

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 Microsoft Visual C++ 2008

Einführung in Standard-C++, C++/CLI
und die objektorientierte
Windows .NET-Programmierung

 Springer

Prof. Richard Kaiser
Schwärzlocher Str. 53
72070 Tübingen
Deutschland
<http://www.rkaiser.de/>

Xpert.press ISSN 1439-5428
ISBN 978-3-540-23869-0 e-ISBN 978-3-540-68844-0
DOI 10.1007/978-3-540-68844-0
Springer Heidelberg Dordrecht London New York

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

© Springer-Verlag Berlin Heidelberg 2009

Dieses Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, des Vortrags, der Entnahme von Abbildungen und Tabellen, der Funk- sendung, der Mikroverfilmung oder der Vervielfältigung auf anderen Wegen und der Speicherung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Eine Ver- vielfältigung 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.

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

Einbandentwurf: KuenkelLopka GmbH, Heidelberg

Gedruckt auf säurefreiem Papier

Springer ist Teil der Fachverlagsgruppe Springer Science+Business Media (www.springer.com)

Für Daniel, Alex und Kathy

Geleitwort

Wenn man heute nach Literatur über Programmiersprachen sucht, so findet man viel Neues über Sprachen wie C# und Java, die von einer virtuellen Maschine ausgeführt werden, aber auch über die dynamischen Sprachen wie Python, Ruby und PHP. Es könnte der Eindruck entstehen, dass Softwareentwicklung im Wesentlichen mit diesen Sprachen stattfindet. Dies ist aber nicht der Fall. Ein nicht zu unterschätzender Teil der professionellen Softwareentwicklung wird auf der Basis der Programmiersprache C++ durchgeführt. In meiner beruflichen Praxis, in der ich Unternehmen betreue, die mit den Softwareentwicklungswerkzeugen von Microsoft arbeiten, begegne ich häufig der Programmiersprache C++.

C++ ist auch heute immer noch die erste Wahl, wenn es darum geht, hocheffiziente Software für den technisch-wissenschaftlichen Bereich oder hardwarenahe Aufgaben zu entwickeln. Man darf auch nicht vergessen, dass für C++ umfangreiche, qualitativ hochwertige und plattformunabhängige Klassenbibliotheken für viele Anwendungsdomänen existieren. Ein anderer Vorteil von C++ ist meiner Ansicht nach, dass die Sprache sowohl das prozedurale als auch das objektorientierte Entwicklungsparadigma unterstützt. Für Microsoft spielt C++ weiterhin eine wichtige Rolle bei der Softwareentwicklung im eigenen Haus, aber auch in der Weiterentwicklung der Werkzeuge für die Programmierung mit C++, um diese noch produktiver und sicherer zu machen. Microsoft arbeitet aktiv im ISO Komitee zur Standardisierung von C++ mit und hat mit dem Standard ECMA-372 (C++/CLI) C++ für den Einsatz auf der .NET Laufzeitumgebung erweitert. Außerdem ist die einfache Integration von nativen C++ Klassenbibliotheken mit Programmen, die für Microsoft .NET entwickelt werden bzw. wurden, eine nicht zu unterschätzende Eigenschaft der .NET Laufzeitumgebung, einmal unter dem Gesichtspunkt des Investitionsschutzes, als auch unter dem Aspekt der Laufzeiteffizienz.

Das vorliegende Buch von Richard Kaiser, der selbst aktiv im DIN an der Standardisierung von C++ mitarbeitet, gibt eine umfassende Einführung in alle Aspekte von Standard C++, sowie in die Spezialitäten von C++ auf der Microsoft Windows Plattform und C++/CLI für Microsoft .NET. Die Kapitel 1 und 2 bieten eine sehr gute Einführung in das Arbeiten mit Microsoft Visual Studio 2008. Sehr positiv ist noch, dass die prozeduralen und objektorientierten Eigenschaften von C++ explizit dargestellt werden, und dass in Kapitel 3.7 eine Einführung in die Techniken und Möglichkeiten der Programmverifikation im Kontext von C++

gegeben wird. Diese Techniken werden an Bedeutung gewinnen, um die Qualität und Sicherheit von Software besser zu gewährleisten.

Ich wünsche dem Buch viel Erfolg, denn es bietet eine fundierte Einführung in alle Aspekte der Softwareentwicklung mit C++ auf der Microsoft Windows Plattform, einschließlich dem .NET Framework, und es trägt dazu bei, den Einstieg in diese mächtige Programmiersprache zu erleichtern.

Klaus Rohe, Platform Strategy Manager, Developer Platform & Strategy Group, Microsoft Deutschland GmbH

Vorwort

Dieses Buch entstand ursprünglich aus dem Wunsch, in meinen Vorlesungen über C++ nicht nur Textfensterprogramme (Konsolenanwendungen), sondern von Anfang an Windows-Programme zu entwickeln. Dafür ist Visual Studio 2008 sehr gut geeignet. Es ist so einfach zu bedienen, 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.

Dieses Buch richtet sich aber nicht nur an Anfänger, sondern ebenso an professionelle Software-Entwickler. In den letzten 10 Jahren habe ich zahlreiche C++-Kurse für Entwickler aus der Industrie gehalten. Dabei wurde ich mit einer Fülle von Anregungen aus ihrer täglichen Arbeit konfrontiert, die dem Buch viele praxisorientierte Impulse gaben. Für diese Leser wird C++ und C++/CLI umfassend dargestellt.

Dieses Buch besteht aus drei Teilen:

- Teil 1 stellt die Entwicklungsumgebung Visual Studio 2008 vor und zeigt, wie man Windows-Programme mit den wichtigsten Steuerelementen entwickelt.
- Teil 2 stellt den gesamten Sprachumfang des C++-Standards umfassend vor. Dazu gehören nicht nur die Sprachelemente von C sowie Klassen, Templates und Exception-Handling, sondern auch die C++-Standardbibliothek.
- Teil 3 behandelt den gesamten C++/CLI-Standard und gibt einen Einblick in die .NET Klassenbibliothek. Die C++/CLI-Erweiterungen bieten die Möglichkeit, mit C++ Windows .NET-Programme zu schreiben und die .NET Klassenbibliothek zu nutzen. Insbesondere kann man bestehende C++-Quelltexte in Windows .NET-Anwendungen aufzunehmen und so vorhandenen Code nutzen.

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 null-terminierten 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 vielen 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. Bitte senden Sie diese an die Mail-Adresse auf meiner Internetseite.

Bei meinen Schulungskunden und Studenten bedanke ich mich für die zahlreichen Anregungen und anregenden Fragen. Herrn Engesser und seinem Team vom Springer-Verlag danke ich für die Unterstützung und Geduld.

Tübingen, im Juni 2009

Richard Kaiser

Inhalt

Teil 1: Windows .NET Programme mit Visual Studio

1 Die Entwicklungsumgebung	1
1.1 Visuelle Programmierung: Ein erstes kleines Programm.....	1
1.2 Erste Schritte in C++.....	6
1.3 Der Quelltexteditor	8
1.4 Kontextmenüs und Symbolleisten (Toolbars).....	13
1.5 Projekte, Projektdateien und Projektoptionen.....	14
1.6 Einige Tipps zur Arbeit mit Projekten	16
1.7 Die Online-Hilfe (MSDN Dokumentation).....	19
1.8 Projektmappen und der Projektmappen-Explorer Θ	23
1.9 Hilfsmittel zur Gestaltung von Formularen Θ	24
1.10 Win32-, MFC- und Konsolen-Anwendungen Θ	25
1.10.1 Win32-Anwendungen Θ	26
1.10.2 MFC-Anwendungen Θ	26
1.10.3 Win32 Konsolen-Anwendungen Θ.....	29
1.10.4 CLR Konsolen-Anwendungen Θ	31
1.10.5 Der Start des Compilers von der Kommandozeile Θ.....	32
1.11 Setup-Projekte für den Windows-Installer Θ	32
1.11.1 .NET-Laufzeitbibliotheken: Das Redistributable-Package	33
1.11.2 Optionen für das Setup-Programm Θ	34

2	Steuerelemente für die Benutzeroberfläche	39
2.1	Die Online-Hilfe zu den Steuerelementen.....	39
2.1.1	Die Online-Hilfe über den Index	40
2.1.2	Die Online-Hilfe mit <i>FI</i>	43
2.2	Namen.....	44
2.3	Labels, Datentypen und Compiler-Fehlermeldungen.....	46
2.4	Funktionen, Methoden und die Komponente <i>TextBox</i>	52
2.4.1	Funktionen.....	53
2.4.2	Mehrzeilige <i>TextBox</i> en.....	57
2.5	Klassen, <i>ListBox</i> und <i>ComboBox</i>	59
2.6	Buttons und Ereignisse.....	64
2.6.1	Parameter der Ereignisbehandlungsroutinen	65
2.6.2	Der Fokus und die Tabulatorreihenfolge	67
2.6.3	Einige weitere Eigenschaften von Buttons	68
2.7	<i>CheckBox</i> en, <i>RadioButton</i> s und einfache <i>if</i> -Anweisungen.....	70
2.8	Container-Komponenten: <i>GroupBox</i> , <i>Panel</i> , <i>TabControl</i>	72
2.9	Hauptmenüs und Kontextmenüs.....	74
2.9.1	Hauptmenüs und der Menüdesigner	74
2.9.2	Kontextmenüs.....	77
2.10	Standarddialoge	78
2.11	Einfache Meldungen mit <i>MessageBox::Show</i> anzeigen.....	82
2.12	Eine Vorlage für viele Projekte und Übungsaufgaben	83

Teil 2: Standard-C++

3	Elementare Datentypen und Anweisungen.....	91
3.1	Syntaxregeln	91
3.2	Variablen und Bezeichner.....	95
3.3	Ganzzahldatentypen.....	98
3.3.1	Die interne Darstellung von Ganzzahlwerten	101
3.3.2	Ganzzahl литерale und ihr Datentyp	104
3.3.3	Zuweisungen und Standardkonversionen bei Ganzzahlausdrücken.....	106
3.3.4	Operatoren und die „üblichen arithmetischen Konversionen“.....	109
3.3.5	Der Datentyp <i>bool</i>	114
3.3.6	Die Datentypen <i>char</i> und <i>wchar_t (Char)</i>	119
3.3.7	C++/CLI-Ganzzahldatentypen: <i>Int32</i> , <i>Int64</i> usw.....	125
3.4	Kontrollstrukturen und Funktionen.....	126
3.4.1	Die <i>if</i> - und die Verbundanweisung	126
3.4.2	Wiederholungsanweisungen	131
3.4.3	Funktionen und der Datentyp <i>void</i>	134

3.4.4	Eine kleine Anleitung zum Erarbeiten der Lösungen	138
3.4.5	Werte- und Referenzparameter	142
3.4.6	Die Verwendung von Bibliotheken und Namensbereichen	142
3.4.7	Zufallszahlen	144
3.5	Tests und der integrierte Debugger	146
3.5.1	Systematisches Testen	146
3.5.2	Unit-Tests: Testfunktionen für automatisierte Tests	152
3.5.3	Unit-Tests in Visual Studio 2008	155
3.5.4	Der integrierte Debugger	158
3.6	Gleitkommatentypen	162
3.6.1	Die interne Darstellung von Gleitkommawerten	163
3.6.2	Der Datentyp von Gleitkommaliteralen	166
3.6.3	Standardkonversionen	167
3.6.4	Mathematische Funktionen in Standard-C++ und in .NET	172
3.6.5	C++/CLI-Gleitkommatentypen: <i>Single</i> , <i>Double</i> und <i>Decimal</i> ...	174
3.6.6	Datentypen für exakte und kaufmännische Rechnungen	175
3.6.7	Ein Kriterium für annähernd gleiche Gleitkommazahlen	183
3.7	Programmverifikation und Programmierlogik	185
3.7.1	Ablaufprotokolle	186
3.7.2	Schleifeninvarianten mit Ablaufprotokollen erkennen	189
3.7.3	Symbolische Ablaufprotokolle	193
3.7.4	Schleifeninvarianten und vollständige Induktion Θ	200
3.7.5	Verifikationen, Tests und Bedingungen zur Laufzeit prüfen	207
3.7.6	Funktionsaufrufe und Programmierstil für Funktionen	212
3.7.7	Einfache logische Regeln und Wahrheitstabellen Θ	219
3.7.8	Bedingungen in und nach <i>if</i> -Anweisungen und Schleifen Θ	221
3.8	Konstanten	230
3.9	Syntaxregeln für Deklarationen und Initialisierungen Θ	233
3.10	Arrays und Container	235
3.10.1	Einfache <i>typedef</i> -Deklarationen	236
3.10.2	Eindimensionale Arrays	236
3.10.3	Die Initialisierung von Arrays bei ihrer Definition	243
3.10.4	Arrays als Container	245
3.10.5	Mehrdimensionale Arrays	252
3.10.6	Dynamische Programmierung	256
3.11	Strukturen und Klassen	257
3.11.1	Mit <i>struct</i> definierte Klassen	258
3.11.2	C++/CLI-Erweiterungen: Mit <i>struct</i> definierte Werteklassen	264
3.11.3	Mit <i>union</i> definierte Klassen Θ	266
3.11.4	Bitfelder Θ	269
3.12	Zeiger, Strings und dynamisch erzeugte Variablen	271
3.12.1	Die Definition von Zeigervariablen	272
3.12.2	Der Adressoperator, Zuweisungen und generische Zeiger	275
3.12.3	Ablaufprotokolle für Zeigervariablen	279
3.12.4	Dynamisch erzeugte Variablen: <i>new</i> und <i>delete</i>	280
3.12.5	Garbage Collection mit der Smart Pointer Klasse <i>shared_ptr</i>	290
3.12.6	Dynamisch erzeugte eindimensionale Arrays	292

3.12.7	Arrays, Zeiger und Zeigerarithmetik	294
3.12.8	Arrays als Funktionsparameter Θ	298
3.12.9	Konstante Zeiger	301
3.12.10	Stringlitterale, nullterminierte Strings und <i>char*</i> -Zeiger	303
3.12.11	Konversionen zwischen <i>char*</i> und <i>String</i>	306
3.12.12	Verkettete Listen	311
3.12.13	Binärbäume	322
3.12.14	Zeiger als Parameter Θ	325
3.12.15	C++/CLI-Erweiterungen: Garbage Collection und der GC-Heap	327
3.12.16	C-Bibliotheksfunktionen in <i>string.h</i> für nullterminierte Strings Θ	329
3.12.17	Die „Secure Library Functions“ Θ	333
3.12.18	Zeiger auf Zeiger auf Zeiger auf ... Θ	334
3.13	C++/CLI-Erweiterungen: Die Stringklasse <i>String</i>	335
3.13.1	Die Definition von Variablen eines Klassentyps	336
3.13.2	Elementfunktionen der Klasse <i>String</i>	338
3.14	Deklarationen mit <i>typedef</i> und <i>typeid</i> -Ausdrücke	342
3.15	Aufzählungstypen	344
3.15.1	<i>enum</i> Konstanten und Konversionen Θ	346
3.15.2	C++/CLI-Aufzählungstypen Θ	348
3.16	Kommentare und interne Programmdokumentation	348
3.16.1	Kommentare zur internen Dokumentation	349
3.16.2	Dokumentationskommentare	352
3.17	Globale, lokale und dynamische Variablen	352
3.17.1	Die Deklarationsanweisung	352
3.17.2	Die Verbundanweisung und der lokale Gültigkeitsbereich	353
3.17.3	Statische lokale Variablen	355
3.17.4	Lebensdauer von Variablen und Speicherklassenspezifizierer Θ	355
3.18	Referenztypen, Werte- und Referenzparameter	358
3.18.1	Werteparameter	359
3.18.2	Referenzparameter	360
3.18.3	Konstante Referenzparameter	363
3.19	Weitere Anweisungen	365
3.19.1	Die Ausdrucksanweisung	366
3.19.2	Exception-Handling: <i>try</i> und <i>throw</i>	368
3.19.3	Die <i>switch</i> -Anweisung Θ	372
3.19.4	Die <i>do</i> -Anweisung Θ	375
3.19.5	Die <i>for</i> -Anweisung Θ	376
3.19.6	Die Sprunganweisungen <i>goto</i> , <i>break</i> und <i>continue</i> Θ	378
3.19.7	Assembler-Anweisungen Θ	381
3.20	Ausdrücke	382
3.20.1	Primäre Ausdrücke Θ	383
3.20.2	Postfix-Ausdrücke Θ	385
3.20.3	Unäre Ausdrücke Θ	386
3.20.4	Typkonversionen in Typecast-Schreibweise Θ	389
3.20.5	Zeiger auf Klasselemente Θ	389
3.20.6	Multiplikative Operatoren Θ	389
3.20.7	Additive Operatoren Θ	390

3.20.8	Shift-Operatoren Θ	390
3.20.9	Vergleichsoperatoren Θ	391
3.20.10	Gleichheitsoperatoren Θ	392
3.20.11	Bitweise Operatoren Θ	393
3.20.12	Logische Operatoren Θ	394
3.20.13	Der Bedingungsoperator Θ	394
3.20.14	Konstante Ausdrücke Θ	396
3.20.15	Zuweisungsoperatoren	396
3.20.16	Der Komma-Operator Θ	397
3.20.17	L-Werte und R-Werte Θ	399
3.20.18	Die Priorität und Assoziativität der Operatoren Θ	399
3.20.19	Alternative Zeichenfolgen Θ	402
3.20.20	Explizite Typkonversionen Θ	403
3.21	Namensbereiche	410
3.21.1	Die Definition von benannten Namensbereichen	411
3.21.2	Die Verwendung von Namen aus Namensbereichen	413
3.21.3	Header-Dateien und Namensbereiche	416
3.21.4	Aliasnamen für Namensbereiche	418
3.21.5	Unbenannte Namensbereiche	419
3.22	Präprozessoranweisungen	421
3.22.1	Die <code>#include</code> -Anweisung	422
3.22.2	Makros Θ	423
3.22.3	Bedingte Kompilation Θ	428
3.22.4	Pragmas Θ	434
3.23	Separate Kompilation und statische Bibliotheken	437
3.23.1	C++-Dateien, Header-Dateien und Object-Dateien	437
3.23.2	Bindung Θ	439
3.23.3	Deklarationen und Definitionen Θ	441
3.23.4	Die „One Definition Rule“ Θ	443
3.23.5	Die Elemente von Header-Dateien und C++-Dateien Θ	445
3.23.6	Object-Dateien und statische Bibliotheken linken Θ	447
3.23.7	Der Aufruf von in C geschriebenen Funktionen Θ	447
4	Die C++-Standardbibliothek	449
4.1	Die Stringklassen <i>string</i> und <i>wstring</i>	449
4.1.1	Gemeinsamkeiten und Unterschiede der Stringklassen	450
4.1.2	Konversionen zwischen <i>string</i> und <i>String</i>	452
4.1.3	Einige Elementfunktionen der Klasse <i>string</i>	453
4.1.4	Stringstreams	457
4.2	Sequenzielle Container der Standardbibliothek	463
4.2.1	Die Container-Klasse <i>vector</i>	463
4.2.2	Iteratoren	466
4.2.3	Die STL/CLR Containerklassen in Visual C++ 2008	470
4.2.4	Algorithmen der Standardbibliothek	473
4.2.5	Die Speicherverwaltung bei Vektoren Θ	480

4.2.6	Mehrdimensionale Vektoren Θ	482
4.2.7	Die Container-Klassen <i>list</i> und <i>deque</i>	484
4.2.8	Gemeinsamkeiten und Unterschiede der sequenziellen Container.....	485
4.2.9	Die Container-Adapter <i>stack</i> , <i>queue</i> und <i>priority_queue</i> Θ	487
4.2.10	Container mit Zeigern.....	489
4.2.11	Geprüfte Iteratoren (Checked Iterators).....	490
4.2.12	Die Container-Klasse <i>bitset</i> Θ	492
4.3	Dateibearbeitung mit den Stream-Klassen	494
4.3.1	Stream-Variablen, ihre Verbindung mit Dateien und ihr Zustand	494
4.3.2	Fehler und der Zustand von Stream-Variablen	499
4.3.3	Lesen und Schreiben von Binärdaten mit <i>read</i> und <i>write</i>	500
4.3.4	Lesen und Schreiben von Daten mit den Operatoren << und >>.....	509
4.3.5	Manipulatoren und Funktionen zur Formatierung von Texten Θ	517
4.3.6	Dateibearbeitung im Direktzugriff Θ	519
4.3.7	C-Funktionen zur Dateibearbeitung Θ	523
4.4	Assoziative Container	526
4.4.1	Die Container <i>set</i> und <i>multiset</i>	526
4.4.2	Die Container <i>map</i> und <i>multimap</i>	528
4.4.3	Iteratoren der assoziativen Container	530
4.5	Die numerischen Klassen der Standardbibliothek.....	533
4.5.1	Komplexe Zahlen Θ	533
4.5.2	Valarrays Θ	536
4.6	TR1-Erweiterungen der Standardbibliothek Θ	538
4.6.1	Ungeordnete Assoziative Container (Hash-Container).....	539
4.6.2	Reguläre Ausdrücke mit <regex> Θ	542
4.6.3	Fixed Size Array Container mit <array> Θ	545
4.6.4	Tupel mit <tuple> Θ	547
4.6.5	Zufallszahlen mit <random> Θ	548
4.7	Die Boost-Bibliotheken Θ	550

5 Funktionen..... 551

5.1	Die Verwaltung von Funktionsaufrufen über den Stack.....	552
5.1.1	Aufrufkonventionen Θ	555
5.2	Funktionszeiger und der Datentyp einer Funktion	555
5.2.1	Der Datentyp einer Funktion	555
5.2.2	Zeiger auf Funktionen.....	557
5.3	Rekursion	563
5.3.1	Grundlagen	564
5.3.2	Quicksort	569
5.3.3	Ein rekursiv absteigender Parser	573
5.3.4	Rekursiv definierte Kurven Θ	578
5.3.5	Indirekte Rekursion Θ	581
5.3.6	Rekursive Datenstrukturen und binäre Suchbäume	581
5.3.7	Verzeichnisse rekursiv nach Dateien durchsuchen Θ	585
5.4	Die Funktionen <i>main</i> und <i>_tmain</i> und ihre Parameter.....	587

5.5	Default-Argumente	588
5.6	Inline-Funktionen.....	590
5.7	Überladene Funktionen.....	593
5.7.1	Funktionen, die nicht überladen werden können	594
5.7.2	Regeln für die Auswahl einer passenden Funktion.....	596
5.8	Überladene Operatoren mit globalen Operatorfunktionen	602
5.8.1	Globale Operatorfunktionen	604
5.8.2	Die Inkrement- und Dekrementoperatoren	605
5.8.3	Referenzen als Funktionswerte	607
5.8.4	Die Ein- und Ausgabe von selbst definierten Datentypen	609

6 Objektorientierte Programmierung..... 613

6.1	Klassen.....	614
6.1.1	Datenelemente und Elementfunktionen	614
6.1.2	Der Gültigkeitsbereich von Klasselementen	622
6.1.3	Datenkapselung: Die Zugriffsrechte <i>private</i> und <i>public</i>	626
6.1.4	Der Aufruf von Elementfunktionen und der <i>this</i> -Zeiger	632
6.1.5	Konstruktoren und Destruktoren	634
6.1.6	OO Analyse und Design: Der Entwurf von Klassen.....	645
6.1.7	Programmierlogik: Klasseninvarianten und Korrektheit	653
6.1.8	UML-Diagramme und Klassendiagramme in Visual Studio 2008	661
6.2	Klassen als Datentypen	664
6.2.1	Der Standardkonstruktor.....	664
6.2.2	Objekte als Klasselemente und Elementinitialisierer	667
6.2.3	<i>friend</i> -Funktionen und -Klassen	672
6.2.4	Überladene Operatoren als Elementfunktionen	676
6.2.5	Der Kopierkonstruktor.....	684
6.2.6	Der Zuweisungsoperator = für Klassen	690
6.2.7	Benutzerdefinierte Konversionen Θ	700
6.2.8	Explizite Konstruktoren.....	704
6.2.9	Statische Klasselemente.....	705
6.2.10	Konstante Klasselemente und Objekte.....	708
6.2.11	Klassen und Header-Dateien	711
6.3	Vererbung und Komposition.....	713
6.3.1	Die Elemente von abgeleiteten Klassen.....	713
6.3.2	Zugriffsrechte auf die Elemente von Basisklassen.....	715
6.3.3	Die Bedeutung von Elementnamen in einer Klassenhierarchie	717
6.3.4	<i>using</i> -Deklarationen in abgeleiteten Klassen Θ	719
6.3.5	Konstruktoren, Destruktoren und implizit erzeugte Funktionen.....	720
6.3.6	Vererbung bei Formularen in Visual Studio	726
6.3.7	OO Design: <i>public</i> Vererbung und „ist ein“-Beziehungen	727
6.3.8	OO Design: Komposition und „hat ein“-Beziehungen	732
6.3.9	Konversionen zwischen <i>public</i> abgeleiteten Klassen.....	734
6.3.10	<i>protected</i> und <i>private</i> abgeleitete Klassen Θ	739
6.3.11	Mehrfachvererbung und virtuelle Basisklassen	742

6.4	Virtuelle Funktionen, späte Bindung und Polymorphie	749
6.4.1	Der statische und der dynamische Datentyp	749
6.4.2	Virtuelle Funktionen	750
6.4.3	Die Implementierung von virtuellen Funktionen: <i>vp</i> tr und <i>vt</i> bl	757
6.4.4	Virtuelle Konstruktoren und Destruktoren	766
6.4.5	Virtuelle Funktionen in Konstruktoren und Destruktoren	768
6.4.6	OO-Design: Einsatzbereich und Test von virtuellen Funktionen	769
6.4.7	OO-Design und Erweiterbarkeit	771
6.4.8	Rein virtuelle Funktionen und abstrakte Basisklassen	775
6.4.9	OO-Design: Virtuelle Funktionen und abstrakte Basisklassen	778
6.4.10	OOAD: Zusammenfassung	781
6.4.11	Interfaces und Mehrfachvererbung	785
6.4.12	Zeiger auf Klasselemente Θ	786
6.4.13	UML-Diagramme für Vererbung und Komposition	791
6.5	Laufzeit-Typinformationen	793
6.5.1	Typinformationen mit dem Operator <i>typeid</i> Θ	794
6.5.2	Typkonversionen mit <i>dynamic_cast</i> Θ	797
6.5.3	Anwendungen von Laufzeit-Typinformationen Θ	800
6.5.4	<i>static_cast</i> mit Klassen Θ	802
7	Exception-Handling	807
7.1	Die <i>try</i> -Anweisung	808
7.2	Exception-Handler und Exceptions der Standardbibliothek	813
7.3	Einige vordefinierte C++/CLI und .NET Exceptions	817
7.4	<i>throw</i> -Ausdrücke und selbst definierte Exceptions	818
7.5	Fehler und Exceptions	824
7.6	Die Freigabe von Ressourcen bei Exceptions	827
7.6.1	Ressource Aquisition is Initialization (RAII)	828
7.6.2	Die Smart Pointer Klasse <i>shared_ptr</i>	829
7.7	Exceptions in Konstruktoren und Destruktoren	829
7.8	Exception-Spezifikationen Θ	834
7.9	Die Funktion <i>terminate</i> Θ	836
8	Templates und die STL	837
8.1	Generische Funktionen: Funktions-Templates	838
8.1.1	Die Deklaration von Funktions-Templates mit Typ-Parametern	839
8.1.2	Spezialisierungen von Funktions-Templates	840
8.1.3	Funktions-Templates mit Nicht-Typ-Parametern	847
8.1.4	Explizit instanziierte Funktions-Templates Θ	849
8.1.5	Explizit spezialisierte und überladene Templates	850
8.1.6	Rekursive Funktions-Templates Θ	853
8.2	Generische Klassen: Klassen-Templates	857
8.2.1	Die Deklaration von Klassen-Templates mit Typ-Parametern	858

8.2.2	Spezialisierungen von Klassen-Templates.....	858
8.2.3	Templates mit Nicht-Typ-Parametern	866
8.2.4	Explizit instanziierte Klassen-Templates Θ	868
8.2.5	Partielle und vollständige Spezialisierungen Θ	868
8.2.6	TR1-Erweiterungen: Type Traits Θ	875
8.2.7	Elemente und <i>friend</i> -Funktionen von Klassen-Templates Θ	878
8.2.8	Ableitungen von Templates Θ	881
8.3	Funktionsobjekte in der STL.....	884
8.3.1	Der Aufrufoperator ()	885
8.3.2	Prädikate und arithmetische Funktionsobjekte	888
8.3.3	Binder, Funktionsadapter und TR1-Erweiterungen	892
8.4	Iteratoren und die STL-Algorithmen.....	899
8.4.1	Die verschiedenen Arten von Iteratoren	900
8.4.2	Umkehriteratoren.....	902
8.4.3	Einfügefunktionen und Einfügeiteratoren.....	903
8.4.4	Stream-Iteratoren.....	905
8.4.5	Container-Konstrukturen mit Iteratoren	906
8.4.6	STL-Algorithmen für alle Elemente eines Containers	907
8.5	Die Algorithmen der STL	910
8.5.1	Lineares Suchen.....	911
8.5.2	Zählen.....	912
8.5.3	Der Vergleich von Bereichen	913
8.5.4	Suche nach Teilfolgen	915
8.5.5	Minimum und Maximum.....	916
8.5.6	Elemente vertauschen	917
8.5.7	Kopieren von Bereichen.....	918
8.5.8	Elemente transformieren und ersetzen.....	919
8.5.9	Elementen in einem Bereich Werte zuweisen.....	921
8.5.10	Elemente entfernen	922
8.5.11	Die Reihenfolge von Elementen vertauschen	923
8.5.12	Permutationen.....	924
8.5.13	Partitionen	925
8.5.14	Bereiche sortieren.....	926
8.5.15	Binäres Suchen in sortierten Bereichen	927
8.5.16	Mischen von sortierten Bereichen	929
8.5.17	Mengenoperationen auf sortierten Bereichen	930
8.5.18	Heap-Operationen.....	931
8.5.19	Verallgemeinerte numerische Operationen.....	932

Teil 3: C++/CLI und die .NET Bibliothek

9 C++/CLI	937
9.1 Assemblies, CLR-Optionen und DLLs	939
9.1.1 CLR-Projekte.....	939
9.1.2 CLR-Optionen	941
9.1.3 CLR-DLLs.....	942
9.1.4 Assembly-bezogene Zugriffsrechte	944
9.1.5 Win32-DLLs erzeugen Θ	945
9.1.6 Win32-DLLs und Win32-API Funktionen in CLR-Projekten Θ ...	946
9.2 Garbage Collection und der GC-Heap	950
9.3 Die Basisklasse <i>System::Object</i> und ihre Methoden	956
9.4 Die Stringklasse <i>String</i>	959
9.4.1 Einige Konstruktoren der Klasse <i>String</i>	959
9.4.2 Elementfunktionen der Klasse <i>String</i>	960
9.4.3 Konvertierungs- und Formatierungsfunktionen	966
9.4.4 Die Klasse <i>StringBuilder</i>	970
9.4.5 Die Zeichen eines String: Die Klasse <i>System::Char</i>	971
9.5 CLI-Arrays.....	972
9.6 Verweisklassen.....	975
9.6.1 Elemente von Verweisklassen	976
9.6.2 Objekte, „heap -“ und „stack semantics“	978
9.6.3 Der Kopierkonstruktor und Zuweisungsoperator.....	980
9.6.4 Statische Elemente.....	983
9.6.5 Konstante Datenelemente mit <i>static const</i> , <i>literal</i> und <i>initonly</i>	986
9.6.6 Statische Konstruktoren.....	988
9.6.7 Statische Operatoren.....	989
9.6.8 Gleichheit bei Verweisklassen: <i>Equals</i> und <i>operator==</i>	990
9.6.9 Typkonversionen mit <i>safe_cast</i> , <i>static_cast</i> und <i>dynamic_cast</i>	993
9.6.10 <i>explicit</i> bei Konstruktoren und Konversionsfunktionen Θ	994
9.6.11 Destruktoren und Finalisierer	996
9.6.12 Vererbung bei Verweisklassen	1001
9.6.13 Konstruktoraufrufe in einer Hierarchie von Verweisklassen	1002
9.6.14 Virtuelle Funktionen, <i>new</i> und <i>override</i>	1004
9.6.15 Abstrakte Klassen und Elementfunktionen	1006
9.6.16 Versiegelte (<i>sealed</i>) Klassen und Elementfunktionen	1006
9.7 Werttypen und Wertklassen.....	1007
9.7.1 Fundamentale Typen	1007
9.7.2 C++/CLI-Aufzählungstypen	1009
9.7.3 Wertklassen	1010
9.7.4 Vererbung bei Wertklassen.....	1014
9.8 Interface-Klassen	1014
9.8.1 Die .NET Interface-Klasse <i>Comparable</i>	1017

9.8.2	Collection-Typen, <i>IEnumerable</i> und die <i>for each</i> -Anweisung	1018
9.8.3	<i>IFormattable</i> und die Darstellung selbstdefinierter Datentypen	1020
9.9	C++/CLI Exception-Handling	1022
9.9.1	Unterschiede zu Standard-C++	1022
9.9.2	Die Basisklasse <i>Exception</i>	1023
9.9.3	Vordefinierte C++/CLI und .NET Exceptions	1025
9.9.4	Die Freigabe von Ressourcen mit <i>try-finally</i>	1026
9.9.5	Nicht behandelte Exceptions in Windows Forms-Anwendungen	1028
9.9.6	Die Protokollierung von Exceptions in einem EventLog	1029
9.10	C++/CLI-Erweiterungen für native Klassen	1030
9.11	Parameter-Arrays	1031
9.12	Visuelle Programmierung und Properties (Eigenschaften)	1032
9.12.1	Lesen und Schreiben von Eigenschaften	1032
9.12.2	Indizierte Properties	1035
9.12.3	Properties und Vererbung	1037
9.13	Delegat-Typen und Events	1039
9.13.1	Ausgangspunkt: Funktionszeiger	1040
9.13.2	Delegat-Typen und -Instanzen	1041
9.13.3	Ereignisse (events)	1046
9.13.4	Selbst definierte Komponenten und ihre Ereignisse	1048
9.13.5	Nichttriviale und Statische Ereignisse	1053
9.14	Ein kleiner Überblick über die .NET Klassenbibliothek	1054
9.14.1	Komponenten und die Klasse <i>Component</i>	1054
9.14.2	Steuerelemente und die Klasse <i>Control</i>	1055
9.14.3	Verschachtelte Controls: Die Klasse <i>ControlCollection</i>	1058
9.14.4	Botschaften für ein Steuerelement	1060
9.14.5	Botschaften für eine Anwendung	1061
9.15	Steuerelementbibliotheken: Die Erweiterung der Toolbox	1062
9.15.1	Die Erweiterung der Toolbox um selbstdefinierte Komponenten	1063
9.15.2	Klassen für selbstdefinierte Komponenten	1066
9.15.3	Beispiel: Eine selbstdefinierte Tachometer-Komponente	1067
9.15.4	Attribute für selbstdefinierte Komponenten	1069
9.16	Laufzeit-Typinformationen und Reflektion	1071
9.16.1	Laufzeit-Typinformationen der Klasse <i>Type</i>	1071
9.16.2	Reflektion mit der Klasse <i>Assembly</i>	1073
9.16.3	Dynamisch erzeugte Datentypen und Plugins	1074
9.17	Attribute	1077
9.17.1	Vordefinierte Attribute	1079
9.17.2	Selbstdefinierte Laufzeitattribute	1080
9.18	Generische Programmierung	1084
9.18.1	<i>generic</i> und <i>template</i> : Gemeinsamkeiten und Unterschiede	1084
9.18.2	Typparameter-Einschränkungen (Constraints)	1088
9.19	Dokumentationskommentare und CHM-Hilfdateien	1090
9.19.1	Dokumentationskommentare und XML-Dateien	1091
9.19.2	Aus XML-Dateien Hilfdateien im CHM-Format erzeugen	1095
9.20	Managed C++ und C++/CLI Θ	1096

10 Einige Elemente der .NET-Klassenbibliothek.....	1099
10.1 Formatierte Texte mit RichTextBox	1099
10.2 Steuerelemente zur Eingabe und Prüfung von Daten	1102
10.2.1 Fehleranzeigen mit <i>ErrorProvider</i>	1102
10.2.2 Weitere Formulare und selbstdefinierte Dialoge anzeigen	1103
10.2.3 Das <i>Validating</i> -Ereignis	1107
10.2.4 Texteingaben mit einer <i>MaskedTextBox</i> filtern	1109
10.2.5 Tastatureingaben filtern mit dem <i>KeyPress</i> -Ereignis.....	1112
10.2.6 Hilfe-Informationen mit <i>ToolTip</i> und <i>HelpProvider</i>	1113
10.2.7 Auf/Ab-Steuerelemente	1114
10.2.8 Schieberegler: <i>VScrollBar</i> und <i>HScrollBar</i>	1116
10.2.9 Lokalisierung	1118
10.3 Symbolleisten, Status- und Fortschrittsanzeigen.....	1121
10.3.1 Symbolleisten mit Panels und Buttons.....	1121
10.3.2 Status- und Fortschrittsanzeigen	1121
10.3.3 Symbolleisten mit <i>ToolStrip</i>	1123
10.3.4 <i>ToolStripContainer</i>	1124
10.3.5 <i>ToolStripLabel</i> und <i>LinkLabel</i>	1126
10.3.6 <i>NotifyIcon</i> : Elemente im Infobereich der Taskleiste	1127
10.4 Größenänderung von Steuerelementen zur Laufzeit	1129
10.4.1 Die Eigenschaften <i>Dock</i> und <i>Anchor</i>	1129
10.4.2 <i>SplitContainer</i> : Zur Größenanpassung von zwei Panels	1129
10.4.3 <i>TableLayoutPanel</i> : Tabellen mit Steuerelementen Θ	1130
10.4.4 Automatisch angeordnete Steuerelemente: <i>FlowLayoutPanel</i> Θ ..	1132
10.5 <i>ImageList</i> , <i>ListView</i> und <i>TreeView</i>	1133
10.5.1 Die Verwaltung von Bildern mit einer <i>ImageList</i>	1133
10.5.2 Die Anzeige von Listen mit <i>ListView</i>	1134
10.5.3 <i>ListView</i> nach Spalten sortieren	1138
10.5.4 Die Anzeige von Baumstrukturen mit <i>TreeView</i>	1140
10.6 Die Erweiterung der Toolbox	1146
10.7 MDI-Programme.....	1148
10.8 Uhrzeiten, Kalenderdaten und Timer	1151
10.8.1 Die Klassen <i>DateTime</i> und <i>TimeSpan</i>	1151
10.8.2 Steuerelemente zur Eingabe von Kalenderdaten und Zeiten.....	1154
10.8.3 Timer und zeitgesteuerte Ereignisse	1155
10.8.4 Hochauflösende Zeitmessung mit der Klasse <i>Stopwatch</i>	1158
10.8.5 Kulturspezifische Datumsformate und Kalender	1159
10.8.6 Kalenderklassen.....	1162
10.9 Asynchrone Programmierung und Threads.....	1163
10.9.1 Die verschiedenen Thread-Zustände	1165
10.9.2 Multithreading mit der Klasse <i>BackgroundWorker</i>	1165
10.9.3 Ereignisbasierte asynchrone Programmierung.....	1169
10.9.4 Die Klasse <i>Thread</i> und der Zugriff auf Steuerelemente	1170
10.9.5 <i>AsyncResult</i> -basierte asynchrone Programmierung	1175
10.9.6 <i>Sleep</i> und Threads	1178
10.9.7 Kritische Abschnitte und die Synchronisation von Threads	1179

10.9.8	ThreadPool	1185
10.10	Grafiken zeichnen mit <i>PictureBox</i> und <i>Graphics</i>	1187
10.10.1	Grafiken mit einer <i>PictureBox</i> anzeigen	1187
10.10.2	Grafiken auf einer <i>PictureBox</i> und anderen Controls zeichnen	1187
10.10.3	Welt- und Bildschirmkoordinaten	1189
10.10.4	Figuren	1193
10.10.5	Farben, Stifte und Pinsel	1193
10.10.6	Text zeichnen	1195
10.10.7	Drucken mit <i>Graphics</i>	1196
10.11	Die Steuerung von MS-Office 2003 Anwendungen	1203
10.11.1	Word	1203
10.11.2	Excel	1206
10.12	Collection-Klassen	1208
10.12.1	Generische und nicht generische Collection-Klassen	1209
10.12.2	Generische Interface-Klassen: <i>ICollection<T></i> und <i>IList<T></i>	1210
10.12.3	Die generische Collection-Klasse <i>List<T></i>	1212
10.12.4	<i>DataGridView</i> : Die Anzeige von Daten in einer Tabelle	1214
10.12.5	Die generische Collection-Klasse <i>Queue<T></i>	1217
10.12.6	Die generische Collection-Klasse <i>HashSet<T></i>	1217
10.12.7	Die generische Collection-Klasse <i>LinkedList<T></i>	1218
10.12.8	Die generische Collection-Klasse <i>Stack<T></i>	1219
10.12.9	Dictionaries und die generische Interface-Klasse <i>IDictionary</i>	1220
10.12.10	Spezielle Collection-Klassen	1225
10.13	Systeminformationen und -operationen	1226
10.13.1	Die Klasse <i>Environment</i>	1226
10.13.2	Die Klasse <i>Process</i>	1227
10.13.3	Die Klasse <i>Clipboard</i>	1229
10.14	.NET-Klassen zur Dateibearbeitung	1229
10.14.1	Textdateien bearbeiten: <i>StreamReader</i> und <i>StreamWriter</i>	1230
10.14.2	Die Klasse <i>FileStream</i>	1234
10.14.3	<i>BinaryReader/Writer</i> und <i>StreamReader/Writer</i> mit <i>FileStreams</i>	1236
10.14.4	Der gleichzeitige Zugriff auf eine Datei und Record-Locking	1237
10.14.5	XML-Dateien: Die Klassen <i>XmlReader</i> und <i>XmlWriter</i>	1238
10.14.6	Klassen für Laufwerke, Verzeichnisse, Pfade und Dateien	1241
10.14.7	Die Klasse <i>FileSystemWatcher</i>	1248
10.14.8	Komprimieren und Dekomprimieren von Dateien	1248
10.15	Serialisierung	1250
10.15.1	Serialisierung mit <i>BinaryFormatter</i>	1252
10.15.2	Serialisierung mit <i>SoapFormatter</i>	1255
10.15.3	Serialisierung mit <i>XmlSerializer</i>	1256
10.16	Datenbanken	1259
10.16.1	Eine Datenbank anlegen	1260
10.16.2	Die Verbindung zu einer Datenbank herstellen	1265
10.16.3	SQL-Anweisungen	1269
10.16.4	Die Klasse <i>DataTable</i> und die Anzeige in einem <i>DataGridView</i>	1273
10.16.5	Die Klasse <i>DataSet</i>	1277
10.16.6	Datenbanken als XML-Dateien lesen und schreiben	1278

10.16.7 Gekoppelte Datenquellen: Master/Detail DataGridViews.....	1278
10.16.8 Datenquellen in Visual C++ 2005 und in Visual C# 2008 Θ.....	1280
10.17 Datenbindung	1281
10.17.1 Komplexe Datenbindung	1281
10.17.2 BindingSource	1283
10.17.3 Einfache Datenbindung	1286
10.18 Reguläre Ausdrücke	1287
10.19 Internet-Komponenten.....	1294
10.19.1 Die WebBrowser-Komponente der Toolbox.....	1294
10.19.2 Up- und Downloads mit der Klasse <i>WebClient</i>	1295
10.19.3 E-Mails versenden mit <i>SmtpClient</i>	1297
10.19.4 Netzwerkinformationen und die Klasse <i>Ping</i>	1298
10.19.5 TCP-Clients und Server mit <i>TcpClient</i> und <i>TcpListener</i>	1299
Literaturverzeichnis.....	1303
Inhalt Buch-CD	1309
Index	1311

Θ Angesichts des Umfangs dieses Buches habe ich einige Abschnitte mit dem Zeichen Θ 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.

Teil 1

Windows .NET Programme

mit Visual Studio

1 Die Entwicklungsumgebung

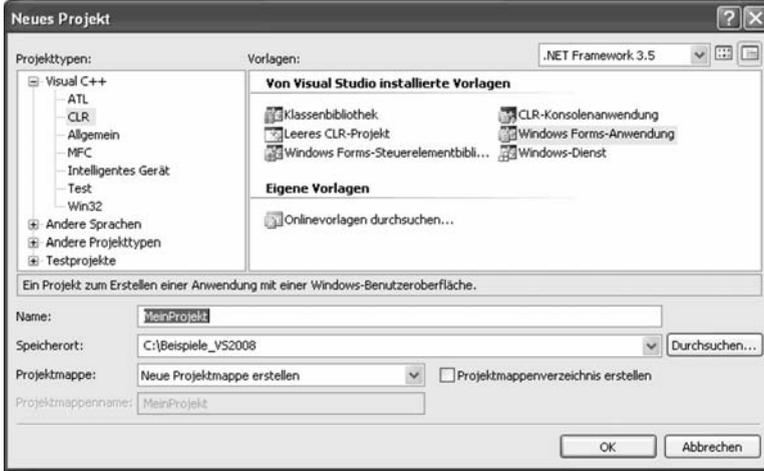
Visual Studio besteht aus verschiedenen Werkzeugen (Tools), die einen Programmierer bei der Entwicklung von Software unterstützen. Eine solche Zusammenstellung von Werkzeugen zur Softwareentwicklung bezeichnet man auch als Programmier- oder **Entwicklungsumgebung**.

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 in Visual Studio in einem einzigen Programm integriert sind, spricht man auch von einer **integrierten Entwicklungsumgebung** (engl.: „integrated development environment“, **IDE**).

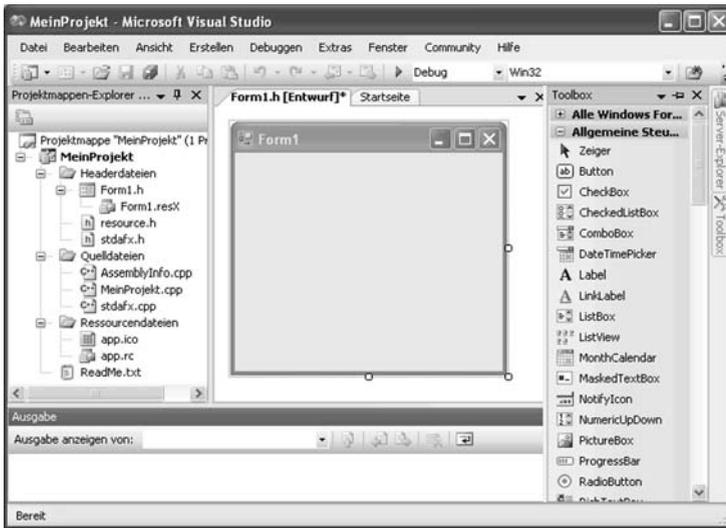
In diesem Kapitel wird zunächst an einfachen Beispielen gezeigt, wie man mit Visual Studio 2008 Windows-Programme mit einer grafischen Benutzeroberfläche entwickeln kann. Anschließend (ab Abschnitt 1.3) werden dann die wichtigsten Werkzeuge von Visual Studio ausführlicher vorgestellt. Für viele einfache 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 xxiv) 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

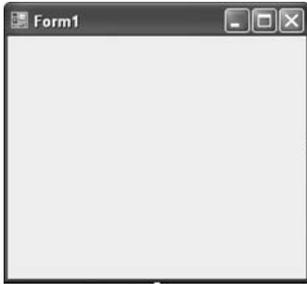
In Visual Studio 2008 findet man unter **Datei\Neu\Projekt** Vorlagen für verschiedene Arten von Anwendungen. Eine Windows-Anwendung mit einer grafischen Benutzeroberfläche erhält man mit dem Projekttyp **CLR** (Common Language Runtime) und der Vorlage **Windows Forms-Anwendung**. Ein solches Projekt legt man dann an, indem man nach *Name* einen Namen und nach *Speicherort* ein Verzeichnis für das Projekt eingibt und dann den OK-Button anklickt:



Anschließend werden einige Tools der IDE angezeigt. Die hier rechts eingblendete **Toolbox** wird angezeigt, wenn man mit der Maus über den Toolbox-Rider fährt, oder mit *Ansicht/Toolbox*:



Das **Formular** (hier *Form1*) ist der Ausgangspunkt für alle Windows Forms Anwendungen, die mit Visual Studio 2008 entwickelt werden. Es entspricht dem Fenster, das beim Start des Programms angezeigt wird:

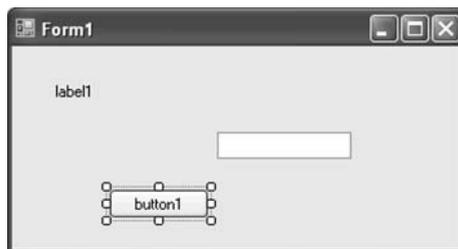


Ein Formular kann mit den in der **Toolbox** verfügbaren **Steuerelementen (Controls)** gestaltet werden. Die Toolbox zeigt praktisch alle der unter Windows üblichen Steuerelemente an, wenn das Formular angezeigt wird. Sie sind auf verschiedene Gruppen verteilt (z.B. *Allgemeine Steuerelemente*, *Container* usw.), 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 ein *Timer* aus der Gruppe *Komponenten*, sind im laufenden Programm nicht sichtbar.

Falls Ihnen die Namen und die kleinen Icons nicht allzu viel sagen, lassen Sie einfach den Mauszeiger kurz auf einer Komponente stehen: Dann erscheint ein kleines gelbes Hinweisfenster mit einer kurzen Beschreibung.

Um eine **Komponente** aus der Toolbox **auf das Formular** zu **setzen**, zieht man sie einfach von der Toolbox auf das Formular. Oder man klickt mit der Maus zuerst auf die Komponente (sie wird dann als markiert dargestellt) und dann auf die Stelle im Formular, an die ihre linke obere Ecke kommen soll.

Beispiel: Nachdem man ein Label (Zeile sieben in *Allgemeine Steuerelemente*, mit dem großen A), eine TextBox (vierte Zeile von unten, Aufschrift *ab*) und einen Button (zweite Zeile mit der Aufschrift *ab*) 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 *Debuggen\Debuggen starten* von der Menüleiste, oder
- mit *F5* von einem beliebigen Fenster in Visual Studio 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.

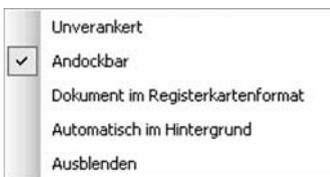
Bemerkenswert an diesem Programm ist vor allem der im Vergleich zu einem nichtvisuellen Entwicklungssystem **geringe Aufwand**, mit dem es erstellt wurde. So braucht Petzold in seinem Klassiker „Programmierung unter Windows“ (Petzold 1992, S. 33) ca. 80 Zeilen nichttriviale C-Anweisungen, um den Text „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 Programmiersprache entwickelt, wird es bei der visuellen Programmierung ganz oder teilweise aus vorgefertigten grafischen Komponenten zusammengesetzt.

Mit Visual Studio 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 Benutzereingaben (Mausklicks usw.) erfolgen sollen, werden dagegen konventionell in einer Programmiersprache (z.B. C++) geschrieben.

Wenn die Toolbox oft verwendet wird, ist es am einfachsten, sie in der Entwicklungsumgebung zu fixieren, indem man in ihrem Kontextmenü (rechte Maustaste) „Automatisch im Hintergrund“ deaktiviert.

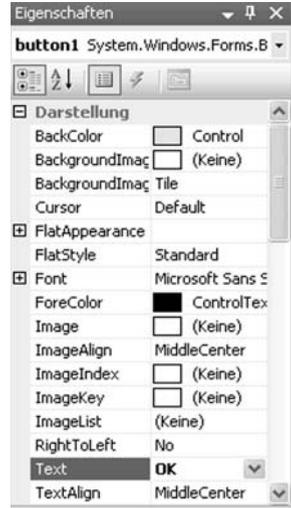


Die zuletzt auf einem Formular (bzw. im Pull-down-Menü des Eigenschaftensfensters) angeklickte Komponente wird als die **aktuell ausgewählte Komponente**

bezeichnet. Man erkennt sie an den kleinen Quadraten an ihrem Rand, den sogenannten **Ziehquadraten**. An ihnen kann man mit der Maus ziehen und so die Größe der Komponente verändern. Ein **Formular** wird dadurch zur aktuell ausgewählten Komponente, dass man mit der Maus eine freie Stelle im Formular anklickt.

Beispiel: Im letzten Beispiel ist *button1* die aktuell ausgewählte Komponente.

Im **Eigenschaftfenster** (Kontextmenü der Komponente auf dem Formular, oder *Ansicht|Weitere Fenster|Eigenschaftfenster*, nicht mit *Ansicht|Eigenschaftfenster-Manager* verwechseln) werden die Eigenschaften (properties) der aktuell ausgewählten Komponente angezeigt. 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 Spalte 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 **Text** 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 **BackColor** (z.B. bei einem Button) kann man über ein Pull-down-Menü die **Hintergrundfarbe** auswählen.

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

Eine Komponente auf dem Formular wird nicht nur an ihre Eigenschaften im Eigenschaftfenster angepasst, sondern auch umgekehrt: Wenn man die Größe einer Komponente durch Ziehen an den Ziehquadraten verändert, werden die Werte der entsprechenden Eigenschaften (*Location* und *Size* im Abschnitt *Layout*) im Eigenschaftfenster automatisch aktualisiert.

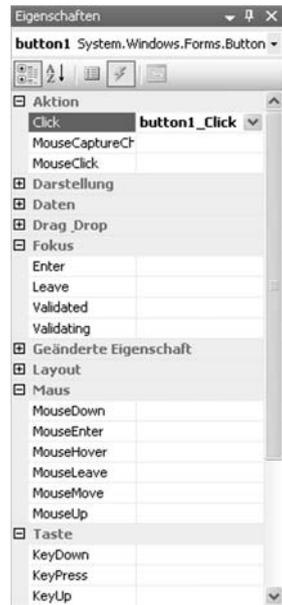
1.2 Erste Schritte in C++

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 z.B. *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 das Eigenschaftenfenster an, wenn man das Icon für die *Events* (**Ereignisse**) anklickt. 

Die Abbildung rechts zeigt einige Ereignisse für einen Button. Dabei steht *Click* 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 von Visual Studio durch einen Doppelklick auf die Zeile 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 Visual Studio die folgende Funktion: