.2.5 Mehrdimensionale Vektoren Θ484
 - 4.2.6 Die Container-Klassen *list* und *deque*485
 - 4.2.7 Gemeinsamkeiten und Unterschiede der sequenziellen Container.487
 - 4.2.8 Die Container-Adapter *stack*, *queue* und *priority_queue* Θ 489
 - 4.2.9 Container mit Zeigern.....491
 - 4.2.10 Die verschiedenen STL-Implementationen im C++Builder Θ491
 - 4.2.11 Die Container-Klasse *bitset* Θ 492
- 4.3 Dateibearbeitung mit den Stream-Klassen493
 - 4.3.1 Stream-Variablen, ihre Verbindung mit Dateien und ihr Zustand ..494
 - 4.3.2 Fehler und der Zustand von Stream-Variablen498
 - 4.3.3 Lesen und Schreiben von Binärdaten mit *read* und *write*499
 - 4.3.4 Lesen und Schreiben von Daten mit den Operatoren << und >>...508
 - 4.3.5 Manipulatoren und Funktionen zur Formatierung von Texten Θ ...516
 - 4.3.6 Dateibearbeitung im Direktzugriff Θ519
 - 4.3.7 Sortieren, Mischen und Gruppenverarbeitung Θ 522
 - 4.3.8 C-Funktionen zur Dateibearbeitung Θ529
- 4.4 Assoziative Container532
 - 4.4.1 Die Container *set* und *multiset*.....533
 - 4.4.2 Die Container *map* und *multimap*.....534
 - 4.4.3 Iteratoren der assoziativen Container536
- 4.5 Die numerischen Klassen der Standardbibliothek.....539
 - 4.5.1 Komplexe Zahlen Θ540

- 4.5.2 Valarrays Θ543
- 4.6 C++0x-Erweiterungen der Standardbibliothek Θ545
 - 4.6.1 Ungeordnete Assoziative Container (Hash Container)545
 - 4.6.2 Die Installation der Boost-Bibliotheken Θ 549
 - 4.6.3 Fixed Size Array Container Θ 552
 - 4.6.4 Tupel Θ553
- 5 Funktionen..... 557**
 - 5.1 Die Verwaltung von Funktionsaufrufen über den Stack.....558
 - 5.1.1 Aufrufkonventionen Θ561
 - 5.2 Funktionszeiger und der Datentyp einer Funktion561
 - 5.2.1 Der Datentyp einer Funktion561
 - 5.2.2 Zeiger auf Funktionen.....563
 - 5.3 Rekursion.....569
 - 5.3.1 Grundlagen570
 - 5.3.2 Quicksort576
 - 5.3.3 Ein rekursiv absteigender Parser580
 - 5.3.4 Rekursiv definierte Kurven Θ585
 - 5.3.5 Indirekte Rekursion Θ 588
 - 5.3.6 Rekursive Datenstrukturen und binäre Suchbäume588
 - 5.3.7 Verzeichnisse rekursiv nach Dateien durchsuchen Θ 593
 - 5.4 Funktionen und Parameter Θ 596
 - 5.4.1 Seiteneffekte und die Reihenfolge von Auswertungen Θ 596
 - 5.4.2 Syntaxregeln für Funktionen Θ599
 - 5.4.3 Der Funktionsbegriff in der Mathematik und in C++ Θ 602
 - 5.4.4 Der Aufruf von Funktionen aus Delphi im C++Builder Θ 603
 - 5.4.5 Unspezifizierte Anzahl und Typen von Argumenten Θ 604
 - 5.4.6 Die Funktionen *main* bzw. *WinMain* und ihre Parameter Θ 606
 - 5.4.7 Traditionelle K&R-Funktionsdefinitionen Θ608
 - 5.5 Default-Argumente610
 - 5.6 Inline-Funktionen.....611
 - 5.7 Überladene Funktionen614
 - 5.7.1 Funktionen, die nicht überladen werden können616
 - 5.7.2 Regeln für die Auswahl einer passenden Funktion617
 - 5.8 Überladene Operatoren mit globalen Operatorfunktionen623
 - 5.8.1 Globale Operatorfunktionen625
 - 5.8.2 Die Inkrement- und Dekrementoperatoren627
 - 5.8.3 Referenzen als Funktionswerte629
 - 5.8.4 Die Ein- und Ausgabe von selbst definierten Datentypen631

6 Objektorientierte Programmierung..... 635

- 6.1 Klassen.....636
 - 6.1.1 Datenelemente und Elementfunktionen636
 - 6.1.2 Der Gültigkeitsbereich von Klasselementen640
 - 6.1.3 Datenkapselung: Die Zugriffsrechte *private* und *public*644
 - 6.1.4 Der Aufruf von Elementfunktionen und der *this*-Zeiger650
 - 6.1.5 Konstruktoren und Destruktoren652
 - 6.1.6 OO Analyse und Design: Der Entwurf von Klassen.....664
 - 6.1.7 Programmierlogik: Klasseninvarianten und Korrektheit672
 - 6.1.8 UML-Diagramme mit Together im C++Builder 2007.....679
- 6.2 Klassen als Datentypen682
 - 6.2.1 Der Standardkonstruktor.....683
 - 6.2.2 Objekte als Klasselemente und Elementinitialisierer685
 - 6.2.3 *friend*-Funktionen und -Klassen690
 - 6.2.4 Überladene Operatoren als Elementfunktionen693
 - 6.2.5 Der Copy-Konstruktor702
 - 6.2.6 Der Zuweisungsoperator = für Klassen709
 - 6.2.7 Benutzerdefinierte Konversionen717
 - 6.2.8 Explizite Konstruktoren Θ722
 - 6.2.9 Statische Klasselemente723
 - 6.2.10 Konstante Klasselemente und Objekte.....725
 - 6.2.11 Klassen und Header-Dateien728
- 6.3 Vererbung und Komposition.....731
 - 6.3.1 Die Elemente von abgeleiteten Klassen.....732
 - 6.3.2 Zugriffsrechte auf die Elemente von Basisklassen.....734
 - 6.3.3 Die Bedeutung von Elementnamen in einer Klassenhierarchie736
 - 6.3.4 *using*-Deklarationen in abgeleiteten Klassen Θ 738
 - 6.3.5 Konstruktoren, Destruktoren und implizit erzeugte Funktionen739
 - 6.3.6 Vererbung bei Formularen im C++Builder.....745
 - 6.3.7 OO Design: *public* Vererbung und „ist ein“-Beziehungen746
 - 6.3.8 OO Design: Komposition und „hat ein“-Beziehungen751
 - 6.3.9 Konversionen zwischen *public* abgeleiteten Klassen.....753
 - 6.3.10 *protected* und *private* abgeleitete Klassen Θ 758
 - 6.3.11 Mehrfachvererbung und virtuelle Basisklassen761
- 6.4 Virtuelle Funktionen, späte Bindung und Polymorphie768
 - 6.4.1 Der statische und der dynamische Datentyp768
 - 6.4.2 Virtuelle Funktionen.....769
 - 6.4.3 Die Implementierung von virtuellen Funktionen: *vptr* und *vtbl*.....780
 - 6.4.4 Virtuelle Konstruktoren und Destruktoren786
 - 6.4.5 Virtuelle Funktionen in Konstruktoren und Destruktoren788
 - 6.4.6 OO-Design: Einsatzbereich und Test von virtuellen Funktionen....789
 - 6.4.7 OO-Design und Erweiterbarkeit791
 - 6.4.8 Rein virtuelle Funktionen und abstrakte Basisklassen794
 - 6.4.9 OO-Design: Virtuelle Funktionen und abstrakte Basisklassen798
 - 6.4.10 OOAD: Zusammenfassung800
 - 6.4.11 Interfaces und Mehrfachvererbung.....804

6.4.12	Zeiger auf Klassenelemente Θ	805
6.4.13	UML-Diagramme für Vererbung und Komposition	810
6.5	Laufzeit-Typinformationen	812
6.5.1	Typinformationen mit dem Operator <i>typeid</i> Θ	813
6.5.2	Typkonversionen mit <i>dynamic_cast</i> Θ	816
6.5.3	Anwendungen von Laufzeit-Typinformationen Θ	819
6.5.4	<i>static_cast</i> mit Klassen Θ	822
6.5.5	Laufzeit-Typinformationen für die Klassen der VCL Θ	823
7	Exception-Handling	827
7.1	Die <i>try</i> -Anweisung	828
7.2	Exception-Handler und Exceptions der Standardbibliothek	831
7.3	Vordefinierte Exceptions der VCL	836
7.4	Der Programmablauf bei Exceptions	838
7.5	Das vordefinierte Exception-Handling der VCL.....	841
7.6	<i>throw</i> -Ausdrücke und selbst definierte Exceptions	842
7.7	Fehler, Exceptions und die Korrektheit von Programmen	848
7.8	Die Freigabe von Ressourcen bei Exceptions	851
7.9	Exceptions in Konstruktoren und Destruktoren	854
7.10	Exception-Spezifikationen	859
7.11	Die Funktion <i>terminate</i> Θ	861
7.12	Das Win32-Exception-Handling mit <i>try-__except</i> Θ	862
8	Die Bibliothek der visuellen Komponenten (VCL)	863
8.1	Besonderheiten der VCL.....	864
8.2	Visuelle Programmierung und Properties (Eigenschaften)	868
8.2.1	Lesen und Schreiben von Eigenschaften	868
8.2.2	Array-Properties Θ	871
8.2.3	Indexangaben Θ	873
8.2.4	Die Speicherung von Eigenschaften in der Formulardatei Θ	874
8.2.5	Die Redeklaration von Eigenschaften.....	876
8.3	Die Klassenhierarchie der VCL	876
8.4	Selbst definierte Komponenten und ihre Ereignisse.....	884
8.5	Die Erweiterung der Tool-Palette	892
8.6	Klassenreferenztypen und virtuelle Konstruktoren	898
8.7	Botschaften (Messages)	903
8.7.1	Die Message Queue und die Window-Prozedur.....	903
8.7.2	Botschaften für eine Anwendung.....	906
8.7.3	Botschaften für ein Steuerelement	907
8.7.4	Selbst definierte Reaktionen auf Botschaften	909
8.7.5	Botschaften versenden.....	914

9 Templates und die STL..... 920

- 9.1 Generische Funktionen: Funktions-Templates921
 - 9.1.1 Die Deklaration von Funktions-Templates mit Typ-Parametern922
 - 9.1.2 Spezialisierungen von Funktions-Templates923
 - 9.1.3 Funktions-Templates mit Nicht-Typ-Parametern930
 - 9.1.4 Explizit instanziierte Funktions-Templates Θ931
 - 9.1.5 Explizit spezialisierte und überladene Templates932
 - 9.1.6 Rekursive Funktions-Templates Θ 936
- 9.2 Generische Klassen: Klassen-Templates.....939
 - 9.2.1 Die Deklaration von Klassen-Templates mit Typ-Parametern940
 - 9.2.2 Spezialisierungen von Klassen-Templates.....941
 - 9.2.3 Templates mit Nicht-Typ-Parametern949
 - 9.2.4 Explizit instanziierte Klassen-Templates Θ950
 - 9.2.5 Partielle und vollständige Spezialisierungen Θ 951
 - 9.2.6 Elemente und *friend*-Funktionen von Klassen-Templates Θ 957
 - 9.2.7 Ableitungen von Templates Θ 961
 - 9.2.8 UML-Diagramme für parametrisierte Klassen Θ962
- 9.3 Funktionsobjekte in der STL.....965
 - 9.3.1 Der Aufrufoperator ()965
 - 9.3.2 Prädikate und arithmetische Funktionsobjekte968
 - 9.3.3 Binder, Funktionsadapter und C++0x-Erweiterungen973
- 9.4 Iteratoren und die STL-Algorithmen.....980
 - 9.4.1 Die verschiedenen Arten von Iteratoren981
 - 9.4.2 Umkehriteratoren.....983
 - 9.4.3 Einfügefunktionen und Einfügeiteratoren.....984
 - 9.4.4 Stream-Iteratoren.....986
 - 9.4.5 Container-Konstruktoren mit Iteratoren987
 - 9.4.6 STL-Algorithmen für alle Elemente eines Containers988
- 9.5 Die Algorithmen der STL991
 - 9.5.1 Lineares Suchen.....991
 - 9.5.2 Zählen.....993
 - 9.5.3 Der Vergleich von Bereichen994
 - 9.5.4 Suche nach Teilfolgen995
 - 9.5.5 Minimum und Maximum.....996
 - 9.5.6 Elemente vertauschen998
 - 9.5.7 Kopieren von Bereichen.....998
 - 9.5.8 Elemente transformieren und ersetzen.....1000
 - 9.5.9 Elementen in einem Bereich Werte zuweisen.....1002
 - 9.5.10 Elemente entfernen1002
 - 9.5.11 Die Reihenfolge von Elementen vertauschen1004
 - 9.5.12 Permutationen.....1005
 - 9.5.13 Partitionen1006
 - 9.5.14 Bereiche sortieren.....1007
 - 9.5.15 Binäres Suchen in sortierten Bereichen1008
 - 9.5.16 Mischen von sortierten Bereichen1009
 - 9.5.17 Mengenoperationen auf sortierten Bereichen1010

9.5.18	Heap-Operationen.....	1012
9.5.19	Verallgemeinerte numerische Operationen.....	1013

10 Verschiedenes 1015

10.1	Symbolleisten, Menüs und Aktionen	1015
10.1.1	Symbolleisten mit Panels und SpeedButtons	1016
10.1.2	Symbolleisten mit Toolbars	1016
10.1.3	Verschiebbare Komponenten mit CoolBar und ControlBar	1017
10.1.4	Die Verwaltung von Aktionen	1017
10.2	Eigene Dialoge, Frames und die Objektablage	1023
10.2.1	Die Anzeige von weiteren Formularen und modale Fenster	1023
10.2.2	Vordefinierte Dialogfelder der Objektablage	1026
10.2.3	Funktionen, die vordefinierte Dialogfelder anzeigen.....	1027
10.2.4	Die Erweiterung der Tool-Palette mit Frames	1029
10.2.5	Datenmodule.....	1030
10.2.6	Die Objektablage	1030
10.3	Größenänderung von Steuerelementen zur Laufzeit	1031
10.3.1	Die Eigenschaften <i>Align</i> und <i>Anchor</i>	1031
10.3.2	Die Komponenten <i>Splitter</i> und <i>HeaderControl</i>	1032
10.3.3	GridPanel: Tabellen mit Steuerelementen	1033
10.3.4	Automatisch angeordnete Steuerelemente: FlowPanel	1035
10.4	Listview und TreeView	1035
10.4.1	Die Anzeige von Listen mit Listview	1035
10.4.2	Listview nach Spalten sortieren	1038
10.4.3	Die Anzeige von Baumdiagrammen mit TreeView	1040
10.5	Formatierte Texte mit der RichEdit-Komponente.....	1045
10.6	Tabellen	1047
10.7	Schieberegler: ScrollBar und TrackBar	1049
10.8	Weitere Eingabekomponenten	1051
10.8.1	Texteingaben mit MaskEdit filtern.....	1051
10.8.2	Die Auswahl von Laufwerken und Verzeichnissen	1053
10.9	Status- und Fortschrittsanzeigen	1055
10.10	Klassen und Funktionen zu Uhrzeit und Kalenderdatum	1056
10.10.1	<i>TDateTime</i> -Funktionen.....	1056
10.10.2	Zeitgesteuerte Ereignisse mit einem Timer.....	1058
10.10.3	Hochauflösende Zeitmessung	1059
10.10.4	Kalenderdaten und Zeiten eingeben	1060
10.11	Multitasking und Threads.....	1062
10.11.1	Multithreading mit der Klasse <i>TThread</i>	1063
10.11.2	Der Zugriff auf VCL-Elemente mit <i>Synchronize</i>	1065
10.11.3	Kritische Abschnitte und die Synchronisation von Threads	1067
10.12	TrayIcon.....	1069
10.13	<i>TCanvas</i> und <i>TImage</i> : Grafiken anzeigen und zeichnen	1070
10.13.1	Grafiken anzeigen mit <i>TImage</i>	1070
10.13.2	Grafiken zeichnen mit <i>TCanvas</i>	1070

- 10.13.3 Welt- und Bildschirmkoordinaten1071
- 10.13.4 Figuren, Farben, Stifte und Pinsel1073
- 10.13.5 Text auf einen Canvas schreiben1075
- 10.13.6 Drucken mit *TPrinter*1076
- 10.13.7 Grafiken im BMP- und WMF-Format speichern.....1077
- 10.13.8 Auf den Canvas einer PaintBox oder eines Formulars zeichnen ..1078
- 10.14 Die Steuerung von MS-Office: Word-Dokumente erzeugen.....1085
- 10.15 Datenbank-Komponenten der VCL.....1088
 - 10.15.1 Verbindung mit ADO-Datenbanken – der Connection-String.....1089
 - 10.15.2 Tabellen und die Komponente *TDataSet*.....1096
 - 10.15.3 Tabellendaten lesen und schreiben1098
 - 10.15.4 Die Anzeige von Tabellen mit einem *DBGrid*.....1101
 - 10.15.5 SQL-Abfragen1102
- 10.16 Internet-Komponenten.....1104
- 10.17 MDI-Programme1107
- 10.18 Die Klasse *Set*1110
- 10.19 3D-Grafik mit OpenGL1113
 - 10.19.1 Initialisierungen1114
 - 10.19.2 Grafische Grundelemente: Primitive1117
 - 10.19.3 Modelltransformationen1121
 - 10.19.4 Vordefinierte Körper1124
 - 10.19.5 Lokale Transformationen.....1126
 - 10.19.6 Beleuchtungseffekte1129
 - 10.19.7 Texturen1132
- 10.20 Win32-Funktionen zur Dateibearbeitung1136
 - 10.20.1 Elementare Funktionen.....1137
 - 10.20.2 File-Sharing1140
 - 10.20.3 Record-Locking.....1141
 - 10.20.4 VCL-Funktionen zur Dateibearbeitung und *TFileStream*.....1142
- 10.21 Datenübertragung über die serielle Schnittstelle1145
 - 10.21.1 Grundbegriffe1145
 - 10.21.2 Standards für die serielle Schnittstelle: RS-232C bzw. V.24.....1146
 - 10.21.3 Win32-Funktionen zur seriellen Kommunikation.....1148

Literaturverzeichnis.....1153

Inhalt Buch-CD.....1159

Index1161

Ø Angesichts des Umfangs dieses Buches habe ich einige Abschnitte mit dem Zeichen Ø in der Überschrift als „weniger wichtig“ gekennzeichnet. Damit will ich dem Anfänger eine kleine Orientierung durch die Fülle des Stoffes geben. Diese Kennzeichnung bedeutet aber keineswegs, dass dieser Teil unwichtig ist – vielleicht sind gerade diese Inhalte für Sie besonders relevant.

1 Die Entwicklungsumgebung

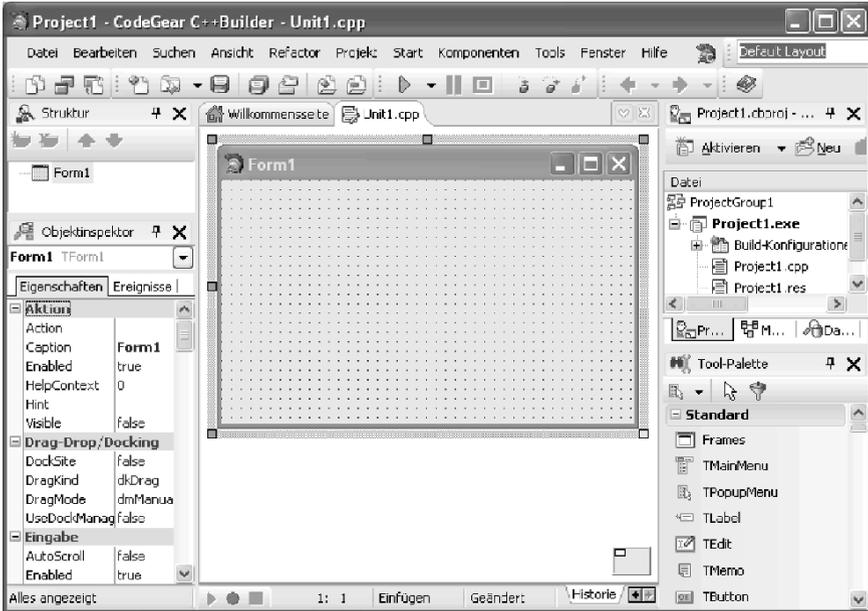
Der C++Builder besteht aus verschiedenen Werkzeugen (Tools), die einen Programmierer bei der Entwicklung von Software unterstützen. Eine solche Zusammenstellung von Werkzeugen zur Softwareentwicklung bezeichnet man auch als Programmier- oder **Entwicklungsumgebung**.

Einfache Entwicklungsumgebungen bestehen nur aus einem Editor und einem Compiler. Für eine effiziente Entwicklung von komplexeren Anwendungen (dazu gehören viele Windows-Anwendungen) sind aber oft weitere Werkzeuge notwendig. Wenn diese wie im C++Builder in einem einzigen Programm integriert sind, spricht man auch von einer **integrierten Entwicklungsumgebung** (engl.: „integrated development environment“, **IDE**).

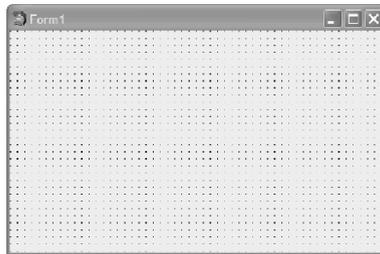
In diesem Kapitel wird zunächst an einfachen Beispielen gezeigt, wie man mit dem C++Builder Windows-Programme mit einer grafischen Benutzeroberfläche entwickeln kann. Anschließend (ab Abschnitt 1.3) werden dann die wichtigsten Werkzeuge des C++Builders ausführlicher vorgestellt. Für viele einfache Anwendungen (wie z.B. die Übungsaufgaben) reichen die Abschnitte bis 1.7. Die folgenden Abschnitte sind nur für anspruchsvollere oder spezielle Anwendungen notwendig. Sie sind deshalb mit dem Zeichen Θ (siehe Seite xxiii) gekennzeichnet und können übergangen werden. Weitere Elemente der Entwicklungsumgebung werden später beschrieben, wenn sie dann auch eingesetzt werden können.

1.1 Visuelle Programmierung: Ein erstes kleines Programm

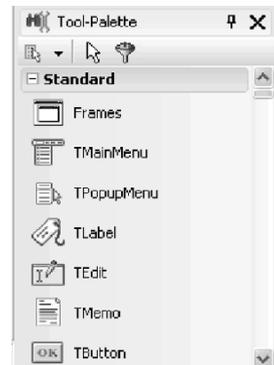
Im C++Builder kann man mit *Datei/Neu* Projekte für verschiedene Arten von Anwendungen anlegen. Ein Projekt für ein Windowsprogramm mit einer grafischen Benutzeroberfläche erhält man mit **VCL-Formularanwendung** – **C++Builder**. Anschließend wird die Entwicklungsumgebung mit einigen ihrer Tools angezeigt:



Das **Formular** ist der Ausgangspunkt für alle Windows-Anwendungen, die mit dem C++Builder entwickelt werden. Es entspricht dem Fenster, das beim Start der Anwendung angezeigt wird:

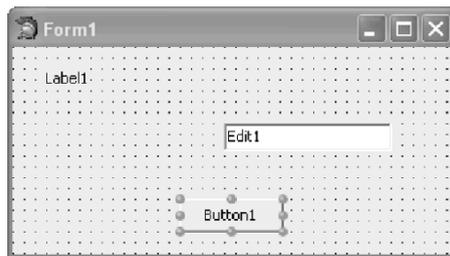


Ein Formular kann mit den in der **Tool-Palette** verfügbaren **Steuerelementen (Controls)** gestaltet werden. Die Tool-Palette zeigt praktisch alle der unter Windows üblichen Steuerelemente an, wenn das Formular angezeigt wird. Sie sind auf verschiedene Gruppen (*Standard*, *Zusätzlich* usw.) verteilt, die über die Icons + und – auf- und zugeklappt werden können. Ein Teil dieser Komponenten (wie z.B. ein Button) entspricht Steuerelementen, die im laufenden Programm angezeigt werden. Andere, wie der *Timer* von der Seite *System*, sind im laufenden Programm nicht sichtbar.



Um eine Komponente aus der Tool-Palette auf das Formular zu setzen, klickt man sie zuerst mit der Maus an (sie wird dann als markiert dargestellt). Anschließend klickt man mit dem Mauszeiger auf die Stelle im Formular, an die die linke obere Ecke der Komponente kommen soll.

Beispiel: Nachdem man ein Label (die Komponente mit dem Namen *TLabel*), ein Edit-Fenster (Name *TEdit*) und einen Button (Name *TButton*, mit der Aufschrift *OK*) auf das Formular gesetzt hat, sieht es etwa folgendermaßen aus:



Durch diese Spielereien haben Sie **schon ein richtiges Windows-Programm** erstellt – zwar kein besonders nützliches, aber immerhin. Sie können es folgendermaßen starten:

- mit *Start/Start* von der Menüleiste, oder
- mit *F9* von einem beliebigen Fenster im C++Builder oder
- durch den Aufruf der vom Compiler erzeugten Exe-Datei.

Dieses Programm hat schon viele Eigenschaften, die man von einem Windows-Programm erwartet: Man kann es mit der Maus verschieben, vergrößern, verkleinern und schließen.

Bemerkenswert an diesem Programm ist vor allem der im Vergleich zu einem nichtvisuellen Entwicklungssystem **geringe Aufwand**, mit dem es erstellt wurde. So braucht Petzold in seinem Klassiker „Programmierung unter Windows“ (Petzold 1992, S. 33) ca. 80 Zeilen nichttriviale C-Anweisungen, um den Text „Hello Windows“ wie in einem Label in ein Fenster zu schreiben. Und in jeder dieser 80 Zeilen kann man einiges falsch machen.

Vergessen Sie nicht, Ihr **Programm** zu **beenden**, bevor Sie es weiterbearbeiten. Sie können den Compiler nicht erneut starten, solange das Programm noch läuft.

Diese Art der Programmierung bezeichnet man als **visuelle Programmierung**. Während man bei der konventionellen Programmierung ein Programm ausschließlich durch das Schreiben von Anweisungen (Text) in einer Programmier-

sprache entwickelt, wird es bei der visuellen Programmierung ganz oder teilweise aus vorgefertigten grafischen Komponenten zusammengesetzt.

Mit dem C++Builder kann die Benutzeroberfläche eines Programms visuell gestaltet werden. Damit sieht man bereits beim Entwurf des Programms, wie es später zur Laufzeit aussehen wird. Die Anweisungen, die als Reaktionen auf Benutzer-eingaben (Mausklicks usw.) erfolgen sollen, werden dagegen konventionell in der Programmiersprache C++ geschrieben.

Die zuletzt auf einem Formular (bzw. im Pull-down-Menü des Objektinspektors) angeklickte Komponente wird als die **aktuell ausgewählte Komponente** bezeichnet. Man erkennt sie an den 8 kleinen blauen Punkten an ihrem Rand. An ihnen kann man mit der Maus ziehen und so die Größe der Komponente verändern. Ein **Formular** wird dadurch zur **aktuell ausgewählten Komponente**, indem man mit der Maus eine freie Stelle im Formular anklickt.

Beispiel: Im letzten Beispiel ist *Button1* die aktuell ausgewählte Komponente.

Der **Objektinspektor** zeigt die Eigenschaften (properties) der aktuell ausgewählten Komponente an. In der linken Spalte stehen die **Namen** und in der rechten die **Werte** der Eigenschaften. Mit der Taste *F1* erhält man eine Beschreibung der Eigenschaft.

Den Wert einer Eigenschaft kann man über die rechte Spalte verändern. Bei manchen Eigenschaften kann man den neuen Wert über die Tastatur eintippen. Bei anderen wird nach dem Anklicken der rechten Spalten ein kleines Dreieck für ein Pull-down-Menü angezeigt, über das ein Wert ausgewählt werden kann. Oder es wird ein Symbol mit drei Punkten „...“ angezeigt, über das man Werte eingeben kann.



Beispiel: Bei der Eigenschaft *Caption* kann man mit der Tastatur einen Text eingeben. Bei einem Button ist dieser Text die Aufschrift auf dem Button (z.B. „OK“), und bei einem Formular die Titelzeile (z.B. „Mein erstes C++-Programm“).

Bei der Eigenschaft *Color* (z.B. bei einer Edit-Komponente) kann man über ein Pull-down-Menü die **Hintergrundfarbe** auswählen (z.B. ein wunderschönes *clLime*). Bei der Eigenschaft *Cursor* kann man über ein Pull-down-Menü die Form des Cursors auswählen, die zur Laufzeit angezeigt wird, wenn der Cursor über dem Steuerelement ist.

Klickt man die rechte Spalte der Eigenschaft *Font* und dann das Symbol „...“ an, kann man die **Schriftart** der Eigenschaft *Caption* auswählen.

Eine Komponente auf dem Formular wird nicht nur an ihre Eigenschaften im Objektinspektor angepasst, sondern auch umgekehrt: Wenn man die Größe einer

Komponente durch Ziehen an den Ziehquadraten verändert, werden die Werte der entsprechenden Eigenschaften (*Left*, *Top*, *Height* oder *Width*) im Objektinspektor automatisch aktualisiert.

1.2 Erste Schritte in C++

Als nächstes soll das Programm aus dem letzten Abschnitt so erweitert werden, dass als Reaktion auf Benutzereingaben (z.B. beim Anklicken eines Buttons) Anweisungen ausgeführt werden.

Windows-Programme können Benutzereingaben in Form von Mausklicks oder Tastatureingaben entgegennehmen. Im Unterschied zu einfachen Konsolen-Programmen (z.B. DOS-Programmen) muss man in einem Windows-Programm aber keine speziellen Funktionen (wie *Readln* in Pascal oder *scanf* in C) aufrufen, die auf solche Eingaben warten. Stattdessen werden alle Eingaben von Windows zentral entgegengenommen und als sogenannte Botschaften (Messages) an das entsprechende Programm weitergegeben. Dadurch wird in diesem Programm ein sogenanntes **Ereignis** ausgelöst.

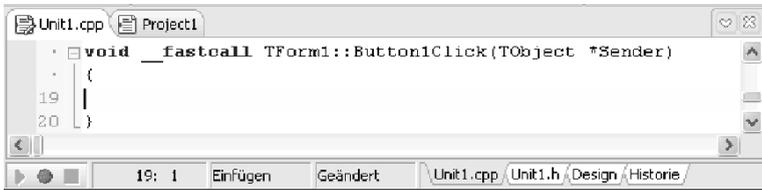
Die Ereignisse, die für die aktuell ausgewählte Komponente eintreten können, zeigt der Objektinspektor an, wenn man das **Register Ereignisse** anklickt.

Die Abbildung rechts zeigt einige Ereignisse für einen Button. Dabei steht *OnClick* für das Ereignis, das beim Anklicken des Buttons eintritt. Offensichtlich kann ein Button nicht nur auf das Anklicken reagieren, sondern auch noch auf zahlreiche andere Ereignisse.

Einem solchen Ereignis kann eine Funktion zugeordnet werden, die dann aufgerufen wird, wenn das Ereignis eintritt. Diese Funktion wird auch als **Ereignisbehandlungsroutine** (engl. **event handler**) bezeichnet. Sie wird vom C++Builder durch einen Doppelklick auf die rechte Spalte des Ereignisses erzeugt und im **Quelltexteditor** angezeigt. Der Cursor steht dann am Anfang der Funktion.

Vorläufig soll unser Programm allerdings nur auf das Anklicken eines Buttons reagieren. Die bei diesem Ereignis aufgerufene Funktion erhält man am einfachsten durch einen Doppelklick auf den Button im Formular. Dadurch erzeugt der C++Builder die folgende Funktion:



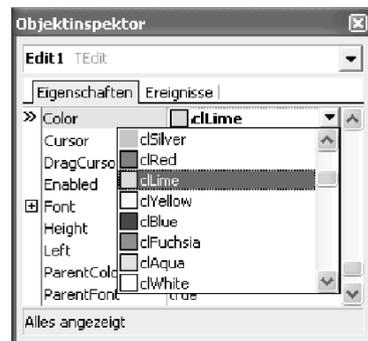


Zwischen die geschweiften Klammern „{“ und „}“ schreibt man dann die **Anweisungen**, die ausgeführt werden sollen, wenn das Ereignis *OnClick* eintritt.

Welche Anweisungen hier möglich sind und wie diese aufgebaut werden müssen, ist der Hauptgegenstand dieses Buches und wird ab Kapitel 3 ausführlich beschrieben. Im Rahmen dieses einführenden Kapitels sollen nur einige wenige Anweisungen vorgestellt werden und diese auch nur so weit, wie das zum Grundverständnis des C++Builders notwendig ist. Falls Ihnen Begriffe wie „Variablen“ usw. neu sind, lesen Sie trotzdem weiter – aus dem Zusammenhang erhalten Sie sicherlich eine intuitive Vorstellung, die zunächst ausreicht. Später werden diese Begriffe dann genauer erklärt.

Eine beim Programmieren häufig verwendete Anweisung ist die **Zuweisung** (mit dem Operator „=“), mit der man einer Variablen einen Wert zuweisen kann. Als Variablen sollen zunächst nur solche Eigenschaften von Komponenten verwendet werden, die auch im Objektinspektor angezeigt werden. Diesen Variablen können dann die Werte zugewiesen werden, die auch im Objektinspektor in der rechten Spalte der Eigenschaften vorgesehen sind.

In der Abbildung rechts sieht man einige zulässige Werte für die Eigenschaft *Color* (Abschnitt *Visuell*). Sie werden nach dem Aufklappen des Pull-down-Menüs angezeigt.



Schreibt man jetzt die Anweisung

```
Edit1->Color = clLime;
```

zwischen die geschweiften Klammern

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Edit1->Color = clLime;
}
```

erhält die Eigenschaft *Color* von *Edit1* beim Anklicken von *Button1* während der Ausführung des Programms den Wert *clLime*, der für die Farbe Limonengrün steht. Wenn Sie das Programm jetzt mit *F9* starten und dann *Button1* anklicken, erhält das Edit-Fenster tatsächlich diese Farbe.

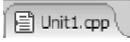
Auch wenn dieses Programm noch nicht viel sinnvoller ist als das erste, haben Sie doch gesehen, wie mit dem C++Builder Windows-Anwendungen entwickelt werden. Dieser **Entwicklungsprozess** besteht immer aus den folgenden Aktivitäten:

1. Man gestaltet die Benutzeroberfläche, indem man Komponenten aus der Tool-Palette auf das Formular setzt (drag and drop) und ihre Eigenschaften im Objektinspektor oder das Layout mit der Maus anpasst (visuelle Programmierung).
2. Man schreibt in C++ die Anweisungen, die als Reaktion auf Benutzereingaben erfolgen sollen (nichtvisuelle Programmierung).
3. Man startet das Programm und testet, ob es sich auch wirklich so verhält, wie es sich verhalten soll.

Der Zeitraum der Programmentwicklung (Aktivitäten 1. und 2.) wird auch als **Entwurfszeit** bezeichnet. Im Unterschied dazu bezeichnet man die Zeit, während der ein Programm läuft, als **Laufzeit** eines Programms.

1.3 Der Quelltexteditor

Der Quelltexteditor (kurz: **Editor**) ist das Werkzeug, mit dem die Quelltexte geschrieben werden. Er ist in die Entwicklungsumgebung integriert und kann auf verschiedene Arten aufgerufen werden, wie z.B.

- durch Anklicken eines Registers, wie z.B.  oder 
- über *Ansicht/Units* oder *F12*
- durch einen Doppelklick auf die rechte Spalte eines Ereignisses im Objektinspektor. Der Cursor befindet sich dann in der Ereignisbehandlungsroutine für dieses Ereignis.
- durch einen Doppelklick auf eine Komponente in einem Formular. Der Cursor befindet sich dann in einer bestimmten Ereignisbehandlungsroutine dieser Komponente.

Da die letzten beiden Arten den Cursor in eine bestimmte Ereignisbehandlungsroutine platzieren, bieten sie eine einfache Möglichkeit, diese Funktion zu finden, ohne sie im Editor suchen zu müssen.

Der Editor enthält über Tastenkombinationen zahlreiche Funktionen, mit denen sich nahezu alle Aufgaben effektiv durchführen lassen, die beim Schreiben von Programmen auftreten. In der ersten der nächsten beiden Tabellen sind einige der Funktionen zusammengestellt, die man auch in vielen anderen Editoren findet.

Tastenkürzel	Aktion oder Befehl
<i>Strg+F</i>	wie <i>Suchen/Suchen</i>
<i>F3</i>	wie <i>Suchen/Suche wiederholen</i>
<i>Strg+R</i>	wie <i>Suchen/Ersetzen</i>
<i>Strg+S</i>	wie <i>Datei/Speichern</i>
<i>Strg+Entf</i>	löscht das Wort ab der Cursorposition
<i>Strg+Y</i>	löscht die gesamte Zeile
<i>Strg+Rücktaste</i>	löscht das Wort links vom Cursor
<i>Strg+Umschalt+Y</i>	löscht die Zeile ab dem Cursor bis zum Ende
<i>Alt+Rücktaste</i> oder <i>Strg+Z</i>	wie <i>Bearbeiten/Rückgängig</i> . Damit können Editor-Aktionen rückgängig gemacht werden
<i>Alt+Umschalt+Rücktaste</i> oder <i>Strg+Umschalt+Z</i>	wie <i>Bearbeiten/Wiederherstellen</i>
<i>Pos1</i> bzw. <i>Ende</i>	an den Anfang bzw. das Ende der Zeile springen
<i>Strg+Pos1</i> bzw. <i>Strg+Ende</i>	an den Anfang bzw. das Ende der Datei springen
<i>Strg+←</i> bzw. <i>Strg+→</i>	um ein Wort nach links bzw. rechts springen
<i>Strg+Bild↑</i> bzw. <i>Strg+Bild↓</i>	an den Anfang bzw. das Ende der Seite springen
<i>Strg+↑</i> bzw. <i>Strg+↓</i>	Text um eine Zeile nach oben bzw. unten verschieben; die Position des Cursors im Text bleibt gleich
<i>Einfg</i>	schaltet zwischen Einfügen und Überschreiben um

Dazu kommen noch die üblichen **Tastenkombinationen unter Windows**, wie Markieren eines Textteils mit gedrückter Umschalt-Taste und gleichzeitigem Bewegen des Cursors bzw. der Maus bei gedrückter linker Maustaste. Ein markierter Bereich kann mit *Strg+X* ausgeschnitten, mit *Strg+C* in die Zwischenablage kopiert und mit *Entf* gelöscht werden. *Strg+V* fügt den Inhalt der Zwischenablage ein.

Die nächste Tabelle enthält Funktionen, die vor allem beim Programmieren nützlich sind, und die man in einer allgemeinen Textverarbeitung nur selten findet. Einige dieser Optionen werden auch in einer Symbolleiste (*Ansicht/Symbolleisten*) angezeigt:

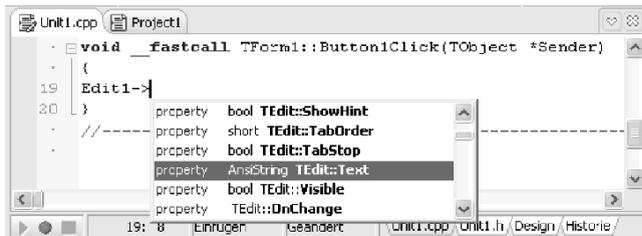
Tastenkürzel	Aktion oder Befehl
<i>F9</i> oder 	kompilieren und starten, wie <i>Start/Start</i>
<i>Strg+F9</i>	kompilieren, aber nicht starten
<i>Strg+F2</i> oder 	Laufendes Programm beenden, wie <i>Start/Programm abbrechen</i> . Damit können oft auch Programme beendet werden, die mit  nicht beendet werden können. Versuchen Sie immer zuerst diese Option wenn Sie meinen, Sie müssten den C++Builder mit dem Windows Task Manager beenden.
<i>F1</i> bzw. <i>Strg+F1</i>	wie <i>Hilfe/Borland-Hilfe</i> , oft kontextsensitiv
<i>Strg+Enter</i>	falls der Text unter dem Cursor einen Dateinamen darstellt, wird diese Datei geöffnet
<i>Strg+Umschalt+I</i> bzw. <i>Strg+Umschalt+U</i>	rückt den als Block markierten Text eine Spalte nach links bzw. rechts (z.B. zum Aus- und Einrücken von {}-Blöcken)
<i>Alt+[</i> bzw. <i>Alt+]</i> bei einer amerikanischen Tastatur und <i>Strg+Q+Ü</i> bei einer deutschen	setzt den Cursor vor die zugehörige Klammer, wenn er vor einer Klammer (z.B. (), {}, [] oder <>) steht
<i>Strg+#</i>	einen markierten Block mit // auskommentieren bzw. die Auskommentierung entfernen
rechte Maustaste, <i>Umgeben</i>	einen markierten Block in einen /*...*/-Kommentar oder eine Anweisung einfügen
rechte Maustaste, <i>Einblenden/Ausblenden</i>	ganze Funktionen, Klassen usw. auf- oder zuklappen
<i>F11</i>	wie <i>Ansicht/Objektinspektor</i>
<i>F12</i>	wie <i>Ansicht/Umschalten Formular/Unit</i>
<i>Alt+0</i>	zeigt Liste offenen Fenster, wie <i>Ansicht/Fensterliste</i>
<i>Alt+Maus bewegen</i> bzw. <i>Alt+Umschalt+Pfeiltaste</i> (←, →, ↑ oder ↓)	zum Markieren von Spalten, z.B. <pre> /// <summary> /// Clean up any /// </summary> /// <param name= </pre>
<i>Strg+K+n</i> (n eine Ziffer)	setzt oder löscht die Positionsmarke n (wie die Positionsmarken-Befehle im Kontextmenü)
<i>Strg+n</i>	springt zur Positionsmarke n
<i>Strg+Tab</i> bzw. <i>Strg+Umschalt+Tab</i>	zeigt das nächste bzw. das vorherige Editor-Fenster an

Eine ausführlichere Beschreibung der Tastaturbelegung findet man in der Online-Hilfe (*Hilfe/Borland Hilfe*) unter *Hilfe/Inhalt/Borland Hilfe/Developer Studio 2006 (Allgemein)/Referenz/Tastenzuordnungen/Standard-Tastaturvorlage*.

Die folgenden Programmierhilfen beruhen auf einer Analyse des aktuellen Programms. Sie werden zusammen mit einigen weiteren sowohl unter dem Oberbegriff **Code Insight** als auch unter dem Oberbegriff **Programmierhilfe** zusammengefasst:

- **Code-Vervollständigung:** Nachdem man den Namen einer Komponente (genauer: eines Klassenobjekts bzw. eines Zeigers auf ein Klassenobjekt) und den zugehörigen Operator („.“ oder „->“) eingetippt hat, wird eine Liste mit allen Elementen der Klasse angezeigt. Aus dieser Liste kann man mit der Enter-Taste ein Element auswählen.
- **Code-Parameter:** Zeigt nach dem Eintippen eines Funktionsnamens und einer öffnenden Klammer die Parameter der Funktion an
- **Symbolinformation durch Kurzhinweis:** Wenn man mit der Maus über einen Namen für ein zuvor definiertes Symbol fährt, werden Informationen über die Deklaration angezeigt.
- Wenn die Option *Tools/Optionen/Editor-Optionen/Programmierhilfe/Quelltext Template Vervollständigung* aktiviert ist, wird nach dem Eintippen eines Wortes aus der mit *Ansicht/Templates* angezeigten Liste und einem Tab- bzw. Leerzeichen der Code entsprechend vervollständigt.

Beispiel: Wenn das Formular eine Edit-Komponente *Edit1* enthält, wird nach dem Eintippen von „Edit1->“ eine Liste mit allen Elementen von *Edit1* angezeigt:



Tipt man weitere Buchstaben ein, werden nur die Elemente mit diesen Anfangsbuchstaben angezeigt.

Falls Sie einen langsamen Rechner und große Programme haben, können diese Programmierhilfen den C++Builder unangenehm langsam machen. Dann kann man sie unter *Tools/Optionen/Editor-Optionen/Programmierhilfe* abschalten.

Der linke Rand im Editor enthält Zeilennummern und grüne und gelbe Linien. Sie bedeuten, dass der Text in der aktuellen Sitzung geschrieben bzw. noch nicht gespeichert ist. Vor ausführbaren Anweisungen stehen blaue Punkte.

Wenn man einen Block markiert, der mindestens einen Bezeichner mehrfach enthält, wird das **Sync-Bearbeitungsmodus** Symbol  angezeigt. Klickt man es