Titel der Windows Forms Version:

C++ mit Visual Studio 2019 und Windows Forms Anwendungen

C++17 für Studierende und erfahrene Programmierer – Windows Programme mit C++ entwickeln

Vorwort

Der Ausgangspunkt für dieses Buch war der Wunsch nach einem C++-Lehrbuch, in dem von Anfang an Programme für eine grafische Benutzeroberfläche (Windows) entwickelt werden, und nicht wie meist üblich Konsolen-Anwendungen. Programme, in denen Ein- und Ausgaben über eine Konsole erfolgen, sind für viele Anfänger weltfremd wie steinzeitliche DOS-Programme und schrecken davon ab, sich überhaupt mit C++ beschäftigen zu wollen.

Windows Forms Anwendungen sind ein idealer Rahmen für C++-Programme mit einer attraktiven Benutzeroberfläche: Der Zugriff auf Windows Steuerelemente (Buttons, Text-Boxen usw.) ist einfach. Der Unterschied zu einem Standard C++ Programm ist meist nur, dass Ein- und Ausgaben über ein Windows-Steuerelement (meist eine TextBox) erfolgen

```
textBox1->AppendText("Hello World");
```

während in Standard-C++ die Konsole mit *cout* verwendet wird:

```
cout << "Hello world" << endl;
```

Aber nicht nur Studierende können von C++ mit einer grafischen Benutzeroberfläche profitieren. Mit Windows Forms Projekten können bestehende C oder C++-Programme ohne großen Aufwand mit einer grafischen Benutzeroberfläche ausgestattet werden. Und wer C oder C++ kann und keine neue Sprache für eine GUI lernen will, kann seine bestehenden Programme mit einfachen Mitteln schöner und leichter bedienbar machen.

C++ hat sich in den letzten Jahren rasant entwickelt: Die Neuerungen von C++11, C++14, C++17 und C++20 haben viele Verbesserungen und neue Möglichkeiten gebracht. Vieles, was 2010 noch gut und empfehlenswert war, kann heute besser und sicherer gemacht werden.

Das merkt man als Buchautor und Trainer, der diese ganze Entwicklung begleitet hat, besonders deutlich: Vieles, was man früher einmal geschrieben hat, sollte man heute anders machen. Zwar würde es immer noch kompiliert werden. Aber es ist kein modernes C++ mehr, das dem aktuellen Stand der Technik entspricht und alle Vorteile nutzt.

Dieses Buch stellt C++ auf dem Stand von Visual Studio 2019 im Mai 2020 vor. Das ist der Umfang von C++17.

Dieses Buch entstand einerseits aus meinen Vorlesungen an der Dualen Hochschule Lörrach und andererseits aus zahlreichen Seminaren für Firmen, die sich an Software-Entwickler richten, die C++ professionell einsetzen. Dementsprechend richtet es sich einerseits an **Studierende ohne Vorkenntnisse**. Die Lernziele für diese Gruppe sind:

- Modernes C++ auf dem Stand von 2020 zu lernen. C++ ist nach wie vor eine der am häufigsten eingesetzten Programmiersprachen.
- Programmieren zu lernen, d.h. Programme zu schreiben, die konkrete, vorgegebene Aufgaben lösen. Das ist nur mit viel Übung möglich. Programmieren lernt man nur, indem man es tut. Deshalb enthält dieses Buch auch viele Übungsaufgaben. Es ist unerlässlich, zahlreiche Aufgaben selbständig zu lösen (das müssen nicht die aus diesem Buch sein). 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 Aufgaben findet man auf *https://www.rkaiser.de*.
- Eine moderne Entwicklungsumgebung kennen- und einsetzen lernen. Visual Studio ist das in der Industrie wohl am häufigsten eingesetzte Werkzeug zur Software-Entwicklung.
- Man hört immer wieder, dass C++ zu schwierig ist, um damit das Programmieren zu lernen. Dieses Buch soll ein in vielen Jahren erprobtes Gegenargument zu dieser Meinung sein. Damit will ich die Komplexität von C++ nicht abstreiten. Aber wer C++ kann, findet sich leicht mit C#, Java usw. zurecht. Der umgekehrte Weg ist meist schwieriger.

Dieses Buch richtet sich aber ebenso an **professionelle Software-Entwickler** mit einer jahrelangen C++-Praxis. Viele C++-Neuerungen machen elementare Sprachkonzepte einfacher und sicherer. Dazu kommen anspruchsvollere Neuerungen, die bessere und effizientere Lösungen ermöglichen, die so vor einigen Jahren noch nicht möglich waren. Sowohl die einfachen als auch die anspruchsvolleren Möglichkeiten haben sich in der industriellen Praxis oft noch nicht herumgesprochen. Oft wird mit C++ noch wie vor 30 Jahren mit C programmiert. Das ist fehleranfällig, unnötig aufwendig und verschenkt Vorteile und Möglichkeiten.

- Für professionelle Entwickler soll dieses Buch einen nahezu vollständigen Überblick über C++17 geben und die Sprachelemente so vorstellen, dass sie in der praktischen Arbeit eingesetzt werden können. Die Erfahrungen aus meinen Firmenseminaren geben mir die Hoffnung, dass die allermeisten Themen abgedeckt sind, die im industriellen Einsatz notwendig sind.
- Da fast ausschließlich Standard-C++ behandelt wird, ist dieses Buch nicht auf Visual Studio beschränkt. Praktisch alle Ausführungen gelten für jeden standardkonformen C++-Compiler (gcc, Intel, Embarcadero usw.).
- Das moderne C++ bietet auch Programmierern von eingebetteten Anwendungen viele Vorteile gegenüber dem in diesem Anwendungsbereich verbreiteten C. Mit dem modernen C++ gibt es noch weniger Gründe als vorher, solche Anwendungen in C zu schreiben.
- Professionelle Entwickler sollen sich nicht an den Übungsaufgaben stören. Sie haben schon genügend Aufgaben gemacht, sonst wären sie keine Profis geworden. Und außerdem bietet das Tagesgeschäft genügend Übungsaufgaben.

Ich habe nicht nur versucht, die Sprachelemente anschaulich zu beschreiben, sondern auch Kriterien dafür anzugeben, wann und wie man sie sinnvoll einsetzen kann. Viele Empfehlungen aus style guides sind in den Text eingearbeitet, ohne dass explizit darauf hingewiesen wird.

Dieses Buch erscheint in zwei weitgehend identischen Ausgaben:

- **In der vorliegenden Ausgabe** werden Programme mit einer grafischen Benutzeroberfläche geschrieben. Alle Ein- und Ausgaben erfolgen über eine Windows-Benutzeroberfläche.

- **In der anderen Ausgabe** „C++ mit Visual Studio 2019“ werden reine Standard-C++-Programme geschrieben, d.h. ohne graphische Benutzeroberfläche. Alle Ein- und Ausgaben erfolgen über die Konsole.

Der Unterschied zwischen den beiden Ausgaben ist meist nur, dass in „C++ mit Visual Studio 2019 und Windows Forms Anwendungen“ Ein- und Ausgaben über ein Windows-Steuerelement (meist eine TextBox) erfolgen, während in „C++ mit Visual Studio 2019“ die Konsole verwendet wird.

Dazu kommen noch etwa 100 Seiten mit unterschiedlichen Inhalten, da diese in Visual Studio nur für den jeweiligen Anwendungstyp verfügbar sind. Die folgenden Themen sind nur in „C++ mit Visual Studio 2019“ enthalten: Multithreading (Kapitel 17), 15.10 parallele Algorithmen der Standardbibliothek, 2.13 C++ Core Guideline Checks, 11.2 polymorphe Memory Ressourcen (pmr,), Sprachelemente von C++20: 2.14 Module, 14.6 concepts, 4.2.4 *std::span* , 8.2.5 der dreifache Vergleichsoperator <=> usw.

Dagegen sind diese Themen nur in der Ausgabe für Windows Forms Anwendungen enthalten: „Steuerelemente für die Benutzeroberfläche“ und „C#/.NET Interoperabilität“ (Kapitel 2 und 19).

Die vorliegende Ausgabe ist eine umfassende Überarbeitung meiner Bücher über C++ mit Visual Studio 2017. Obwohl sich die beiden Jahreszahlen im Titel nur um 2 unterscheiden, ist über die Hälfte komplett neu:

- Große Teile, die heute nicht mehr aktuell sind, wurden entfernt. Vieles wurde auf den aktuellen Stand des modernen C++ gebracht. Wichtige Erweiterungen kamen dazu.
- Eigentlich wollte ich den Umfang wieder auf ca. 800 Seiten begrenzen, um potenzielle Leser nicht schon allein durch das Gewicht und die Fülle des Stoffs abzuschrecken. Das ist mir aber trotz intensivster Bemühungen nicht gelungen.

Anregungen, Korrekturhinweise und Verbesserungsvorschläge sind willkommen. Bitte senden Sie diese an die Mail-Adresse auf meiner Internetseite *www.rkaiser.de*.

Bei meinen Seminarteilnehmern und Studenten bedanke ich mich für die zahlreichen Anregungen. Dem Team vom Springer-Verlag danke ich für die Unterstützung und Geduld.

Tübingen, im Juni 2020 Richard Kaiser

Inhalt

1 Die Entwicklungsumgebung 1

1.1 Windows Forms Projekte mit C++ 1
1.1.1 Installation von Visual Studio für Windows Forms Projekte 1
1.1.2 Installation der Visual Studio Erweiterung für Windows Forms Projekte 2
1.1.3 Ein Windows Forms Projekt erstellen 4
1.1.4 Probleme beim Erstellen eines Windows Forms Projekts Θ 6
1.1.5 Ein Windows Forms Projekt manuell erstellen Θ 8
1.2 Visuelle Programmierung: Ein erstes kleines Programm 13
1.3 Das Eigenschaftenfenster 16
1.4 Erste Schritte in C++ 17
1.5 Der Quelltexteditor 19
1.5.1 Tastenkombinationen 19
1.5.2 Intellisense 21
1.5.3 Die Formatierung des Quelltexts 22
1.5.4 Definitionen einsehen 22
1.5.5 Symbole suchen 23
1.5.6 Namen umbenennen 24
1.5.7 Zeichenfolgen suchen und ersetzen 26
1.6 Kontextmenüs und Symbolleisten 27
1.7 Einige Tipps zur Arbeit mit Projekten 28
1.8 Online-Dokumentation 32
1.8.1 Die Microsoft-Dokumentation 32
1.8.2 en.cppreference.com 35
1.9 Projekte und der Projektmappen-Explorer 35
1.9.1 Projekte, Projektdateien und Projektoptionen 36
1.9.2 Projektmappen und der Projektmappen-Explorer 37
1.10 Weiterführende Möglichkeiten Θ 39
1.10.1 Navigieren 39
1.10.2 Code-Ausschnitte 41
1.10.3 Aufgabenliste 41
1.10.4 Der Objektkatalog und die Klassenansicht Θ 42
1.10.5 Die Fenster von Visual Studio anordnen Θ 42
1.10.6 Einstellungen für den Editor Θ 43
1.11 Hilfsmittel zur Gestaltung von Formularen 44
1.12 Windows Forms Anwendungen auf anderen Rechnern ausführen 45

2 Steuerelemente für die Benutzeroberfläche 47

2.1 Namen 47
2.2 Labels, Datentypen und Compiler-Fehlermeldungen 50
2.3 Funktionen, Methoden und das Steuerelement *TextBox* 55
2.3.1 Funktionen 55
2.3.2 Mehrzeilige TextBoxen 59
2.4 Klassen, ListBox und ComboBox 61
2.5 Buttons und Ereignisse 65
2.5.1 Parameter der Ereignisbehandlungsroutinen 67
2.5.2 Der Fokus und die Tabulatorreihenfolge 68
2.6 CheckBoxen, RadioButtons und einfache *if*-Anweisungen 70
2.7 Container-Steuerelemente: GroupBox, Panel, TabControl 72
2.8 Hauptmenüs und Kontextmenüs 74
2.8.1 Hauptmenüs und der Menüdesigner 75
2.8.2 Kontextmenüs 77
2.9 Standarddialoge 77
2.10 Einfache Meldungen mit *MessageBox::Show* anzeigen 82
2.11 Eine Vorlage für viele Projekte und Übungsaufgaben 83

3 Elementare Datentypen und Anweisungen 87

3.1 Syntaxregeln 87
3.2 Variablen und Bezeichner 91
3.3 Ganzzahldatentypen 94
3.3.1 Die interne Darstellung von Ganzzahlwerten 96
3.3.2 Ganzzahlliterale und ihr Datentyp 99
3.3.3 Typ-Inferenz: Implizite Typzuweisungen mit *auto* 102
3.3.4 Initialisierungslisten und Konversionen 103
3.3.5 Zuweisungen und Standardkonversionen bei Ganzzahlausdrücken Θ 105
3.3.6 Operatoren und die „üblichen arithmetischen Konversionen“ 108
3.3.7 Die Datentypen *char* und *wchar_t* 113
3.3.8 Der Datentyp *bool* 117
3.3.9 Ganzzahlwerte bei Formularanwendungen ein- und ausgeben 122
3.4 Kontrollstrukturen und Funktionen 123
3.4.1 Die *if*- und die Verbundanweisung 124
3.4.2 Die *for*- und die *while*-Schleife 128
3.4.3 Funktionen und der Datentyp *void* 131
3.4.4 Eine kleine Anleitung zum Erarbeiten der Lösungen 136
3.4.5 Wert- und Referenzparameter 140
3.4.6 Die Verwendung von Bibliotheken und Namensbereichen 140
3.4.7 Zufallszahlen 142
3.4.8 Default-Argumente 144
3.4.9 *if* und *switch* mit Variablendefinitionen 147
3.4.10 Programmierstil für Funktionen 148
3.4.11 Rekursive Funktionen 154
3.4.12 Die *switch*-Anweisung Θ 161
3.4.13 Die *do*-Anweisung Θ 164

3.4.14 Bedingte Kompilation mit *if constexpr* Θ ... 165
3.4.15 Die Sprunganweisungen *goto*, *break* und *continue* Θ ... 166
3.4.16 Assembler-Anweisungen Θ ... 168
3.5 Gleitkommadatentypen ... 169
3.5.1 Die interne Darstellung von Gleitkommawerten ... 169
3.5.2 Der Datentyp von Gleitkommaliteralen ... 172
3.5.3 Standardkonversionen ... 173
3.5.4 Mathematische Funktionen ... 179
3.5.5 Gleitkommawerte bei Formularanwendungen ein- und ausgeben ... 181
3.6 Der Debugger, Tests und Ablaufprotokolle ... 185
3.6.1 Der Debugger ... 185
3.6.2 Meldungen in Ausgabefenster/Konsolenfenster ... 189
3.6.3 Der Debugger – Weitere Möglichkeiten Θ ... 191
3.6.4 CPU Auslastung beobachten mit dem Leistungs-Profiler Θ ... 196
3.6.5 Speicherauslastung beobachten mit dem Leistungs-Profiler Θ ... 197
3.6.6 Systematisches Testen ... 200
3.6.7 Unittests: Funktionen, die Funktionen testen ... 206
3.6.8 Ablaufprotokolle ... 210
3.6.9 Symbolische Ablaufprotokolle ... 214
3.7 Konstanten ... 219
3.7.1 Laufzeitkonstanten mit *const* ... 222
3.7.2 Compilezeit-Konstanten mit *constexpr* ... 223
3.7.3 *constexpr* Funktionen ... 224
3.7.4 *static_assert* und Unittests zur Compilezeit ... 226
3.8 Kommentare ... 227
3.8.1 Kommentare zur internen Dokumentation ... 228
3.8.2 Intellisense-- und Doxygen Kommentare ... 230
3.9 Exception-Handling Grundlagen: *try*, *catch* und *throw* ... 232
3.10 Namensbereiche – Grundlagen ... 237
3.10.1 .Net-Elemente in Header-Dateien ansprechen Θ ... 238
3.11 Präprozessoranweisungen ... 239
3.11.1 Die *#include*-Anweisung ... 240
3.11.2 Makros Θ ... 241
3.11.3 Bedingte Kompilation ... 243
3.11.4 Pragmas Θ ... 248
3.12 Attribute ... 250

4 Die Stringklassen: *string*, *wstring* usw. ... 253

4.1 Die Definition von Variablen eines Klassentyps ... 254
4.2 Strings mit .NET Steuerelementen anzeigen und einlesen ... 256
4.3 Einige Elementfunktionen der Klasse *string* ... 257
4.4 Raw-String-Literale (Rohzeichenfolgen) ... 267
4.5 Stringkonversionen ... 269
4.5.1 C++11-Konversionsfunktionen: *to_string*, *stoi* usw. ... 270
4.5.2 C++17-Konversionsfunktionen: *to_chars* und *from_chars* ... 272
4.5.3 Konversionen mit Stringstreams Θ ... 276
4.6 *string_view* –Strings zum Anschauen ... 278

4.7 Reguläre Ausdrücke Θ 282

5 Arrays und Container 293

5.1 Synonyme für Datentypen 294
5.1.1 Einfache *typedef*-Deklarationen 294
5.1.2 Synonyme für Datentypen mit *using* 294
5.2 Eindimensionale Arrays 295
5.2.1 Arrays im Stil von C 295
5.2.2 Arrays des Typs *std::array* 298
5.2.3 Dynamische Arrays des Typs *std::vector* 299
5.2.4 Die Initialisierung von Arrays bei ihrer Definition 302
5.2.5 Vorteile von *std::array* und *std::vector* gegenüber C-Arrays 304
5.2.6 Ein einfaches Sortierverfahren (Auswahlsort) 307
5.3 Arrays als Container 308
5.4 Mehrdimensionale Arrays Θ 312
5.5 Dynamische Programmierung Θ 313

6 Einfache selbstdefinierte Datentypen 315

6.1 Mit *struct* definierte Klassen 315
6.2 Aufzählungstypen 320
6.2.1 Schwach typisierte Aufzählungstypen (C/C++03) 321
6.2.2 *enum* Konstanten und Konversionen Θ 322
6.2.3 Stark typisierte Aufzählungstypen (C++11) 323

7 Zeiger, dynamisch erzeugte Variablen und smart pointer 327

7.1 Die Definition von Zeigervariablen 328
7.2 Der Adressoperator, Zuweisungen und generische Zeiger 331
7.3 Ablaufprotokolle für Zeigervariable 336
7.4 Explizite Konversionen Θ 338
7.5 Dynamisch erzeugte Variablen 339
7.5.1 *new* und *delete* 339
7.5.2 Der Unterschied zu „gewöhnlichen“ Variablen 343
7.5.3 Der Smart Pointer Typ *unique_ptr* 345
7.5.4 Dynamische erzeugte eindimensionale Arrays 347
7.6 Stringliterale, nullterminierte Strings und *char**-Zeiger 349
7.7 Memory Leaks finden 354
7.8 Verkettete Listen 355
7.9 Binärbäume Θ 365

8 Überladene Funktionen und Operatoren 371

8.1 Inline-Funktionen Θ 371
8.2 Überladene Funktionen 374

8.2.1 Funktionen, die nicht überladen werden können ... 376
8.2.2 Regeln für die Auswahl einer passenden Funktion ... 377
8.3 Überladene Operatoren mit globalen Operatorfunktionen ... 383
8.3.1 Globale Operatorfunktionen ... 385
8.3.2 Die Ein- und Ausgabe von selbst definierten Datentypen ... 388
8.3.3 *new* und *delete* überladen ... 390
8.4 Referenztypen, Wert- und Referenzparameter ... 393
8.4.1 Wertparameter ... 393
8.4.2 Referenztypen ... 394
8.4.3 Referenzparameter ... 395
8.4.4 Referenzen als Rückgabetypen ... 398
8.4.5 Konstante Referenzparameter ... 400
8.5 Reihenfolge der Auswertung von Ausdrücken seit C++17 ... 402

9 Objektorientierte Programmierung ... 405

9.1 Klassen ... 406
9.1.1 Datenelemente und Elementfunktionen ... 406
9.1.2 Der Gültigkeitsbereich von Klassenelementen ... 411
9.1.3 Datenkapselung: Die Zugriffsrechte *private* und *public* ... 414
9.1.4 Der Aufruf von Elementfunktionen und der *this*-Zeiger ... 420
9.1.5 Konstruktoren und Destruktoren ... 421
9.1.6 *shared_ptr*: Smart Pointer für Klassenelemente ... 434
9.1.7 OO Analyse und Design: Der Entwurf von Klassen ... 435
9.1.8 Klassendiagramme ... 438
9.1.9 Initialisierungslisten für Variablen, Argumente und Rückgabewerte ... 440
9.1.10 Initialisierungslisten als Parameter ... 442
9.1.11 Implizite Typzuweisungen mit *auto* ... 444
9.2 Klassen als Datentypen ... 449
9.2.1 Der Standardkonstruktor ... 450
9.2.2 Elementinitialisierer ... 452
9.2.3 *friend*-Funktionen und –Klassen ... 458
9.2.4 Überladene Operatoren mit Elementfunktionen ... 461
9.2.5 Der Kopierkonstruktor ... 465
9.2.6 Der Zuweisungsoperator = für Klassen ... 470
9.2.7 Die Angaben =*delete* und =*default* ... 476
9.2.8 Konvertierende und explizite Konstruktoren Θ ... 478
9.2.9 Konversionsfunktionen mit und ohne *explicit* Θ ... 481
9.2.10 Statische Klassenelemente ... 482
9.2.11 *inline* Variablen, insbesondere *static inline* Datenelemente ... 488
9.2.12 Konstante Objekte und Elementfunktionen ... 490
9.2.13 *std::function*: Ein Datentyp für Funktionen und aufrufbare Objekte ... 493
9.2.14 Delegierende Konstruktoren Θ ... 497
9.2.15 Klassen und Header-Dateien ... 499
9.3 Vererbung und Komposition ... 501
9.3.1 Die Elemente von abgeleiteten Klassen ... 502
9.3.2 Zugriffsrechte auf die Elemente von Basisklassen ... 504
9.3.3 Verdeckte Elemente ... 506

9.3.4 Konstruktoren, Destruktoren und implizit erzeugte Funktionen 508
9.3.5 Vererbung bei Formularen in Windows Forms Anwendungen 516
9.3.6 OO Design: *public* Vererbung und „ist ein“-Beziehungen 516
9.3.7 OO Design: Komposition und „hat ein“-Beziehungen 521
9.3.8 Konversionen zwischen *public* abgeleiteten Klassen 522
9.3.9 Mehrfachvererbung und virtuelle Basisklassen 525
9.4 Virtuelle Funktionen, späte Bindung und Polymorphie 530
9.4.1 Der statische und der dynamische Datentyp 530
9.4.2 Virtuelle Funktionen in C++03 532
9.4.3 Virtuelle Funktionen mit *override* in C++11 533
9.4.4 Die Implementierung von virtuellen Funktionen: *vptr* und *vtbl* 542
9.4.5 Virtuelle Konstruktoren und Destruktoren 548
9.4.6 Virtuelle Funktionen in Konstruktoren und Destruktoren 550
9.4.7 OO-Design: Einsatzbereich und Test von virtuellen Funktionen 551
9.4.8 OO-Design und Erweiterbarkeit 553
9.4.9 Rein virtuelle Funktionen und abstrakte Basisklassen 556
9.4.10 OO-Design: Virtuelle Funktionen und abstrakte Basisklassen 560
9.4.11 Interfaces und Mehrfachvererbung 562
9.4.12 Objektorientierte Programmierung: Zusammenfassung 564
9.5 R-Wert Referenzen und Move-Semantik 565
9.5.1 R-Werte und R-Wert Referenzen 566
9.5.2 move-Semantik 569
9.5.3 R-Werte mit *std::move* erzwingen 570
9.5.4 Move-Semantik in der C++11 Standardbibliothek 575
9.5.5 Move-Semantik für eigene Klassen 577
9.5.6 Perfect forwarding Θ 580

10 Namensbereiche 581

10.1 Die Definition von Namensbereichen 582
10.2 Die Verwendung von Namen aus Namensbereichen 585
10.3 Header-Dateien und Namensbereiche 588
10.4 Aliasnamen für Namensbereiche Θ 592
10.5 inline namespaces Θ 593

11 Exception-Handling 595

11.1 Die *try*-Anweisung 596
11.2 Exception-Handler und Exceptions der Standardbibliothek 601
11.3 Einige vordefinierte C++/CLI und .NET Exceptions 604
11.4 *throw*-Ausdrücke und selbst definierte Exceptions 606
11.5 Exceptions weitergeben 612
11.6 Fehler und Exceptions 614
11.7 Die Freigabe von Ressourcen bei Exceptions: RAII 617
11.8 Exceptions in Konstruktoren und Destruktoren 619
11.9 *noexcept* 626
11.10 Exception-Sicherheit 627

12 Containerklassen der C++-Standardbibliothek 631

12.1 Sequenzielle Container der Standardbibliothek 631
12.1.1 Die Container-Klasse *vector* 631
12.1.2 Iteratoren 636
12.1.3 Geprüfte Iteratoren (Checked Iterators) 640
12.1.4 Die bereichsbasierte *for*-Schleife 641
12.1.5 Iteratoren und die Algorithmen der Standardbibliothek 644
12.1.6 Die Speicherverwaltung bei Vektoren Θ 647
12.1.7 Mehrdimensionale Vektoren Θ 649
12.1.8 Gemeinsamkeiten und Unterschiede der sequenziellen Container 650
12.1.9 Die Container-Adapter *stack*, *queue* und *priority_queue* Θ 653
12.1.10 Container mit Zeigern 654
12.2 Assoziative Container 655
12.2.1 Die Container *set* und *multiset* 655
12.2.2 Die Container *map* und *multimap* 656
12.2.3 Iteratoren der assoziativen Container 658
12.2.4 Ungeordnete Assoziative Container (Hash-Container) 661
12.3 Zusammenfassungen von Datentypen 665
12.3.1 Wertepaare mit *std::pair* 666
12.3.2 Tupel mit *std::tuple* 667
12.3.3 Strukturierte Bindungen Θ 668
12.3.4 *std::optional* – Eine Klasse für einen oder keinen Wert 672
12.3.5 *std::variant* – Eine Klasse für Werte bestimmter Typen 675
12.3.6 *std::any*: Ein Datentyp für Werte beliebiger Typen Θ 678

13 Dateibearbeitung mit den Stream-Klassen 681

13.1 Stream-Variablen, ihre Verbindung mit Dateien und ihr Zustand 681
13.2 Fehler und der Zustand von Stream-Variablen 686
13.3 Lesen und Schreiben von Binärdaten mit *read* und *write* 687
13.4 Lesen und Schreiben mit den Operatoren << und >> 693
13.5 Filesystem 701
13.6 Ein selbstgestrickter Windows-Explorer 705
13.6.1 Die Anzeige von Listen mit ListView 705
13.6.2 ListView nach Spalten sortieren 708
13.6.3 Die Anzeige von Baumstrukturen mit TreeView 710
13.6.4 SplitContainer: Ein selbstgestrickter Windows Explorer 713

14 Funktoren, Funktionsobjekte und Lambda-Ausdrücke 715

14.1 Der Aufrufoperator () 715
14.2 Prädikate und Vergleichsfunktionen 719
14.3 Binder Θ 724
14.4 Lambda-Ausdrücke 727
14.5 Lambda-Ausdrücke – Weitere Konzepte Θ 736
14.5.1 Lambda-Ausdrücke werden zu Funktionsobjekten 736
14.5.2 Nachstehende Rückgabetypen 737

14.5.3 Generische Lambda-Ausdrücke ... 738
14.5.4 Lambda-Ausdrücke höherer Ordnung Θ ... 738

15 Templates ... 741

15.1 Generische Funktionen: Funktions-Templates ... 742
15.1.1 Die Deklaration von Funktions-Templates mit Typ-Parametern ... 743
15.1.2 Spezialisierungen von Funktions-Templates ... 744
15.1.3 Fehlersuche bei Template-Instanziierungen ... 751
15.1.4 Funktions-Templates mit Nicht-Typ-Parametern ... 753
15.1.5 Explizit instanziierte Funktions-Templates Θ ... 755
15.1.6 Explizit spezialisierte und überladene Templates ... 756
15.1.7 Rekursive Funktions-Templates Θ ... 760
15.2 Generische Klassen: Klassen-Templates ... 763
15.2.1 Die Deklaration von Klassen-Templates mit Typ-Parametern ... 763
15.2.2 Spezialisierungen von Klassen-Templates ... 764
15.2.3 Klassen-Templates mit Nicht-Typ-Parametern ... 771
15.2.4 Explizit instanziierte Klassen-Templates Θ ... 773
15.2.5 Partielle und vollständige Spezialisierungen Θ ... 774
15.2.6 Vererbung mit Klassen-Templates Θ ... 780
15.2.7 Die Ableitung von Typ-Argumenten bei Klassen-Templates ... 781
15.2.8 Alias Templates Θ ... 785
15.3 Variablen-Templates Θ ... 786
15.4 Typ-Argument abhängige Templates mit Type Traits ... 788
15.4.1 Eine Konstruktion von type traits ... 788
15.4.2 Die type traits Kategorien ... 789
15.4.3 Type traits und static_assert ... 791
15.4.4 Templates mit *if constexpr* und type traits optimieren ... 793
15.4.5 Typ-Inferenz mit *decltype* ... 794
15.5 Variadische Templates ... 798
15.5.1 Variadische Funktions-Templates ... 799
15.5.2 Fold Ausdrücke ... 802
15.5.3 Variadische Klassen-Templates am Beispiel von *std::tuple* ... 804

16 STL-Algorithmen und Lambda-Ausdrücke ... 807

16.1 Iteratoren ... 807
16.1.1 Die verschiedenen Arten von Iteratoren ... 808
16.1.2 Umkehriteratoren ... 810
16.1.3 Einfügefunktionen und Einfügeiteratoren ... 811
16.1.4 Stream-Iteratoren ... 813
16.1.5 Container-Konstruktoren mit Iteratoren ... 815
16.1.6 Globale Iterator-Funktionen Θ ... 816
16.2 Nichtmodifizierende Algorithmen ... 818
16.2.1 Lineares Suchen ... 818
16.2.2 Zählen ... 820
16.2.3 Der Vergleich von Bereichen ... 821

16.2.4 Suche nach Teilfolgen ... 822
16.2.5 Minimum und Maximum ... 823
16.2.6 Mit *all_of*, *any_of*, *none_of* alle Elemente in einem Bereich prüfen ... 824
16.3 Kopieren und Verschieben von Bereichen ... 825
16.4 Elemente transformieren und ersetzen ... 826
16.5 Elementen in einem Bereich Werte zuweisen Θ ... 828
16.6 Elemente entfernen– das *erase-remove* Idiom ... 830
16.7 Die Reihenfolge von Elementen vertauschen ... 832
16.7.1 Elemente vertauschen ... 832
16.7.2 Permutationen Θ ... 833
16.7.3 Die Reihenfolge umkehren und Elemente rotieren Θ ... 834
16.7.4 Elemente durcheinander mischen Θ ... 835
16.8 Algorithmen zum Sortieren und für sortierte Bereiche ... 835
16.8.1 Partitionen Θ ... 835
16.8.2 Bereiche sortieren ... 836
16.8.3 Binäres Suchen in sortierten Bereichen ... 840
16.8.4 Mischen von sortierten Bereichen ... 841
16.9 Numerische Algorithmen und Datentypen ... 842
16.9.1 Numerische Algorithmen ... 843
16.9.2 Valarrays Θ ... 845
16.9.3 Zufallszahlen mit <random> Θ ... 847
16.9.4 Komplexe Zahlen Θ ... 850
16.9.5 Numerische Bibliotheken neben dem C++-Standard Θ ... 853

17 Zeiten und Kalenderdaten mit *chrono* ... 853

17.1 Brüche als Datentypen: Das Klassen-Template *ratio* ... 854
17.2 Ein Datentyp für Zeiteinheiten: *duration* ... 855
17.3 Datentypen für Zeitpunkte: *time_point* ... 859
17.4 Uhren: *system_clock* und *steady_clock* ... 861

18 Smart Pointer: *shared_ptr*, *unique_ptr* und *weak_ptr* ... 865

18.1 Gemeinsamkeiten von *unique_ptr* und *shared_ptr* ... 866
18.2 *unique_ptr* ... 873
18.3 shared_ptr ... 875
18.4 Deleter und Smart Pointer für Arrays ... 881
18.5 weak_ptr Θ ... 883

19 C++/CLI, .NET-Bibliotheken und C++ Interoperabilität ... 887

19.1 Native C++-Code in C#/.NET DLLs verwenden ... 887
19.2 Native C++-Code in C#/.NET Core DLLs verwenden ... 891
19.3 C++/CLI Grundlagen ... 892
19.3.1 Verweisklassen ... 892
19.3.2 *System::String* und *std::string* konvertieren ... 894
19.3.3 Garbage Collection und der GC-Heap ... 895

19.3.4 Destruktoren und Finalisierer 897
19.4 Assemblies 899
19.4.1 Anwendungen und die *main*-Funktion 901
19.4.2 DLLs 902
19.4.3 Disassembler und Obfuscation 903
19.5 .NET Klassen mit C++/CLI in Windows Forms verwenden 906
19.5.1 Steuerelemente manuell erzeugen 907
19.5.2 Ein Steuerelement manuell einem Formular hinzufügen 908
19.5.3 EMails versenden 908
19.5.4 Grafiken zeichnen 909
19.6 Microsoft Office Anwendungen steuern 914
19.6.1 Microsoft Office Word 915
19.6.2 Excel 917

Literaturverzeichnis 919

Index 921

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.