

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 Ø.....6
 - 1.1.5 Ein Windows Forms Projekt manuell erstellen Ø8
 - 1.1.6 Projektvorlagen exportieren13
 - 1.2 Visuelle Programmierung: Ein erstes kleines Programm 14
 - 1.3 Das Eigenschaftsfenster 17
 - 1.4 Erste Schritte in C++..... 19
 - 1.5 Der Quelltexteditor 21
 - 1.5.1 Tastenkombinationen 21
 - 1.5.2 Intellisense..... 22
 - 1.5.3 Die Formatierung des Quelltexts 23
 - 1.5.4 Definitionen einsehen 24
 - 1.5.5 Symbole suchen..... 25
 - 1.5.6 Namen umbenennen..... 25
 - 1.5.7 Zeichenfolgen suchen und ersetzen..... 27
 - 1.6 Kontextmenüs und Symbolleisten 29
 - 1.7 Einige Tipps zur Arbeit mit Projekten 30
 - 1.8 Online-Dokumentation..... 34
 - 1.8.1 Die Microsoft-Dokumentation..... 34
 - 1.8.2 en.cppreference.com 37
 - 1.9 Projekte und der Projektmappen-Explorer 37
 - 1.9.1 Projekte, Projektdateien und Projektoptionen..... 38
 - 1.9.2 Projektmappen und der Projektmappen-Explorer 39
 - 1.10 Weiterführende Möglichkeiten Ø 41
 - 1.10.1 Navigieren 41
 - 1.10.2 Code-Ausschnitte..... 43
 - 1.10.3 Aufgabenliste..... 43
 - 1.10.4 Der Objektkatalog und die Klassenansicht Ø 44
 - 1.10.5 Die Fenster von Visual Studio anordnen Ø 44
 - 1.10.6 Einstellungen für den Editor Ø 45
 - 1.11 Hilfsmittel zur Gestaltung von Formularen..... 46

1.12	Windows Forms Anwendungen auf anderen Rechnern ausführen.....	47
2	Steuerelemente für die Benutzeroberfläche	49
2.1	Namen.....	49
2.2	Labels, Datentypen und Compiler-Fehlermeldungen.....	52
2.3	Funktionen, Methoden und das Steuerelement <i>TextBox</i>	57
2.3.1	Funktionen.....	58
2.3.2	Mehrzeilige <i>TextBox</i> en.....	62
2.4	Klassen, <i>ListBox</i> und <i>ComboBox</i>	64
2.5	Buttons und Ereignisse.....	68
2.5.1	Parameter der Ereignisbehandlungsroutinen	69
2.5.2	Der Fokus und die Tabulatorreihenfolge.....	71
2.5.3	Eventhandler verwalten.....	71
2.6	<i>CheckBox</i> en, <i>RadioButtons</i> und einfache <i>if</i> -Anweisungen.....	73
2.7	Container-Steuerelemente: <i>GroupBox</i> , <i>Panel</i> , <i>TabControl</i>	75
2.8	Hauptmenüs und Kontextmenüs.....	77
2.8.1	Hauptmenüs und der Menüdesigner.....	78
2.8.2	Kontextmenüs.....	80
2.9	Standarddialoge	80
2.10	Einfache Meldungen mit <i>MessageBox::Show</i> anzeigen.....	85
2.11	Eine Vorlage für viele Projekte und Übungsaufgaben	86
2.11.1	Die Erweiterung „C++ Windows Forms Project with GUI for VS 2022“	86
2.11.2	<i>CppCLR_Utils</i> und <i>SimpleUnitTests</i>	90
3	Elementare Datentypen und Anweisungen	93
3.1	Syntaxregeln.....	93
3.2	Variablen und Bezeichner	97
3.3	Ganzzahldatentypen.....	100
3.3.1	Die interne Darstellung von Ganzzahlwerten	102
3.3.2	Ganzzahl Literale und ihr Datentyp	105
3.3.3	Typ-Inferenz: Implizite Typzuweisungen mit <i>auto</i>	108
3.3.4	Initialisierungslisten und Konversionen	109
3.3.5	Zuweisungen und Standardkonversionen bei Ganzzahlausdrücken Θ	111
3.3.6	Operatoren und die „üblichen arithmetischen Konversionen“.....	114
3.3.7	Die Datentypen <i>char</i> und <i>wchar_t</i>	118
3.3.8	Der Datentyp <i>bool</i>	123
3.3.9	Ganzzahlwerte bei Formularanwendungen ein- und ausgeben.....	128
3.4	Kontrollstrukturen und Funktionen.....	129
3.4.1	Die <i>if</i> - und die Verbundanweisung	129
3.4.2	Die <i>for</i> - und die <i>while</i> -Schleife.....	133
3.4.3	Funktionen und der Datentyp <i>void</i>	137
3.4.4	Eine kleine Anleitung zum Erarbeiten der Lösungen.....	141
3.4.5	Wert- und Referenzparameter.....	145
3.4.6	Die Verwendung von Bibliotheken und Namensbereichen.....	146
3.4.7	Zufallszahlen	148

3.4.8	Default-Argumente	150
3.4.9	<i>if</i> und <i>switch</i> mit Variablendefinitionen	152
3.4.10	Programmierstil für Funktionen.....	154
3.4.11	Rekursive Funktionen	160
3.4.12	Die <i>switch</i> -Anweisung Θ	166
3.4.13	Die <i>do</i> -Anweisung Θ	169
3.4.14	Bedingte Kompilation mit <i>if constexpr</i> Θ	170
3.4.15	Die Sprunganweisungen <i>goto</i> , <i>break</i> und <i>continue</i> Θ	172
3.4.16	Assembler-Anweisungen Θ	174
3.5	Gleitkommatypen	174
3.5.1	Die interne Darstellung von Gleitkommawerten	175
3.5.2	Der Datentyp von Gleitkommaliteralen	178
3.5.3	Standardkonversionen.....	179
3.5.4	Mathematische Funktionen	185
3.5.5	Gleitkommawerte bei Formularanwendungen ein- und ausgeben	186
3.6	Der Debugger, Tests und Ablaufprotokolle.....	190
3.6.1	Der Debugger	191
3.6.2	Meldungen in Ausgabefenster/Konsolenfenster	194
3.6.3	Der Debugger – Weitere Möglichkeiten Θ	197
3.6.4	CPU Auslastung beobachten mit dem Leistungs-Profiler Θ	201
3.6.5	Speicherauslastung beobachten mit dem Leistungs-Profiler Θ	202
3.6.6	Systematisches Testen.....	205
3.6.7	Unittests: Funktionen, die Funktionen testen.....	211
3.6.8	Ablaufprotokolle.....	215
3.6.9	Symbolische Ablaufprotokolle	219
3.7	Konstanten.....	224
3.7.1	Laufzeitkonstanten mit <i>const</i>	227
3.7.2	Compilezeit-Konstanten mit <i>constexpr</i>	228
3.7.3	<i>constexpr</i> Funktionen.....	229
3.7.4	<i>static_assert</i> und Unittests zur Compilezeit	231
3.8	Kommentare	232
3.8.1	Kommentare zur internen Dokumentation	233
3.8.2	Intellisense- und Doxygen Kommentare.....	235
3.9	Exception-Handling Grundlagen: <i>try</i> , <i>catch</i> und <i>throw</i>	237
3.10	Namensbereiche – Grundlagen.....	242
3.10.1	.Net-Elemente in Header-Dateien ansprechen Θ	243
3.11	Präprozessoranweisungen	244
3.11.1	Die <i>#include</i> -Anweisung.....	245
3.11.2	Makros Θ	246
3.11.3	Bedingte Kompilation	248
3.11.4	Pragmas Θ	253
3.12	Attribute	255
4	Die Stringklassen: <i>string</i>, <i>wstring</i> usw.	259
4.1	Die Definition von Variablen eines Klassentyps	260
4.2	Strings mit .NET Steuerelementen anzeigen und einlesen	262
4.3	Einige Elementfunktionen der Klasse <i>string</i>	263

4.4	Raw-String-Literale (Rohzeichenfolgen)	273
4.5	Stringkonversionen	275
4.5.1	C++11-Konversionsfunktionen: <i>to_string</i> , <i>stoi</i> usw.	276
4.5.2	C++17-Konversionsfunktionen: <i>to_chars</i> und <i>from_chars</i>	278
4.5.3	Konversionen mit Stringstreams Θ	282
4.6	<i>string_view</i> –Strings zum Anschauen	284
4.7	Reguläre Ausdrücke Θ	288
5	Arrays und Container.....	299
5.1	Synonyme für Datentypen.....	300
5.1.1	Einfache <i>typedef</i> -Deklarationen.....	300
5.1.2	Synonyme für Datentypen mit <i>using</i>	300
5.2	Eindimensionale Arrays.....	301
5.2.1	Arrays im Stil von C	301
5.2.2	Arrays des Typs <i>std::array</i>	304
5.2.3	Dynamische Arrays des Typs <i>std::vector</i>	305
5.2.4	Die Initialisierung von Arrays bei ihrer Definition	308
5.2.5	Vorteile von <i>std::array</i> und <i>std::vector</i> gegenüber C-Arrays.....	310
5.2.6	Ein einfaches Sortierverfahren (Auswahlort).....	313
5.3	Arrays als Container	314
5.4	Mehrdimensionale Arrays Θ	318
5.5	Dynamische Programmierung Θ	319
6	Einfache selbstdefinierte Datentypen.....	321
6.1	Mit <i>struct</i> definierte Klassen	321
6.2	Aufzählungstypen	326
6.2.1	Schwach typisierte Aufzählungstypen (C/C++03).....	327
6.2.2	<i>enum</i> Konstanten und Konversionen Θ	328
6.2.3	Stark typisierte Aufzählungstypen (C++11).....	329
7	Zeiger, dynamisch erzeugte Variablen und smart pointer	333
7.1	Die Definition von Zeigervariablen	334
7.2	Der Adressoperator, Zuweisungen und generische Zeiger	337
7.3	Explizite Konversionen Θ	342
7.4	Ablaufprotokolle für Zeigervariable	343
7.5	Dynamisch erzeugte Variablen.....	345
7.5.1	<i>new</i> und <i>delete</i>	345
7.5.2	Der Unterschied zu „gewöhnlichen“ Variablen.....	348
7.5.3	Der Smart Pointer Typ <i>unique_ptr</i>	351
7.5.4	Dynamische erzeugte eindimensionale Arrays	352
7.6	Stringliterale, nullterminierte Strings und <i>char*</i> -Zeiger	354
7.7	Arrays, Zeiger und Zeigerarithmetik.....	360
7.8	Memory Leaks finden	362
7.9	Verkettete Listen.....	364

7.10 Binärbäume Θ	374
8 Überladene Funktionen und Operatoren.....	381
8.1 Inline-Funktionen Θ	381
8.2 Überladene Funktionen.....	384
8.2.1 Funktionen, die nicht überladen werden können.....	386
8.2.2 Regeln für die Auswahl einer passenden Funktion.....	387
8.3 Überladene Operatoren mit globalen Operatorfunktionen.....	393
8.3.1 Globale Operatorfunktionen.....	395
8.3.2 Die Ein- und Ausgabe von selbst definierten Datentypen.....	398
8.3.3 <i>new</i> und <i>delete</i> überladen.....	400
8.4 Referenztypen, Wert- und Referenzparameter.....	403
8.4.1 Wertparameter.....	403
8.4.2 Referenztypen.....	404
8.4.3 Referenzparameter.....	405
8.4.4 Referenzen als Rückgabetypen.....	408
8.4.5 Konstante Referenzparameter.....	410
8.5 Reihenfolge der Auswertung von Ausdrücken seit C++17.....	412
9 Objektorientierte Programmierung.....	415
9.1 Klassen.....	416
9.1.1 Datenelemente und Elementfunktionen.....	416
9.1.2 Der Gültigkeitsbereich von Klassenelementen.....	421
9.1.3 Datenkapselung: Die Zugriffsrechte <i>private</i> und <i>public</i>	424
9.1.4 Der Aufruf von Elementfunktionen und der <i>this</i> -Zeiger.....	430
9.1.5 Konstruktoren und Destruktoren.....	431
9.1.6 <i>shared_ptr</i> : Smart Pointer für Klassenelemente und RAI.....	444
9.1.7 OO Analyse und Design: Der Entwurf von Klassen.....	445
9.1.8 Klassendiagramme.....	449
9.1.9 Initialisierungslisten für Variablen, Argumente und Rückgabewerte.....	451
9.1.10 Initialisierungslisten als Parameter.....	452
9.1.11 Implizite Typzuweisungen mit <i>auto</i>	454
9.2 Klassen als Datentypen.....	460
9.2.1 Der Standardkonstruktor.....	460
9.2.2 Elementinitialisierer.....	462
9.2.3 <i>friend</i> -Funktionen und -Klassen.....	468
9.2.4 Überladene Operatoren mit Elementfunktionen.....	472
9.2.5 Der Kopierkonstruktor.....	476
9.2.6 Der Zuweisungsoperator = für Klassen.....	481
9.2.7 Die Angaben = <i>delete</i> und = <i>default</i>	486
9.2.8 Konvertierende und explizite Konstruktoren Θ	488
9.2.9 Konversionsfunktionen mit und ohne <i>explicit</i> Θ	492
9.2.10 Statische Klassenelemente.....	493
9.2.11 <i>inline</i> Variablen, insbesondere <i>static inline</i> Datenelemente.....	498
9.2.12 Konstante Objekte und Elementfunktionen.....	500

9.2.13	<i>std::function</i> : Ein Datentyp für Funktionen und aufrufbare Objekte.....	503
9.2.14	Delegierende Konstruktoren Θ	508
9.2.15	Klassen und Header-Dateien	509
9.3	Vererbung und Komposition	511
9.3.1	Die Elemente von abgeleiteten Klassen	512
9.3.2	Zugriffsrechte auf die Elemente von Basisklassen.....	514
9.3.3	Verdeckte Elemente.....	516
9.3.4	Konstruktoren, Destruktoren und implizit erzeugte Funktionen.....	518
9.3.5	Vererbung bei Formularen in Windows Forms Anwendungen	526
9.3.6	OO Design: <i>public</i> Vererbung und „ist ein“-Beziehungen.....	526
9.3.7	OO Design: Komposition und „hat ein“-Beziehungen.....	531
9.3.8	Konversionen zwischen <i>public</i> abgeleiteten Klassen.....	532
9.3.9	Mehrfachvererbung und virtuelle Basisklassen	535
9.4	Virtuelle Funktionen, späte Bindung und Polymorphie	539
9.4.1	Der statische und der dynamische Datentyp.....	539
9.4.2	Virtuelle Funktionen in C++03.....	541
9.4.3	Virtuelle Funktionen mit <i>override</i> in C++11	542
9.4.4	Die Implementierung von virtuellen Funktionen: <i>vptr</i> und <i>vtbl</i>	552
9.4.5	Virtuelle Konstruktoren und Destruktoren	557
9.4.6	Virtuelle Funktionen in Konstruktoren und Destruktoren	559
9.4.7	OO-Design: Einsatzbereich und Test von virtuellen Funktionen.....	560
9.4.8	OO-Design und Erweiterbarkeit	563
9.4.9	Rein virtuelle Funktionen und abstrakte Basisklassen	566
9.4.10	OO-Design: Virtuelle Funktionen und abstrakte Basisklassen	569
9.4.11	Interfaces und Mehrfachvererbung	572
9.4.12	Objektorientierte Programmierung: Zusammenfassung.....	573
9.5	R-Wert Referenzen und Move-Semantik	575
9.5.1	R-Werte und R-Wert Referenzen.....	576
9.5.2	move-Semantik.....	578
9.5.3	R-Werte mit <i>std::move</i> erzwingen	580
9.5.4	Move-Semantik in der C++11 Standardbibliothek	585
9.5.5	Move-Semantik für eigene Klassen	587
9.5.6	Perfect forwarding Θ	590
10	Namensbereiche	591
10.1	Die Definition von Namensbereichen	592
10.2	Die Verwendung von Namen aus Namensbereichen	595
10.3	Header-Dateien und Namensbereiche	598
10.4	Aliasnamen für Namensbereiche Θ	602
10.5	<i>inline namespaces</i> Θ	603
11	Exception-Handling	605
11.1	Die <i>try</i> -Anweisung.....	606
11.2	Exception-Handler und Exceptions der Standardbibliothek.....	611
11.3	Einige vordefinierte C++/CLI und .NET Exceptions.....	614

11.4 *throw*-Ausdrücke und selbst definierte Exceptions..... 616

11.5 Exceptions weitergeben..... 622

11.6 Fehler und Exceptions..... 624

11.7 Die Freigabe von Ressourcen bei Exceptions: RAII..... 627

11.8 Exceptions in Konstruktoren und Destruktoren..... 629

11.9 *noexcept* 636

11.10 Exception-Sicherheit..... 637

12 Containerklassen der C++-Standardbibliothek..... 641

12.1 Sequenzielle Container der Standardbibliothek 641

12.1.1 Die Container-Klasse *vector*..... 641

12.1.2 Iteratoren..... 646

12.1.3 Geprüfte Iteratoren (Checked Iterators) 650

12.1.4 Die bereichsbasierte *for*-Schleife..... 651

12.1.5 Iteratoren und die Algorithmen der Standardbibliothek 654

12.1.6 Die Speicherverwaltung bei Vektoren Θ 657

12.1.7 Mehrdimensionale Vektoren Θ 659

12.1.8 Gemeinsamkeiten und Unterschiede der sequenziellen Container..... 660

12.1.9 Die Container-Adapter *stack*, *queue* und *priority_queue* Θ 663

12.1.10 Container mit Zeigern 664

12.2 Assoziative Container 665

12.2.1 Die Container *set* und *multiset*..... 665

12.2.2 Die Container *map* und *multimap*..... 666

12.2.3 Iteratoren der assoziativen Container..... 668

12.2.4 Ungeordnete Assoziative Container (Hash-Container) 671

12.3 Zusammenfassungen von Datentypen..... 676

12.3.1 Wertepaare mit *std::pair* 676

12.3.2 Tupel mit *std::tuple*..... 677

12.3.3 Strukturierte Bindungen Θ 679

12.3.4 *std::optional* – Eine Klasse für einen oder keinen Wert..... 683

12.3.5 *std::variant* – Eine Klasse für Werte bestimmter Typen..... 686

12.3.6 *std::any*: Ein Datentyp für Werte beliebiger Typen Θ 689

13 Dateibearbeitung mit den Stream-Klassen 693

13.1 Stream-Variablen, ihre Verbindung mit Dateien und ihr Zustand 693

13.2 Fehler und der Zustand von Stream-Variablen..... 698

13.3 Lesen und Schreiben von Binärdaten mit *read* und *write*..... 699

13.4 Lesen und Schreiben mit den Operatoren *<<* und *>>*..... 705

13.5 Filesystem 713

13.6 Ein selbstgestrickter Windows-Explorer..... 717

13.6.1 Die Anzeige von Listen mit *ListView*..... 718

13.6.2 *ListView* nach Spalten sortieren..... 720

13.6.3 Die Anzeige von Baumstrukturen mit *TreeView*..... 723

13.6.4 *SplitContainer*: Ein selbstgestrickter Windows Explorer 726

14 Funktoren, Funktionsobjekte und Lambda-Ausdrücke.....	727
14.1 Der Aufrufoperator ().....	727
14.2 Prädikate und Vergleichsfunktionen.....	731
14.3 Binder Θ	736
14.4 Lambda-Ausdrücke.....	739
14.5 Lambda-Ausdrücke – Weitere Konzepte Θ	748
14.5.1 Lambda-Ausdrücke werden zu Funktionsobjekten.....	748
14.5.2 Nachstehende Rückgabetypen.....	749
14.5.3 Generische Lambda-Ausdrücke.....	750
14.5.4 Lambda-Ausdrücke höherer Ordnung Θ	750
15 Templates.....	753
15.1 Generische Funktionen: Funktions-Templates.....	754
15.1.1 Die Deklaration von Funktions-Templates mit Typ-Parametern.....	755
15.1.2 Spezialisierungen von Funktions-Templates.....	756
15.1.3 Fehlersuche bei Template-Instanzierungen.....	763
15.1.4 Funktions-Templates mit Nicht-Typ-Parametern.....	765
15.1.5 Explizit instanziierte Funktions-Templates Θ	767
15.1.6 Explizit spezialisierte und überladene Templates.....	768
15.1.7 Rekursive Funktions-Templates Θ	772
15.2 Generische Klassen: Klassen-Templates.....	775
15.2.1 Die Deklaration von Klassen-Templates mit Typ-Parametern.....	775
15.2.2 Spezialisierungen von Klassen-Templates.....	776
15.2.3 Klassen-Templates mit Nicht-Typ-Parametern.....	783
15.2.4 Explizit instanziierte Klassen-Templates Θ	785
15.2.5 Partielle und vollständige Spezialisierungen Θ	786
15.2.6 Vererbung mit Klassen-Templates Θ	792
15.2.7 Die Ableitung von Typ-Argumenten bei Klassen-Templates.....	793
15.2.8 Alias Templates Θ	797
15.3 Variablen-Templates Θ	798
15.4 Typ-Argument abhängige Templates mit Type Traits.....	800
15.4.1 Eine Konstruktion von type traits.....	800
15.4.2 Die type traits Kategorien.....	801
15.4.3 Type traits und <code>static_assert</code>	803
15.4.4 Templates mit <code>if constexpr</code> und type traits optimieren.....	805
15.4.5 Typ-Inferenz mit <code>decltype</code>	806
15.5 Variadische Templates.....	810
15.5.1 Variadische Funktions-Templates.....	811
15.5.2 Fold Ausdrücke.....	814
15.5.3 Variadische Klassen-Templates am Beispiel von <code>std::tuple</code>	816
16 STL-Algorithmen und Lambda-Ausdrücke.....	819
16.1 Iteratoren.....	819
16.1.1 Die verschiedenen Arten von Iteratoren.....	820
16.1.2 Umkehriteratoren.....	822

16.1.3 Einfügefunktionen und Einfügeiteratoren	823
16.1.4 Stream-Iteratoren	825
16.1.5 Container-Konstruktoren mit Iteratoren	827
16.1.6 Globale Iterator-Funktionen Θ	828
16.2 Nichtmodifizierende Algorithmen	830
16.2.1 Lineares Suchen	830
16.2.2 Zählen	832
16.2.3 Der Vergleich von Bereichen	833
16.2.4 Suche nach Teilfolgen	834
16.2.5 Minimum und Maximum	835
16.2.6 Mit <i>all_of</i> , <i>any_of</i> , <i>none_of</i> alle Elemente in einem Bereich prüfen	836
16.3 Kopieren und Verschieben von Bereichen	837
16.4 Elemente transformieren und ersetzen	838
16.5 Elementen in einem Bereich Werte zuweisen Θ	840
16.6 Elemente entfernen– das <i>erase-remove</i> Idiom	842
16.7 Die Reihenfolge von Elementen vertauschen	844
16.7.1 Elemente vertauschen	844
16.7.2 Permutationen Θ	845
16.7.3 Die Reihenfolge umkehren und Elemente rotieren Θ	846
16.7.4 Elemente durcheinander mischen Θ	847
16.8 Algorithmen zum Sortieren und für sortierte Bereiche	847
16.8.1 Partitionen Θ	847
16.8.2 Bereiche sortieren	848
16.8.3 Binäres Suchen in sortierten Bereichen	852
16.8.4 Mischen von sortierten Bereichen	853
16.9 Numerische Algorithmen und Datentypen	854
16.9.1 Numerische Algorithmen	855
16.9.2 Valarrays Θ	857
16.9.3 Zufallszahlen mit <i><random></i> Θ	859
16.9.4 Komplexe Zahlen Θ	862
16.9.5 Numerische Bibliotheken neben dem C++-Standard Θ	865
17 Zeiten und Kalenderdaten mit <i>chrono</i>	867
17.1 Brüche als Datentypen: Das Klassen-Template <i>ratio</i>	867
17.2 Ein Datentyp für Zeiteinheiten: <i>duration</i>	869
17.3 Datentypen für Zeitpunkte: <i>time_point</i>	872
17.4 Uhren: <i>system_clock</i> und <i>steady_clock</i>	875
18 Multithreading	879
18.1 Funktionen als Threads starten	881
18.1.1 Funktionen mit <i>thread</i> als Threads starten	881
18.1.2 Lambda-Ausdrücke als Threads starten	885
18.1.3 Zuweisungen und <i>move</i> für Threads	889
18.1.4 Exceptions in Threads	890
18.1.5 Informationen über Threads	892

- 18.1.6 *Sleep*-Funktionen..... 894
 - 18.1.7 Threads im Debugger..... 896
- 18.2 Kritische Abschnitte..... 897
 - 18.2.1 Atomare Datentypen..... 900
 - 18.2.2 Kritische Bereiche mit *mutex* und *lock_guard* bzw. *scoped_lock* sperren..... 902
 - 18.2.3 Read/Write-Mutexe mit *shared_mutex* und *shared_lock*..... 909
 - 18.2.4 Deadlocks..... 910
 - 18.2.5 *call_once* zur Initialisierung von Daten..... 912
 - 18.2.6 Thread-lokale Daten..... 913
- 19 Smart Pointer: *shared_ptr*, *unique_ptr* und *weak_ptr*..... 915**
 - 19.1 Gemeinsamkeiten von *unique_ptr* und *shared_ptr*..... 916
 - 19.2 *unique_ptr*..... 923
 - 19.3 *shared_ptr*..... 925
 - 19.4 Deleter und Smart Pointer für Arrays..... 931
 - 19.5 *weak_ptr* Θ..... 933
- 20 C++/CLI, .NET-Bibliotheken und C++ Interoperabilität 937**
 - 20.1 Native C++-Code in C# verwenden – .NET Framework DLLs..... 937
 - 20.2 Native C++-Code in C# verwenden – .NET DLLs..... 941
 - 20.3 C++/CLI Grundlagen..... 942
 - 20.3.1 Verweisklassen..... 942
 - 20.3.2 *System::String* und *std::string* konvertieren..... 944
 - 20.3.3 Garbage Collection und der GC-Heap..... 945
 - 20.3.4 Destruktoren und Finalisierer..... 947
 - 20.4 Assemblies..... 949
 - 20.4.1 Anwendungen und die *main*-Funktion..... 951
 - 20.4.2 DLLs..... 952
 - 20.4.3 Disassembler und Obfuscation..... 953
 - 20.5 .NET Klassen mit C++/CLI in Windows Forms verwenden..... 956
 - 20.5.1 Steuerelemente manuell erzeugen..... 957
 - 20.5.2 Ein Steuerelement manuell einem Formular hinzufügen..... 958
 - 20.6 Weitere Formulare und eigene Dialoge..... 958
 - 20.7 EMail's versenden..... 961
 - 20.8 Grafiken zeichnen..... 962
 - 20.9 Microsoft Office Anwendungen steuern..... 966
 - 20.9.1 Microsoft Office Word..... 968
 - 20.9.2 Excel..... 970
 - 20.10 WinAPI-Funktionen aus *windows.h* verwenden..... 971
- Literaturverzeichnis 975**
- Index..... 977**