Microsoft

developer

// Step by step

Microsoft Visual C++/CLI

Intermediate

Julian Templeman

Microsoft

Microsoft Visual C++/CLI Step by Step

Julian Templeman

Copyright © 2013 by Julian Templeman
All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-7356-7517-9

1 2 3 4 5 6 7 8 9 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Microsoft and the trademarks listed at *http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx* are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation,
nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions and Developmental Editor: Russell Jones

Production Editor: Kara Ebrahim

Technical Reviewer: Luca Regnicoli

Copyeditor: Octal Publishing, Inc.

Indexer: BIM Indexing and Proofreading Services

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: Rebecca Demarest

I would like to dedicate this book to my wife, Jane, without whose steadfast love and support none of this would be possible.

—Julian Templeman

Contents at a Glance

Introduction *xxi*

PART I GETTING STARTED WITH C++ .NET

CHAPTER 1 Hello C++! 3
CHAPTER 2 Introducing object-oriented programming 13
CHAPTER 3 Variables and operators 23
CHAPTER 4 Using functions 37
CHAPTER 5 Decision and loop statements 57
CHAPTER 6 More about classes and objects 77
CHAPTER 7 Controlling object lifetimes 103
CHAPTER 8 Inheritance 121

PART II MICROSOFT .NET PROGRAMMING BASICS

CHAPTER 9 Value types 143
CHAPTER 10 Operator overloading 159
CHAPTER 11 Exception handling 175
CHAPTER 12 Arrays and collections 197
CHAPTER 13 Properties 229
CHAPTER 14 Delegates and events 245
CHAPTER 15 The .NET Framework class library 263

PART III USING THE .NET FRAMEWORK

CHAPTER 16 Working with files 281
CHAPTER 17 Reading and writing XML 305
CHAPTER 18 Using ADO.NET 333
CHAPTER 19 Writing a service by using Windows Communication Foundation 351
CHAPTER 20 Introducing Windows Store apps 369
CHAPTER 21 More about Windows Store apps 397

PART IV ADVANCED TOPICS

CHAPTER 22	**Working with unmanaged code**	437
CHAPTER 23	**Attributes and reflection**	453
CHAPTER 24	**Living with COM**	475
	Index	*487*

Contents

Introduction . . . . *xxi*

PART I GETTING STARTED WITH C++ .NET

Chapter 1 Hello C++! 3

What is C++/CLI? 3

Your first C++/CLI application 4

The *main* function 4

C++ keywords and identifiers 5

Creating an executable application—theory 6

Editing the application source files 6

Compiling the source files 6

Running and testing the application 7

Creating an executable application—practice 7

Creating a project 8

Editing the C++ source code 9

Building the executable 9

Executing the application 11

Conclusion 11

Quick reference 11

Chapter 2 Introducing object-oriented programming 13

What is object-oriented programming? 13

Features of object-oriented programming languages 14

Encapsulation 14

Inheritance 15

Polymorphism 15

Classes and objects 16

Benefits to the development life cycle 16

A simple example 17

Quick reference 22

Chapter 3 Variables and operators 23

What is a variable? 23

The fundamental data types 23

Declaring a variable 25

Variable naming 25

Declaring multiple variables 26

Assigning values to variables 26

Handles and pointers 27

Arrays 28

Constants 28

Typedefs 29

The .NET Framework *String* class 29

Operators and expressions 30

Assignment operators 30

Arithmetic operators 30

Relational and logical operators 31

Bitwise operators 32

The ternary operator 33

Type casting 33

Operator precedence and associativity 34

Quick reference 35

Chapter 4 Using functions 37

Declaring function prototypes 38

Declaring a simple function prototype 38

Declaring parameters in a function prototype .39

Declaring the return type in a function prototype 39

Declaring default values for function parameters 40

Defining function bodies 41
Calling functions. 45
Stepping through the application by using debugger 47
Understanding local and global scope 51

Quick reference 55

Chapter 5 Decision and loop statements 57

Making decisions by using the *if* statement. 57
Performing one-way tests 57
Performing two-way tests 61
Performing multiway tests 62
Performing nested tests 64

Making decisions by using the *switch* Statement .65
Defining simple *switch* statements 65
Using fall-through in a *switch* statement. 67

Performing loops 68
Using *while* loops 68
Using *for* loops 70
Using *do-while* loops. 71
Performing unconditional jumps 73

Quick reference 75

Chapter 6 More about classes and objects 77

Organizing classes into header files and source files 78
Declaring a class in a header file. 79
Implementing a class in a source file 81

Creating objects 83

Initializing objects by using constructors. 84
Defining constructors 84
Member initialization lists. 86

Defining class-wide members. 87
Defining class-wide data members. 88
Defining class-wide member functions 90
Class constructors 92

Using constants in classes 93
Using class-wide constants. 93
Using instance constants 94

Defining object relationships 95
Defining the *LoyaltyScheme* Class. 95
Implementing the *LoyaltyScheme* class 96
Creating and using *LoyaltyScheme* objects. .97
Testing the application 100

Quick reference. 101

Chapter 7 Controlling object lifetimes 103

The .NET approach to object lifetimes 103

Destruction and finalization 105
Destructors 105
Finalizers 106
Implementing the destructor and finalizer for a class. 107
Objects and stack semantics 110
Copy constructors 113
Relating objects with stack semantics 116

Quick reference. 119

Chapter 8 Inheritance 121

What is inheritance?. 121
Inheritance terminology. 122
Inheritance and code reuse 122

Designing an inheritance hierarchy. 123
A word on substitutability 123

Defining a base class 124

Defining a derived class. 126

Creating derived class objects 129

Concrete and abstract classes. 130

Overriding member functions 131

Protected access 136

Defining sealed classes. 137

Abstract and sealed 137

Defining and using interfaces. 138

Quick reference 139

PART II MICROSOFT .NET PROGRAMMING BASICS

Chapter 9 Value types 143

Reference types and value types 143

The need for value types 144

Properties of value types 145

Structures 146

Creating and using a simple struct 146

Investigating the structure 147

The differences between structures and classes 149

Implementing constructors for a structure .149

Using one structure within another 150

Copying structures. 152

Enumerations. 153

Creating and using an enumeration. 153

Using enumerations in applications 155

Using memory efficiently 156

Quick reference 156

Chapter 10 Operator overloading 159

What is operator overloading?. 159

What types need overloaded operators? 160

What can you overload?. 160

Rules of overloading 161

Overloading operators in managed types 161

Overloading arithmetic operators. 161

Using static operator overloads. 163

What functions can you overload? .166
Implementing logical operators .167
Implementing increment and decrement .171
Operators and reference types .172
Guidelines for providing overloaded operators .173

Quick reference .174

Chapter 11 Exception handling 175

What are exceptions? .175
How do exceptions work? .177
Exception types .178

Throwing exceptions .178

Handling exceptions .180
Using the *try* and *catch* construct .180
Customizing exception handling .182
Using the exception hierarchy .183
Using exceptions with constructors .184
Nesting and rethrowing exceptions .185
The *finally* block .188
The *catch(...)* block .189

Creating your own exception types .189

Using *safe_cast* for dynamic casting .191

Using exceptions across languages .192

Quick reference .195

Chapter 12 Arrays and collections 197

Native C++ arrays .197
Passing arrays to functions .200
Initializing arrays .202
Multidimensional arrays .202
Dynamic allocation and arrays .203

Generic types .205

Managed arrays .207

The .NET array class 212

Basic operations on arrays 213

More advanced array operations 215

Using enumerators. 218

Other .NET collection classes 219

The *List<T>* class. 219

The *SortedList<K,V>* class 222

Generics and templates 224

The STL/CLR library 224

Quick reference. 227

Chapter 13 Properties 229

What are properties? 229

The two kinds of properties. 230

Implementing scalar properties 231

Errors in properties 232

Auto-implemented properties. 233

Read-only and write-only properties 233

Properties, inheritance, and interfaces. 235

Implementing indexed properties 236

The Bank example 236

Creating *Account* class properties. 239

Adding accounts to the *Bank* class 240

Implementing the *Add* and *Remove* methods 240

Implementing an indexed property to retrieve accounts. 241

Quick reference. 244

Chapter 14 Delegates and events 245

What are delegates?. 245

What is the purpose of delegates?. 246

Defining delegates. 247

Implementing delegates. 247

What are events? 253
Implementing an event source class. 254
Implementing an event receiver 256
Hooking it all together 258
Quick reference 262

Chapter 15 The .NET Framework class library 263

What is the .NET Framework? 263
The Common Language Runtime 264
The Microsoft Intermediate Language. 264
The Common Type System 264
The Common Language Specification 265
The .NET Framework class library 265
Assemblies. 266
Metadata 266
The .NET Framework namespaces 268
Using namespaces in C++ applications 270
The *System* namespace 270
The *Collections* namespaces 272
The *Collections* interfaces 273
The *Diagnostics* namespace 274
The *IO* namespace 274
The *Windows* namespaces 275
The *Net* namespaces 275
The *ServiceModel* namespaces 275
The *Xml* namespaces 276
The *Data* namespaces 276
The *Web* namespaces 277
Quick reference 278

PART III USING THE .NET FRAMEWORK

Chapter 16 Working with files 281

The *System::IO* namespace 282

Implementing text I/O by using readers and writers 283

Using *TextWriter* 283

The *FileStream* class 286

Using *TextReader* 287

Working with files and directories 290

Getting information about files and directories 290

Binary I/O 298

The *BinaryWriter* class 298

The *BinaryReader* class 299

Quick reference 303

Chapter 17 Reading and writing XML 305

XML and .NET 305

The .NET XML namespaces. 306

The XML processing classes 306

Parsing XML by using *XmlReader*. 307

Parsing XML with validation. 315

Writing XML by using *XmlTextWriter* 318

Using *XmlDocument*. 322

What is the W3C DOM? 323

The *XmlDocument* class 323

The *XmlNode* class 325

Quick reference 332

Chapter 18 Using ADO.NET 333

What is ADO.NET? 334

ADO.NET data providers. 334

ADO.NET namespaces. 335

ADO.NET assemblies 336

Creating a connected application 336
Connecting to a database 337
Creating and executing a command. 340
Executing a command that modifies data. .341
Executing queries and processing the results. 342
Creating a disconnected application. 344
Disconnected operation using a *DataSet* 345
Quick reference 350

Chapter 19 Writing a service by using Windows Communication Foundation 351

What is Windows Communication Foundation? 351
Distributed systems 352
Services 352
Connectivity 353
The ABCs of WCF. 353
Endpoints. 353
Address 354
Binding. 355
Contract 356
Message exchange patterns. 357
Behaviors 358
Creating a service 359
Writing a service client 361
Adding metadata to the service 363
Accessing a service by using a proxy 365
Quick reference 368

Chapter 20 Introducing Windows Store apps 369

A (brief) history of writing Windows user interface applications. 369
The Win32 API 369
Microsoft Foundation Classes 370
Windows Forms 370

Windows Presentation Foundation. 371
Windows 8 and Windows Store. 371
Which UI library to choose? 372

Introducing Windows Store apps. 372
Main features of Windows Store apps 373

Writing a Windows Store app. 374
Creating your first Windows Store app 375
Examining the project. 379

Introducing XAML. 380
What is XAML? 380
XAML syntax 381
XAML controls 382
Layout controls. 384
Event handling 389

C++/CX and Windows RT 389
Windows RT. 390
Metadata 390
C++/CX syntax 391
Common namespaces. 393

Quick reference. 395

Chapter 21 More about Windows Store apps 397

Building the basic calculator. 397
Laying out the number buttons 398
Handling number input 401
Adding arithmetic operations 403
Performing calculations 407
Testing the calculator 410
Improving the graphics. 412
Handling different number bases. 416
Using app bars 425
Adding sharing. 428
Where next? 433

Quick reference. 433

PART IV ADVANCED TOPICS

Chapter 22 Working with unmanaged code 437

Managed vs. unmanaged code 437
Mixed classes 437
The *GCHandle* type 438
Pinning and boxing. 440
Interior pointers 441
Pinning pointers 441
Boxing and unboxing 442
Boxing 443
Unboxing 443
Using P/Invoke to call functions in the Win32 API 444
The *DllImportAttribute* class. 447
Passing structures 449
Quick reference 452

Chapter 23 Attributes and reflection 453

Metadata and attributes 453
Using ILDASM 454
Using predefined attributes 457
The AssemblyInfo.cpp file. 457
Using the predefined attribute classes. 458
Defining your own attributes 461
Attribute class properties 463
Design criteria for attribute classes. 463
Writing a custom attribute 463
Using reflection to obtain attribute data 467
The *Type* class 467
Accessing standard attributes 469
Accessing custom attribute data 470
Quick reference 472

Chapter 24 Living with COM 475

COM components and the COM Interop 476

Using COM components from .NET code 476

How do RCWs work? 476

Creating and using RCWs 477

Handling COM errors 480

Late binding to COM objects. 481

Using .NET components as COM components 483

What must .NET types implement to be used as COM objects? . .483

Quick reference 485

Index 487

Introduction

C++ is a powerful, industrial-strength programming language used in tens of thousands of applications around the world, and this book will show you how to get started using C++ on Windows.

Of all the languages supported by Microsoft, C++ gives you access to the widest range of technologies on the Windows platform, from writing games, through low-level system software, to line-of-business applications. This book is going to introduce you to several of the areas in which C++ is used in Windows development.

For over a decade .NET has become established as the way to write desktop applications for Windows, and it provides a wealth of technologies to support developers. C++/CLI is the variant of C++ that runs in the .NET environment, and you can use it, along with other languages such as C#, to create rich desktop applications.

More recently, Windows 8 has introduced many new features to the Windows operating system, but perhaps the most exciting is the debut of Windows Store applications. These graphical applications are designed to run on touch screen and mobile devices, and provide a completely new way to construct user interfaces on Windows. C++ is one of the main languages supported for Windows Store development, and this book will give you an introduction to these applications and how to develop them in C++/CX, another variant of C++ introduced specifically for this purpose.

Who should read this book

This book exists to help programmers learn how to write applications using C++ on the Windows platform. It will be useful to those who want an introduction to writing .NET applications using C++, as well as to those who want to see how to write Windows Store applications.

If you are specifically interested in Windows Store applications, you may wish to look at *Build Windows 8 Apps with Microsoft Visual C++ Step by Step* by Luca Regnicoli, Paolo Pialorsi, and Roberto Brunetti, published by Microsoft Press.

Assumptions

This book expects that you have some experience of programming in a high-level language, so that you are familiar with concepts such as functions and arrays. It is quite sufficient to have experience in a procedural language such as Visual Basic, and I do not assume that you have any experience of object-oriented programming in general, or of C++ in particular (although any knowledge of a "curly bracket" language will be useful).

Who should not read this book

This book is not suitable for complete beginners to programming. For readers who are completely new to programming and want to learn C++, I recommend starting with a book such as *Programming: Principles and Practice Using C++* by Bjarne Stroustrup, published by Addison-Wesley.

This book is also not suitable for those who want to learn standard C++ or older-style Win32 development, because it concentrates on two Microsoft variants (C++/CLI and C++/CX) and does not cover topics such as the CLR or MFC in any detail.

Organization of this book

This book is divided into four sections.

- Part I, "Getting Started," introduces the main parts of the C++ language, getting you used to coding in C++ and building applications in Visual Studio 2012.
- Part II, "Microsoft .NET Programming Basics," continues by introducing those parts of C++ that are specific to Microsoft's C++/CLI language.
- Part III, "Using the .NET Framework," covers the main features in the .NET Framework libraries used for writing .NET applications. This part includes discussion of working with files, XML and databases, and creating graphical applications.
- Part IV, "Advanced Topics," covers some more advanced material, including details for working with legacy code.

Finding your best starting point in this book

The various sections of this book cover a wide range of technologies associated with C++ on the Windows platform. Depending on your needs and your existing understanding of C++, you may wish to focus on specific areas of the book. Use the following table to determine how best to proceed through the book.

If you are	Follow these steps
New to C++	Read Part I carefully before continuing to the rest of the book.
Familiar with OO programming but not with C++	Read Part I carefully, but you can omit Chapter 2.
Familiar with C++	Review Part I, looking for the differences between standard C++ and C++/CLI.
Familiar with .NET, but not Windows Store applications.	Read Chapters 20 and 21.

Most of the book's chapters include exercises that let you try out the concepts you have just learned. Solutions to these exercises can be downloaded using the companion code link from this book's web page. See the "Code samples" section for details on how to download the companion code.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow.

- Each exercise consists of a series of tasks, presented as numbered steps (1, 2, and so on) listing each action you must take to complete the exercise.
- Boxed elements with labels such as "Note" provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (e.g., File | Close) means that you should select the first menu or menu item, then the next, and so on.

System requirements

You will need the following hardware and software to complete the practice exercises in this book:

- One of Windows 7, Windows 8, Windows Server 2008 with Service Pack 2, or Windows Server 2008 R2. Note that if you want to build and run the Windows Store applications featured in Chapters 20 and 21, you will need Windows 8.
- Visual Studio 2012, any edition
- A computer that has a 1.6 GHz or faster processor (2 GHz is recommended)
- 1 GB (32 Bit) or 2 GB (64 Bit) RAM
- 3.5 GB of available hard disk space
- 5400 RPM hard disk drive
- DirectX 9 capable video card running at 1024 x 768 or higher-resolution display
- DVD-ROM drive (if installing Visual Studio from DVD)
- Internet connection to download software or chapter examples

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2012.

Code samples

Most of the chapters in this book include exercises that let you interactively try out new material learned in the main text. All sample projects, in both their pre-exercise and post-exercise formats, can be downloaded from the following page:

http://aka.ms/VCCLISbS/files

Acknowledgments

Producing a book involves a number of people, and I'd like to thank the following in particular.

I'd like to thank all at Microsoft Press and O'Reilly for their help and support, especially Devon Musgrave at Microsoft for inviting me to start this project, and Russell Jones at O'Reilly for providing so much help with writing and editorial matters, and especially his guidance in using the (not always good-tempered) Word templates.

The technical quality of the book has been greatly improved by Luca Regnicoli, who as tech reviewer pointed out numerous errors and omissions. I especially value his input on the Windows Store chapters.

Kara Ebrahim at O'Reilly, along with Dianne Russell and Bob Russell at Octal Publishing, provided excellent editorial support and made sure everything got done on time.

And lastly, I'd like to thank my family, who have put up with all the extra work involved in writing a book, and are probably hoping that this is last one for a while!

Errata and book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/VCCLISbS/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*

CHAPTER 1

Hello C++!

After completing this chapter, you will be able to

- Recognize C++ functions.
- Recognize C++ keywords and identifiers.
- Create a C++ application.

Welcome to the exciting world of programming Microsoft .NET with Microsoft Visual C++. This chapter introduces the C++/CLI language and shows you how to perform simple input/output (I/O).

What is C++/CLI?

C++/CLI is a version of the C++ programming language designed to run on the .NET Framework. It has been available since Microsoft Visual Studio 2005 and is the subject of an international standard. You can find details of the ECMA standard at *http://www.ecma-international.org/publications/standards/Ecma-372.htm*.

To achieve this, some changes had to be made to standard C++. There are some things that you can do in standard C++ that are not permitted in C++/CLI (for example, you cannot inherit from multiple base classes) and there have been some changes to the language geared to support .NET features (such as interfaces and properties) and to work with the .NET Runtime.

Why would you choose to use C++/CLI to write .NET code instead of another .NET language such as C#? Apart from personal preference, there are two very good reasons to choose C++/CLI. The first is for interoperability; C++/CLI makes it simple to incorporate standard C++ code into .NET projects. The second is that we have a .NET version of the C++ Standard Template Library (STL), and so people used to coding against the STL will find it possible to work in the same way in .NET.

Even if neither of these reasons applies to you, C++/CLI is still a perfectly good way to learn about .NET programming because it exposes all of the features that you need to write .NET programs and explore the .NET platform.

Your first C++/CLI application

It’s time to get our hands dirty with a simple C++/CLI application. Of course, no programming book would be complete without including the customary “Hello World” application, so let’s start with that.

```
using namespace System;

int main()
{
  Console::WriteLine("Hello, World!");
  return 0;
}
```

This short application illustrates some fundamental C++/CLI concepts:

- The first line (which begins with *using*) informs the compiler that you’re using the .NET *System* library. Many different libraries could be used in a single project; the *using* statement specifies to the compiler which library you want to use.
- The rest of the application is an example of a C++ *function*. All blocks of code in C++ are called functions—there’s no such thing as a procedure or a subroutine. Each C++ function contains the header (the first line of this application) and the function body (all of the text between the braces, { and }). The header shows the return type of the function (in this case *int*, short for *integer*), the name of the function (*main*), and the list of parameters inside round brackets. Note that you still need to include the round brackets even if you don’t have anything to pass to the function.
- All statements in C++ are terminated with a semicolon.

Of the six lines of code in the example application, only two contain C++ statements: the *Console* line and the *return* line. The *Console* line outputs characters to the console, and the argument to the function consists of the string that you want to output. The *return* line exits from the function—in this case, the application, because there is only one function—and returns zero, which is the standard value to return when execution is successful.

The *main* function

Why is the only function in the previous example called *main*? The simple answer is that the code won’t compile if it isn’t! However, it might be more useful to explain how the language works.

A normal C++ application contains many functions (and also many classes, as is discussed in Chapter 2, “Introducing object-oriented programming”). How does the compiler know which function should be called first? Obviously, you can’t allow the compiler to just randomly choose a function. The rule is that the compiler always generates code that looks for a function named *main*. If you omit the *main* function, the compiler reports an error and doesn’t create a finished executable application.

Free-format languages

C++ falls under the category of a *free-format* language, which means that the compiler ignores all spaces, carriage returns, new-line characters, tabs, form feeds, and so on. Collectively, these characters are referred to as *white space*. The only time the compiler recognizes white space is if it occurs within a string.

Free-format languages give the programmer great scope for using tab or space indenting as a way of organizing application layout. Statements inside a block of code—such as a *for* loop or an *if* statement—are typically indented (often by four space characters). This indentation helps the programmer's eye more easily pick out the contents of the block.

The free-format nature of C++ gives rise to one of the most common (and least useful) arguments in the C++ community: how do you indent the braces? Should they be indented with the code, or should they be left hanging at the beginning of the *if* or *for* statement? There is no right or wrong answer to this question (although some hardened C++ developers might disagree), but a consistent use of either style helps to make your application more readable to humans. As far as the compiler is concerned, your entire application could be written on one line.

So, the compiler will expect a function named *main*. Is that all there is to it? Well, not quite. There are some additional items, such as the return type and parameters being correct, but in the case of *main*, some of the C++ rules are relaxed. In particular, *main* can take parameters that represent the command-line arguments, but you can omit them if you don't want to use the command line.

C++ keywords and identifiers

A C++ *keyword* (also called a *reserved word*) is a word that means something to the compiler. The keywords used in the example application are *using*, *namespace*, and *return*. You're not allowed to use these keywords as variable or function names; the compiler will report an error if you do. You'll find that Visual Studio helps you identify keywords by displaying them in a special color.

An *identifier* is any name that the programmer uses to represent variables and functions. An identifier must start with a letter and must contain only letters, numbers, or underscores. The following are legal C++ identifiers:

- *My_variable*
- *AReallyLongName*

The following are not legal C++ identifiers:

Invalid identifier	Reason for being invalid
0800Number	Must not start with a number
You+Me	Must contain only letters, numbers, and underscores (the plus sign is the culprit here)
return	Must not be a reserved word

Outside of these restrictions, any identifier will work. However, some choices are not recommended, such as the following:

Identifier	Reason it's not recommended
main	Could be confused with the function main.
INT	Too close to the reserved word int.
B4ugotxtme	Just too cryptic!
_identifier1	Underscores at the beginning of names are allowed, but they are not recommended because compilers often use leading underscores when creating internal variable names, and they are also used for variables in system code. To avoid potential naming conflicts, you should not use leading underscores.

Creating an executable application—theory

Several stages are required to build an executable application; Microsoft Visual Studio 2012 helps you accomplish this by automating them. To examine and understand these stages, however, let's look at them briefly. You'll see these stages again later in the chapter when we build our first application.

Editing the application source files

Before you can create an application, you must write something. Visual Studio 2012 provides an integrated C++ editor, complete with color syntax highlighting and Microsoft IntelliSense to show function parameter information and provide word completion.

Compiling the source files

The C++/CLI compiler is the tool for converting text source files into something that can be executed by a computer processor. The compiler takes your source files (which usually have a *.cpp* extension) and builds them into either a stand-alone executable file (with a *.exe* extension) or a library file to be used in other projects (with a *.dll* extension).

Standard C++ and C

If you have ever worked with standard C++ or C, you might be familiar with the idea of compiling to object files and then linking with libraries to build the final executable file—which is commonly referred to simply as an executable. Although you can compile to the equivalent of an object file (called a *module* in the .NET world) and then link those together by using a tool called the *assembly linker*, Visual Studio takes you straight from source to executable without you seeing the intermediate step.

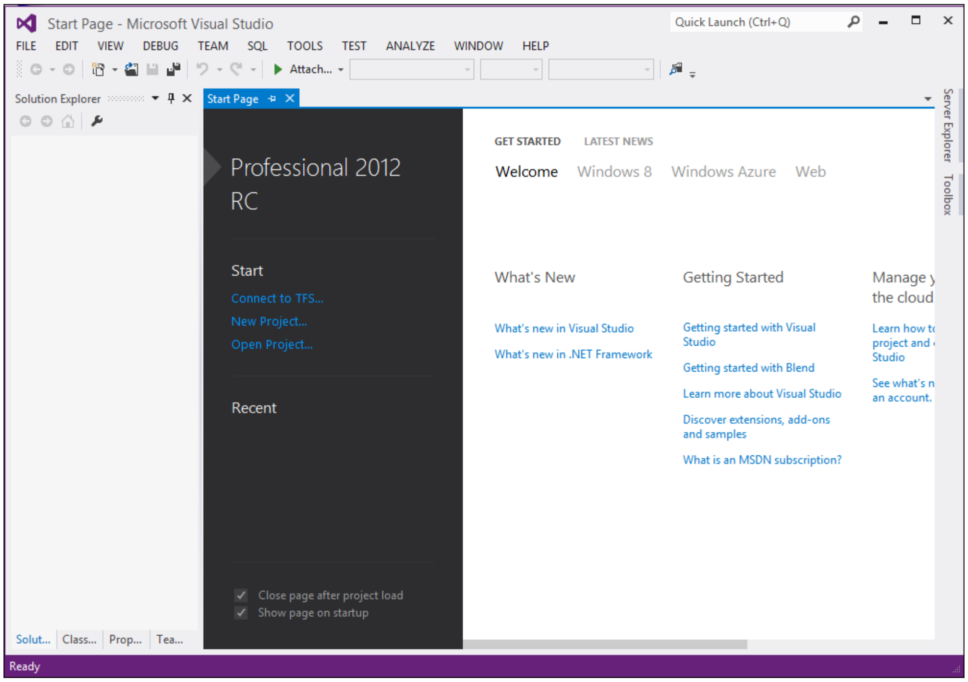
Running and testing the application

After you have successfully built the application, you need to run it and test it.

For many development environments, running and testing is often the most difficult part of the application development cycle. However, Visual Studio 2012 has yet another ace up its sleeve: the integrated debugger. The debugger has a rich set of features with which you can easily perform run-time debugging, such as setting *breakpoints* and *variable watches*.

Creating an executable application—practice

Go ahead and start Visual Studio 2012. An invitingly blank window appears on your screen.


Start Page - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Attach...
Solution Explorer
Start Page
Server Explorer
Toolbox
Professional 2012 RC
Start
Connect to TFS...
New Project...
Open Project...
Recent
Close page after project load
Show page on startup
GET STARTED
LATEST NEWS
Welcome
Windows 8
Windows Azure
Web
What's New
Getting Started
Manage y the cloud
What's new in Visual Studio
What's new in .NET Framework
Getting started with Visual Studio
Getting started with Blend
Learn more about Visual Studio
Discover extensions, add-ons and samples
What is an MSDN subscription?
Learn how to project and Studio
See what's n an account.
Solut... Class... Prop... Tea...
Ready

This window is the powerful Visual Studio integrated development environment (IDE). It contains all the tools you'll need to create full-featured, easy-to-use applications.

Note This book was written by using the Release Candidate (RC) version of Visual Studio 2012. As a result, screen shots and other details might differ from the version you're using when you read this.

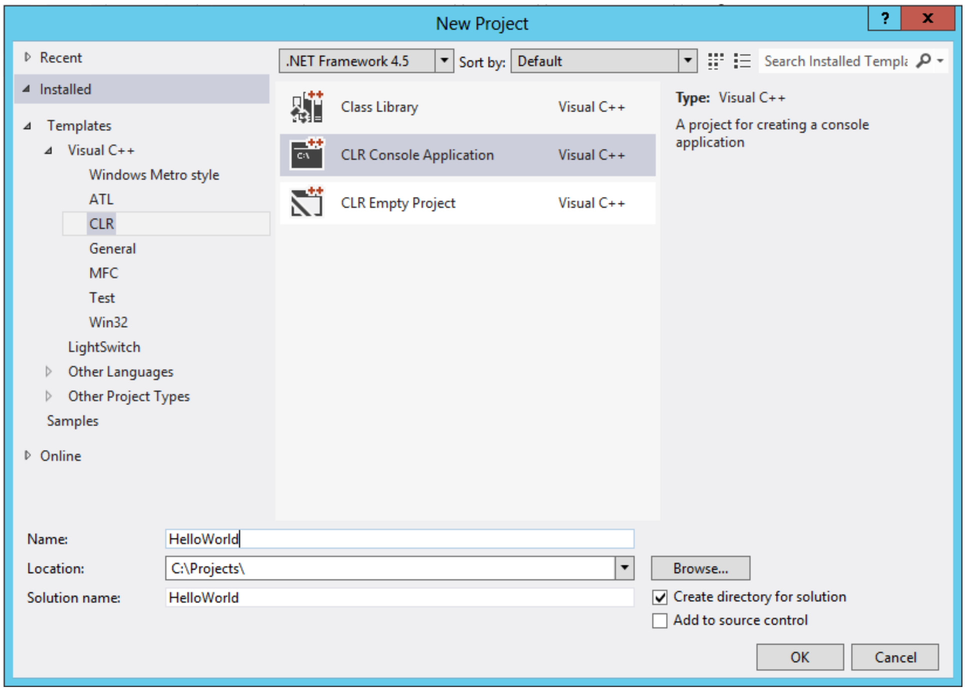
Creating a project

The first task is to create a new project for the “Hello, World” program.

1. In Visual Studio, on the File menu, point to New, and then click Project. (Alternatively, you can press Ctrl+Shift+N.)

Note I am using the Professional version of Visual Studio 2012. If you are using other versions, the way in which you create a project might be different. For example, in the Express version, you will find New Project on the File menu.

The New Project dialog box opens.



Type: Visual C++

A project for creating a console application

Name: HelloWorld

Location: C:\Projects\

Solution name: HelloWorld

☑ Create directory for solution

☐ Add to source control

2. In the navigation pane on the left, under Templates, click Visual C++, and then click CLR. In the center pane, click CLR Console Application and then, toward the bottom of the dialog box, in the Name box, type **HelloWorld**.

Note Depending on how Visual Studio has been set up, you might find Visual C++ under the Other Languages node.

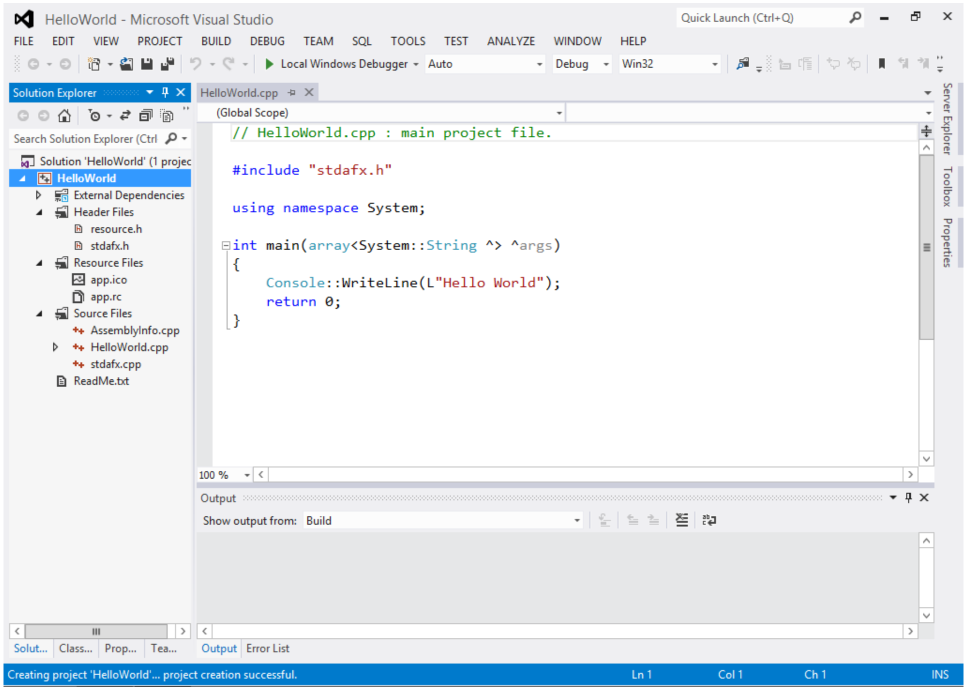
3. Click the Location list and select a location for your new project or click Browse and navigate to an appropriate directory.

4. Click OK to create the project.

   The wizard correctly initializes all the compiler settings for a console project.

Editing the C++ source code

The wizard creates a project for you with all the files needed for a simple console application. It also opens the main source file in the editor that contains just the code we want.



```
// HelloWorld.cpp : main project file.

#include "stdafx.h"

using namespace System;

int main(array<System::String ^> ^args)
{
    Console::WriteLine(L"Hello World");
    return 0;
}
```

Creating project 'HelloWorld'... project creation successful.

Notice that the keywords automatically appear in blue (provided that you spell them correctly).

There are a few things in the automatically generated source code that we don't need, so let's remove them. This will give you some practice in using the editor as well as making the code easier to understand. The application is not going to receive any command-line arguments when you run it, so remove everything between the opening and closing parentheses following *main*—in this example, *array<System::String ^> ^args*. In addition, the "L" before the *"Hello World"* string isn't necessary either (for reasons that I'll explain later), so you can remove that, as well.

Building the executable

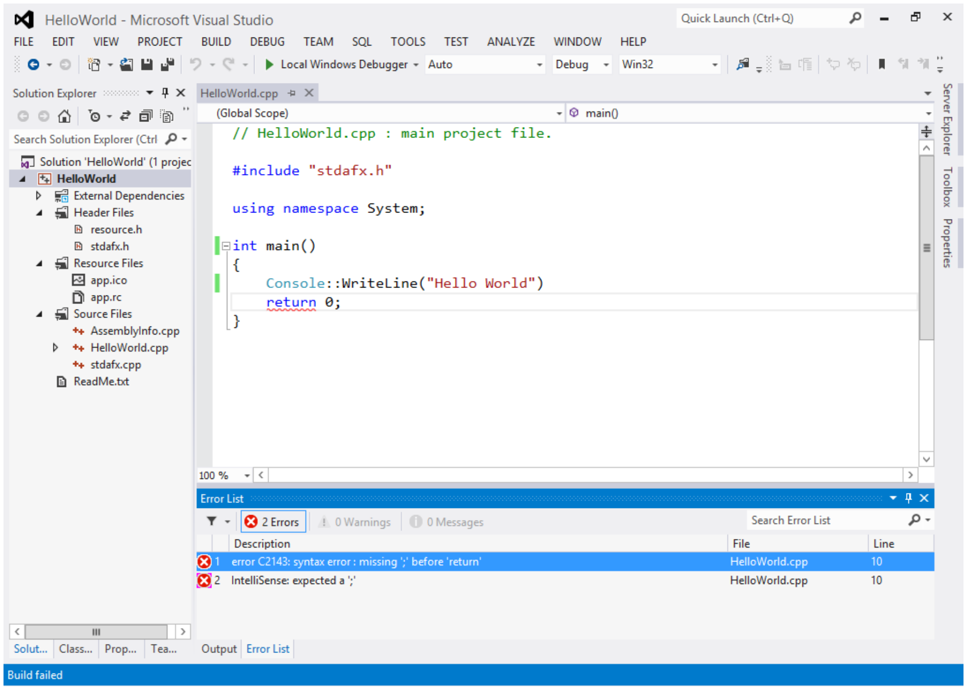
The next step is to build the executable. The term *build* in Visual Studio 2012 refers to compiling and linking the application. Visual Studio compiles any source files that have changed since the last build and—if no compile errors were generated—performs a link.

To build the executable, on the Build menu, click Build Solution or press F7.

Note The shortcut keys might differ depending on the version of Visual Studio you are using. For example, in the Ultimate edition, the shortcut is F6.

An Output window opens near the bottom of the Visual Studio window, showing the build progress. If no errors are encountered, the message *Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped* will appear in the Output window. If this window is closed, you can open it by selecting Output from the View menu.

If any problems occur, the Error List window will contain a list of errors and warnings.



```
// HelloWorld.cpp : main project file.

#include "stdafx.h"

using namespace System;

int main()
{
    Console::WriteLine("Hello World")
    return 0;
}
```

	Description	File	Line
1	error C2143: syntax error : missing ';' before 'return'	HelloWorld.cpp	10
2	IntelliSense: expected a ';'	HelloWorld.cpp	10

You can double-click the error line in the Error List window to place the cursor at the line in the source file where the compiler encountered the error. Fix the error (you might have misspelled a keyword or forgotten a semicolon) and rebuild the project. If the Error List pane is closed, you can open it by selecting Error List from the View menu.

How should you treat warnings?

Always treat warnings as errors—in other words, get rid of them. Warnings are there for a reason; they're telling you that your code is not correct.

Executing the application

After you've eliminated all errors and you've successfully built the project, you can finally execute the application. On the Debug menu, click Start Without Debugging to run the application. You can also press Ctrl+F5 to execute the application.

You'll see the output of your application, with the message "Press any key to continue" at the bottom of the output. This line is added by the IDE so that the console window doesn't simply disappear when the application has finished running.

Conclusion

Although the example in this chapter isn't the most exciting application ever written, it demonstrates some key C++ development points. It introduces the Visual Studio 2012 IDE and the ability to compile and link a application, and it serves as an introduction to the C++/CLI language.

Now, there's no turning back. Every new C++/CLI and Visual Studio 2012 feature that you learn about will fire your imagination to learn more and be increasingly productive. Software development is an exciting world.

Finally, don't forget to have some fun. Go back and try a few variations on the example application, click a few menus, and take some time to become familiar with the environment.

Quick reference

To	Do this
Create a new project in Visual Studio 2012.	Click File | New | Project, or press Ctrl+Shift+N. In the Express version, on the File menu, click New Project.
Add a file to a project.	Click File | New | File, or press Ctrl+N.
Build a Visual Studio 2012 project.	Click Build | Build Solution, or press Ctrl+Shift+B.
Execute a program from within Visual Studio 2012.	Click Debug | Start Without Debugging, or press Ctrl+F5.

CHAPTER 5

Decision and loop statements

After completing this chapter, you will be able to:

- Make decisions by using the *if* statement.
- Make multiway decisions by using the *switch* statement.
- Perform loops by using the *while*, *for*, and *do-while* statements.
- Perform unconditional jumps in a loop by using the *break* and *continue* statements.

All high-level languages provide keywords with which you can make decisions and perform loops. C++ is no exception. C++ provides the *if* statement and the *switch* statement for making decisions, and it provides the *while*, *for*, and *do-while* statements for performing loops. In addition, C++ provides the *break* statement to exit a loop immediately and the *continue* statement to return to the start of the loop for the next iteration.

In this chapter, you will see how to use these statements to control the flow of execution through a C++/CLI application.

Making decisions by using the *if* statement

The most common way to make a decision in C++/CLI is to use the *if* statement. You can use the *if* statement to perform a one-way test, a two-way test, a multiway test, or a nested test. Let's consider a simple one-way test first.

Performing one-way tests

The following example shows how to define a one-way test in C++/CLI:

```
if (number < 0)
    Console::WriteLine("The number is negative");
Console::WriteLine("The end");
```

The *if* keyword is followed by a conditional expression, which must be enclosed in parentheses. If the conditional expression evaluates to true, the next statement is executed, which in this example will display the message "The number is negative". Notice that the message "The end" will always be displayed, regardless of the outcome of the test, because it is outside the body of the *if* statement.

Note There is no semicolon after the closing parenthesis in the *if* test. It is a common C++ programming error to put one in by mistake, as shown here:

```
if (number < 0);   // Note the spurious semicolon
```

This statement is equivalent to the following statement, which is probably not what you intended:

```
if (number < 0)
    ;   // Null if-body - do nothing if number < 0
```

If you want to include more than one statement in the *if* body, enclose the if body in braces ({}), as follows:

```
if (number < 0)
{
    Console::Write("The number ");
    Console::Write(number);
    Console::WriteLine(" is negative");
}
Console::WriteLine("The end");
```

Many developers reckon that it is good practice to enclose the *if* body in braces, even if it only consists of a single statement. This means that the code will still be correct if you (or another developer) add more statements to the *if* body in the future.

In this exercise, you will create a new application to perform one-way tests. As this chapter progresses, you will extend the application to use more complex decision-making constructs and to perform loops. For now, the application asks the user to enter a date and then it performs simple validation and displays the date in a user-friendly format on the console.

1. Start Visual Studio 2012 and create a new CLR Console Application project. Name the application **CalendarAssistant**.

2. At the top of the source code file, immediately below the *using namespace System;* line, add the following function prototypes (you will implement all these functions during this chapter):

   ```
   int GetYear();
   int GetMonth();
   int GetDay(int year, int month);
   void DisplayDate(int year, int month, int day);
   ```

3. At the end of the file, after the end of the *main* function, implement the *GetYear* function as follows:

   ```
   int GetYear()
   {
       Console::Write("Year? ");
       String ^input = Console::ReadLine();
       int year = Convert::ToInt32(input);
       return year;
   }
   ```

4. Implement the *GetMonth* function as follows:

```
int GetMonth()
{
    Console::Write("Month? ");
    String ^input = Console::ReadLine();
    int month = Convert::ToInt32(input);
    return month;
}
```

   This is a simplified implementation; later in this chapter, you will enhance the function to ensure that the user enters a valid month.

5. Implement the *GetDay* function as follows:

```
int GetDay(int year, int month)
{
    Console::Write("Day? ");
    String ^input = Console::ReadLine();
    int day = Convert::ToInt32(input);
    return day;
}
```

   Later, you will enhance this function to ensure that the user enters a valid day for the given year and month.

6. Implement the *DisplayDate* function as shown in the following code to display the date as three numbers:

```
void DisplayDate(int year, int month, int day)
{
    Console::WriteLine("\nThis is the date you entered:");
    Console::Write(year);
    Console::Write("-");
    Console::Write(month);
    Console::Write("-");
    Console::Write(day);
    Console::WriteLine();
}
```

   Later in this chapter you will enhance this function to display the date in a more user-friendly format.

7. Add the following code inside the *main* method, immediately before the *return 0;* Line:

```
Console::WriteLine("Welcome to your calendar assistant");
Console::WriteLine("\nPlease enter a date");
int year = GetYear();
int month = GetMonth();
int day = GetDay(year, month);
```

```
// Simplified test for now - assume there are 31 days in
// every month :-)
if (month >= 1 && month <= 12 && day >= 1 && day <= 31)
{
    DisplayDate(year, month, day);
}
Console::WriteLine("\nThe end\n");
```

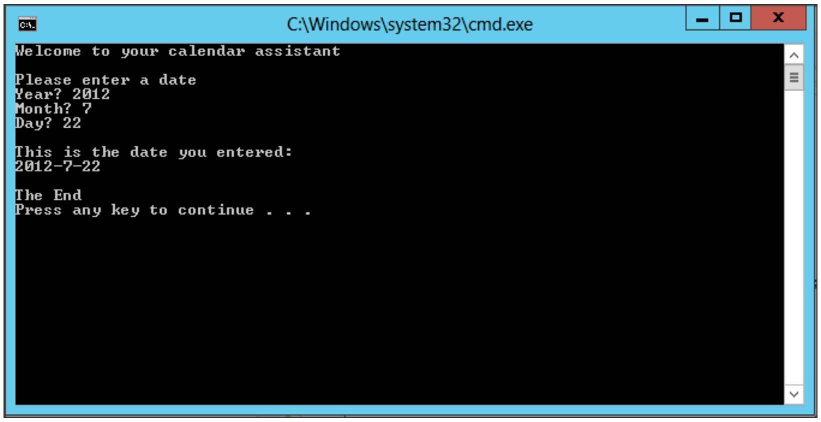
This code asks the user to enter a year, month, and day. If the date passes a simplified validation test, the date is displayed on the console. If the date is invalid, it is not displayed at all.

Note This *if* statement combines several tests by using the logical AND operator *&&*. As you learned in Chapter 3, "Variables and operators," logical tests are performed from left to right. Testing stops as soon as the final outcome has been established. For example, if the month is 0, there is no point performing the other tests—the date is definitely invalid. This is known as *short-circuit evaluation*.

8. Build the application and fix any compiler errors that you might have.

9. Run the application. Type in valid numbers for the year, month, and day (for example, **2012**, **7**, and **22**).

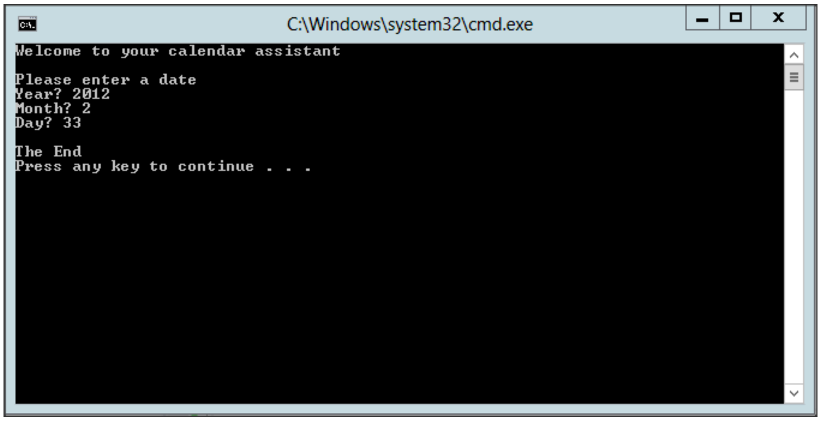
   The application displays the messages shown in the following screen shot:



```
C:\Windows\system32\cmd.exe
Welcome to your calendar assistant

Please enter a date
Year? 2012
Month? 7
Day? 22

This is the date you entered:
2012-7-22

The End
Press any key to continue . . .
```

   Observe that the application displays the date because it is valid. The message "The End" also appears at the end of the program.

10. Run the application again, but this time, type an invalid date (for example, **2012**, **2**, and **33**). The application displays the messages shown in the following screen shot:



```
Welcome to your calendar assistant

Please enter a date
Year? 2012
Month? 2
Day? 33

The End
Press any key to continue . . .
```

Notice that because the date you typed was invalid, the application doesn't display it. Instead, it just displays "The End." You can make the application more user-friendly by displaying an error message if the date is invalid. To do so, you need to use a two-way test.

Performing two-way tests

The following code shows how to define a two-way test for the Calendar Assistant application:

```
if (month >= 1 && month <= 12 && day >= 1 && day <= 31)
{
    DisplayDate(year, month, day);
}
else
{
    Console::WriteLine("Invalid date");
}
Console::WriteLine("\nThe end\n");
```

The *else* body defines what action to perform if the test condition fails.

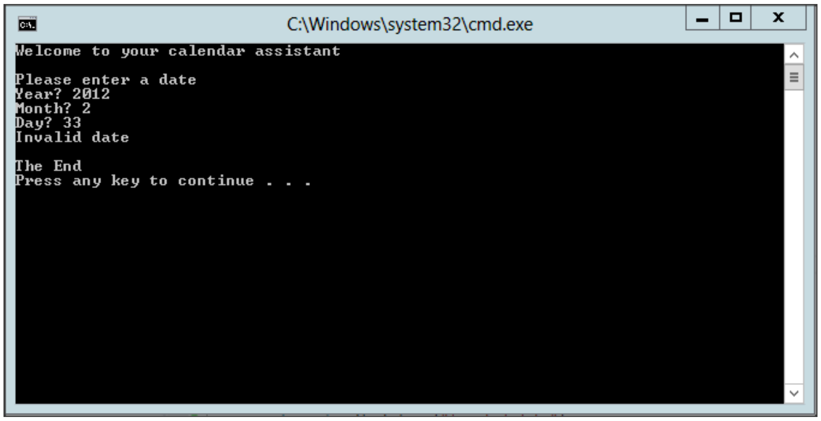
In this exercise, you will enhance your Calendar Assistant application to display an error message if an invalid date is entered.

1. Continue working with the project from the previous exercise.
2. Modify the *main* function, replacing the simple *if* with an *if-else* statement to test for valid or invalid dates.

```
if (month >= 1 && month <= 12 && day >= 1 && day <= 31)
{
    DisplayDate(year, month, day);
}
else
{
    Console::WriteLine("Invalid date");
}
Console::WriteLine("\nThe end\n");
```

3. Build and run the application. Type an invalid date such as **2001**, **0**, and **31**.

   The application now displays an error message, as demonstrated in the following screen shot:



```
Welcome to your calendar assistant

Please enter a date
Year? 2012
Month? 2
Day? 33
Invalid date

The End
Press any key to continue . . .
```

Performing multiway tests

You can arrange *if-else* statements in a cascading fashion to perform multiway decision making.

The following code shows how to use a multiway test to determine the maximum number of days (*maxDay*) in a month:

```
int maxDay;
if (month == 4 || month == 6 || month == 9 || month == 11)
{
    maxDay = 30;
}
else if (month == 2)
{
    maxDay = 28;
}
else
{
    maxDay = 31;
}
```

This code specifies that if the month is April, June, September, or November, set *maxDay* to 30. If the month is February, *maxDay* is set to 28. (We'll ignore leap years for now!) If the month is anything else, set *maxDay* to 31.

Note There is a space between the keywords *else* and *if* because they are distinct keywords. This is unlike Microsoft Visual Basic .NET, which uses the single keyword *ElseIf*.

In this exercise, you will enhance your Calendar Assistant application to display the maximum number of days in the user's chosen month.

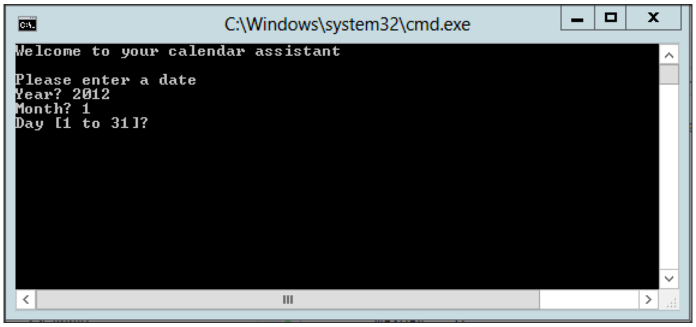
1. Continue working with the project from the previous exercise.

2. Replace the *GetDay* function with the following code so that it uses an *if-else-if* statement to determine the maximum allowable number of days.

```
int GetDay(int year, int month)
{
    int maxDay;
    if (month == 4 || month == 6 || month == 9 || month == 11)
    {
        maxDay = 30;
    }
    else if (month == 2)
    {
        maxDay = 28;
    }
    else
    {
        maxDay = 31;
    }
    Console::Write("Day [1 to ");
    Console::Write(maxDay);
    Console::Write("]? ");

    String ^input = Console::ReadLine();
    int day = Convert::ToInt32(input);
    return day;
}
```

3. Build and run the application. Type the year **2012** and the month **1**.

   The application prompts you to enter a day between 1 and 31, as illustrated in the following screen shot:

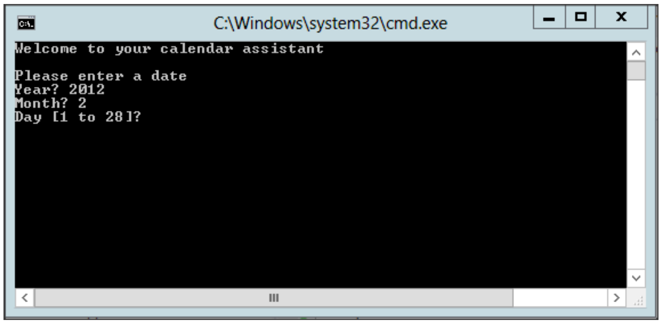


```
Welcome to your calendar assistant

Please enter a date
Year? 2012
Month? 1
Day [1 to 31]?
```

4. Type a valid day and close the console window when the date is displayed.

5. Run the application again. Type the year **2012** and the month **2**.

   The application prompts you to enter a day between 1 and 28, as shown here:



```
Welcome to your calendar assistant

Please enter a date
Year? 2012
Month? 2
Day [1 to 28]?
```

6. Type a valid day and close the console window when the date is displayed. (Don't worry about the date validation in *main:* You will remove it later and replace it with more comprehensive validation in the *GetMonth* and *GetDay* functions.)

Performing nested tests

It is possible to nest tests within one another. This makes it possible for you to perform more complex logical operations. The following code shows how to use nested tests to accommodate leap years correctly in the Calendar Assistant application:

```
int maxDay;
if (month == 4 || month == 6 || month == 9 || month == 11)
{
    maxDay = 30;
}
else if (month == 2)
{
    bool isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    if (isLeapYear)
    {
        maxDay = 29;
    }
    else
    {
        maxDay = 28;
    }
}
else
{
    maxDay = 31;
}
```

If the month is February, you define a *bool* variable to determine if the year is a leap year. A year is a leap year if it is evenly divisible by 4 but not evenly divisible by 100 (except years that are evenly divisible by 400, which are leap years). The following table shows some examples of leap years and non–leap years.

Year	Leap year?
1996	Yes
1997	No
1900	No
2000	Yes

You then use a nested *if* statement to test the *bool* variable *isLeapYear* so that you can assign an ap-propriate value to *maxDay*.

Note There is no explicit test in the nested *if* statement. The condition *if (isLeapYear)* is equivalent to *if (isLeapYear != false)*.

In this exercise, you will enhance your Calendar Assistant application to deal correctly with leap years.

1. Continue working with the project from the previous exercise.
2. Modify the *GetDay* function, replacing the *if...else if...else* statements to match the block of code just described to test for leap years.
3. Build and run the application. Type the year **1996** and the month **2**. The application prompts you to enter a day between 1 and 29. Type a valid day and then when the date is displayed, close the console window.
4. Run the application again. Type the year **1997** and the month **2**. Verify that the application prompts you to enter a day between 1 and 28.
5. Run the application several more times using the test data from the previous table.

Making decisions by using the *switch* Statement

Now that you have seen how the *if* statement works, let's take a look at the *switch* statement. Using the *switch* statement, you can test a single variable and execute one of several branches depending on the variable's value.

Defining simple *switch* statements

The example that follows shows the syntax for the *switch* statement. The *switch* statement tests the *numberOfSides* in a shape and displays a message to describe that shape.

```
int numberOfSides;   // Number of sides in a shape
...
switch (numberOfSides)
{
    case 3:  Console::Write("Triangle");      break;
    case 4:  Console::Write("Quadrilateral"); break;
    case 5:  Console::Write("Pentagon");      break;
    case 6:  Console::Write("Hexagon");       break;
    case 7:  Console::Write("Septagon");      break;
    case 8:  Console::Write("Octagon");       break;
    case 9:  Console::Write("Nonagon");       break;
    case 10: Console::Write("Decagon");       break;
    default: Console::Write("Polygon");       break;
}
```

The *switch* keyword is followed by an expression in parentheses. This expression must evaluate to an integer, a character, or an enumeration value. The body of the switch consists of a series of case branches, each of which comprises the keyword *case*, a value, and a colon.

The value identifying a case branch must be a constant of integer type. This means that integer numbers, enumeration values, and characters are allowed. For example, *5* and *a* are valid, but *abc* is not because it is a string literal.

Note Each case label specifies a single literal value. You can't specify multiple values, you can't define a range of values, and the values must be known at compile time. This means that you can't, for instance, say *case foo*, where *foo* is a variable whose value will only be known when the application executes.

Each case branch can contain any number of statements. At the end of each branch, use a *break* statement to exit the *switch* statement.

Note There is normally no need to use braces around the code in a case branch. The break statement marks the end of each case branch. However, you do need to use braces if you need to declare a variable within the branch code.

You can define an optional *default* branch in the *switch* statement. The *default* branch will be executed if the expression doesn't match any of the case labels.

Tip It's good practice to define a *default* branch even if you don't have any specific processing to perform. Including the *default* branch shows that you haven't just forgotten it. Also, the *default* branch can help you trap unexpected values and display a suitable warning to the user.

In this exercise, you will enhance your Calendar Assistant application to display the month as a string such as January or February.

1. Continue working with the project from the previous exercise.
2. Modify the *DisplayDate* function. Rather than display the month as an integer, replace the *Console::Write(month)* statement with a *switch* statement that displays the month as a string.

   ```
   switch (month)
   {
       case 1:  Console::Write("January");   break;
       case 2:  Console::Write("February");  break;
       case 3:  Console::Write("March");     break;
       case 4:  Console::Write("April");     break;
       case 5:  Console::Write("May");       break;
       case 6:  Console::Write("June");      break;
       case 7:  Console::Write("July");      break;
       case 8:  Console::Write("August");    break;
       case 9:  Console::Write("September"); break;
       case 10: Console::Write("October");   break;
       case 11: Console::Write("November");  break;
       case 12: Console::Write("December");  break;
       default: Console::Write("Unknown");   break;
   }
   ```

3. Build the application.
4. Run the application several times, typing a different month each time. Verify that the application displays the correct month name each time.

Using fall-through in a *switch* statement

If you omit the *break* statement at the end of a case branch, flow of control continues on to the next statement. This process is called *fall-through*. This can be useful to avoid duplication of code, but be careful not to do it accidentally.

The following example illustrates why fall-through might be useful. This example tests a lowercase letter to see if it is a vowel or a consonant:

```
char lowercaseLetter;   // Single lowercase letter, for example 'a'
...
switch (lowercaseLetter)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':  Console::Write("Vowel"); break;

    default:   Console::Write("Consonant"); break;
}
```

There is no *break* statement in the first four case labels. As a result, the flow of control passes on to the next executable statement to display the message *Vowel*. The *default* branch deals with all the other letters and displays the message *Consonant*.

In this exercise, you will enhance your Calendar Assistant application to display the season for the user's date.

1. Continue working with the project from the previous exercise.
2. Modify the *DisplayDate* function. After displaying the year, month, and day, add the following code after the line *Console::Write(day)* to display the season:

```
switch (month)
{
    case 12:
    case 1:
    case 2:  Console::WriteLine(" [Winter]"); break;

    case 3:
    case 4:
    case 5:  Console::WriteLine(" [Spring]"); break;

    case 6:
    case 7:
    case 8:  Console::WriteLine(" [Summer]"); break;

    case 9:
    case 10:
    case 11: Console::WriteLine(" [Fall]"); break;
}
```

3. Build the application.
4. Run the application several times, typing a different month each time. Verify that the application displays the correct season name each time.

Performing loops

For the rest of this chapter, you'll see how to perform loops in C++/CLI. You'll also see how to perform unconditional jumps in a loop by using the *break* and *continue* statements.

C++ has three main loop constructs: the *while* loop, the *for* loop, and the *do-while* loop.

Note There is actually a fourth loop type, the *for-each* loop, but I'll leave discussing that until we get to arrays.

Let's look at the *while* loop first.

Using *while* loops

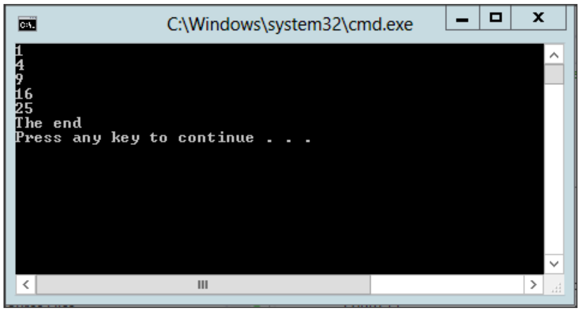
A *while* loop continues executing its body for as long as the condition in parentheses evaluates to true. The following example shows how to write a simple *while* loop in C++/CLI:

```
int count = 1;
while (count <= 5)
{
    Console::WriteLine(count * count);
    count++;
}
Console::WriteLine("The end");
```

You must follow the *while* keyword with a conditional expression enclosed in parentheses. As long as the conditional expression evaluates to true, the *while* body executes. After the loop body has been executed, control returns to the *while* statement and the conditional expression is tested again. This sequence continues until the test evaluates to false.

You must, of course, remember to include some kind of update statement in the loop so that it will terminate eventually. In this example *count++* is incrementing the loop counter. If you don't provide an update statement, the loop will iterate forever, which probably isn't what you want.

The preceding example displays the following output:



```
1
4
9
16
25
The end
Press any key to continue . . .
```

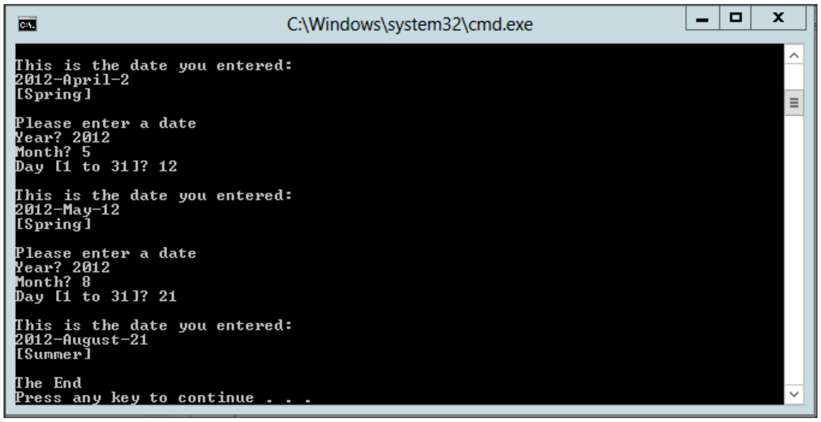
In this exercise, you will enhance your Calendar Assistant application so that the user can type five dates.

1. Continue working with the project from the previous exercise.

2. Modify the code in the *main* function by replacing the entire body of the function with the following code:

```
Console::WriteLine("Welcome to your calendar assistant");

int count = 1;    // Declare and initialize the loop counter
while (count <= 5)   // Test the loop counter
{
    Console::Write("\nPlease enter a date ");
    Console::WriteLine(count);

    int year = GetYear();
    int month = GetMonth();
    int day = GetDay(year, month);
    DisplayDate(year, month, day);

    count++;   // Increment the loop counter
}
```

3. Build and run the application. The application prompts you to enter the first date. After you have typed this date, the application prompts you to enter the second date. This process continues until you have typed five dates, at which point the application closes, as depicted in the following screen shot:



```
This is the date you entered:
2012-April-2
[Spring]

Please enter a date
Year? 2012
Month? 5
Day [1 to 31]? 12

This is the date you entered:
2012-May-12
[Spring]

Please enter a date
Year? 2012
Month? 8
Day [1 to 31]? 21

This is the date you entered:
2012-August-21
[Summer]

The End
Press any key to continue . . .
```

Using *for* loops

The *for* loop is an alternative to the *while* loop. It provides more control over the way in which the loop executes.

The following example shows how to write a simple *for* loop in C++/CLI. This example has exactly the same effect as the *while* loop.

```
for (int count = 1; count <= 5; count++)
{
    Console::WriteLine(count * count);
}

Console::WriteLine("The end");
```

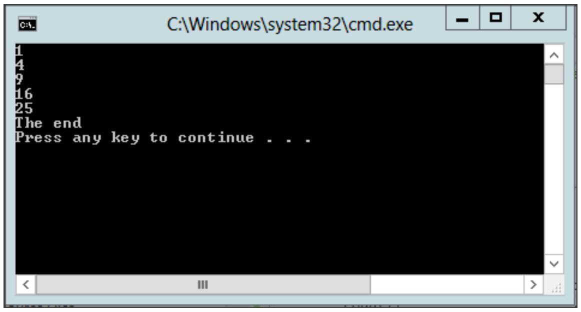
The parentheses after the *for* keyword contain three expressions separated by semicolons. The first expression performs loop initialization, such as initializing the loop counter. This initialization expression is executed once only, at the start of the loop.

Note You can declare loop variables in the first expression of the *for* statement. The preceding example illustrates this technique. The count variable is local to the *for* statement and goes out of scope when the loop terminates.

The second expression statement defines a test. If the test evaluates to *true*, the loop body is executed, but if it is *false*, the loop finishes and control passes to the statement that follows the closing parenthesis. After the loop body has been executed, the final expression in the *for* statement is executed; this expression performs loop update operations, such as incrementing the loop counter.

Note The *for* statement is very flexible. You can omit any of the three expressions in the *for* construct as long as you retain the semicolon separators. You can even omit all three expressions, as in *for(; ;)*, which represents an infinite loop

The preceding example displays the output shown in the following screen shot.



```
1
4
9
16
25
The end
Press any key to continue . . .
```

In this exercise, you will modify your Calendar Assistant application so that it uses a *for* loop rather than a *while* loop to obtain five dates from the user.

1. Continue working with the project from the previous exercise.
2. Modify the code in the *main* function to use a *for* loop rather than a *while* loop, as shown here:

```
Console::WriteLine("Welcome to your calendar assistant");

for (int count = 1; count <= 5; count++)
{
    Console::Write("\nPlease enter date ");
    Console::WriteLine(count);

    int year = GetYear();
    int month = GetMonth();
    int day = GetDay(year, month);
    DisplayDate(year, month, day);
}
```

   Notice that there is no *count++* statement after displaying the date. This is because the *for* statement takes care of incrementing the loop counter.

3. Build and run the application. The application asks you to enter five dates, as before.

Using *do-while* loops

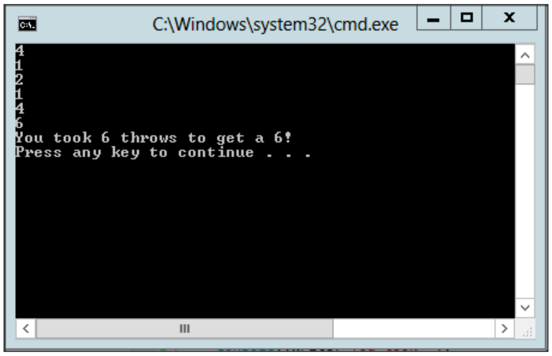
The third loop construct you'll look at here is the *do-while* loop (remember, there's still the *for-each* loop, which you will meet later). The *do-while* loop is fundamentally different from the *while* and *for* loops because the test comes at the end of the loop body, which means that the loop body is always executed at least once.

The following example shows how to write a simple *do-while* loop in C++/CLI. This example generates random numbers between 1 and 6, inclusive, to simulate a die. It then counts how many throws are needed to get a 6.

```
Random ^r = gcnew Random();
int randomNumber;
int throws = 0;
do
{
    randomNumber = r->Next(1, 7);
    Console::WriteLine(randomNumber);
    throws++;
}
while (randomNumber != 6);

Console::Write("You took ");
Console::Write(throws);
Console::WriteLine(" tries to get a 6");
```

The loop starts with the *do* keyword, followed by the loop body, followed by the *while* keyword and the test condition. A semicolon is required after the closing parenthesis of the test condition.

The preceding example displays the output shown in the following screen shot:



```
4
1
2
1
4
6
You took 6 throws to get a 6!
Press any key to continue . . .
```

In this exercise, you will modify your Calendar Assistant application so that it performs input validation, which is a typical use of the *do-while* loop.

1. Continue working with the project from the previous exercise.
2. Modify the *GetMonth* function as follows, which forces the user to type a valid month:

```
int GetMonth()
{
    int month = 0;
    do
    {
        Console::Write("Month [1 to 12]? ");
        String ^input = Console::ReadLine();
        month = Convert::ToInt32(input);
    }
    while (month < 1 || month > 12);
    return month;
}
```

3. Modify the *GetDay* function as follows, which forces the user to type a valid day:

```
int GetDay(int year, int month)
{
    int day = 0;
    int maxDay;

    // Calculate maxDay, as before (code not shown here) … … …

    do
    {
        Console::Write("Day [1 to ");
        Console::Write(maxDay);
        Console::Write("]? ");
        String ^input = Console::ReadLine();
        day = Convert::ToInt32(input);
    }
    while (day < 1 || day > maxDay);
    return day;
}
```

4. Build and run the application.
5. Try to type an invalid month. The application keeps asking you to enter another month until you type a value between 1 and 12, inclusive.
6. Try to type an invalid day. The application keeps asking you to enter another day until you type a valid number (which depends on your chosen year and month).

Performing unconditional jumps

C++/CLI provides two keywords—*break* and *continue*—with which you can jump unconditionally within a loop. The *break* statement causes you to exit the loop immediately. The *continue* statement abandons the current iteration and goes back to the top of the loop ready for the next iteration.

Note The *break* and *continue* statements can make it difficult to understand the logical flow through a loop. Use *break* and *continue* sparingly to avoid complicating your code unnecessarily.

In this exercise, you will modify the main loop in your Calendar Assistant application. You will give the user the chance to break from the loop prematurely, skip the current date and continue on to the next one, or display the current date as normal.

1. Continue working with the project from the previous exercise.

2. Modify the *main* function as follows, which gives the user the option to *break* or *continue* if desired:

```
Console::WriteLine("Welcome to your calendar assistant");
for (int count = 1; count <= 5; count++)
{
    Console::Write("\nPlease enter date ");
    Console::WriteLine(count);
    int year = GetYear();
    int month = GetMonth();
    int day = GetDay(year, month);

    Console::Write("Press B (break), C (continue), or ");
    Console::Write("anything else to display date ");
    String ^input = Console::ReadLine();
    if (input->Equals("B"))
    {
        break;
    }
    else if (input->Equals("C"))
    {
        continue;
    }
    DisplayDate(year, month, day);
}
```

Note The *Equals* method is used here to check that two strings contain the same content. You will see another (and more idiomatic) way to do this using the == operator when we discuss operator overloading.

3. Build and run the application.

4. After you type the first date, you are asked whether you want to break or continue. Press X (or any other key except B or C) and then press Enter to display the date as normal.

5. Type the second date, and then press C followed by Enter, which causes the *continue* statement to be executed.

   The *continue* statement abandons the current iteration without displaying your date. Instead, you are asked to type the third date.

6. Type the third date and then press B, which causes the *break* statement to be executed. The break statement terminates the entire loop.

Quick reference

To	Do this

Perform a one-way test.	Use the *if* keyword followed by a test enclosed in parentheses. You must enclose the *if* body in braces if it contains more than one statement. For example:
	`if (n < 0)`
	`{`
	`    Console::Write("The number ");`
	`    Console::Write(n);`
	`    Console::WriteLine(" is negative");`
	`}`

Perform a two-way test.	Use an *if-else* construct. For example:
	`if (n < 0)`
	`{`
	`    Console::Write("Negative");`
	`}`
	`else`
	`{`
	`    Console::Write("Not negative");`
	`}`

Perform a multiway test.	Use an *if-else-if* construct. For example:
	`if (n < 0)`
	`{`
	`    Console::Write("Negative");`
	`}`
	`else if (n == 0)`
	`{`
	`    Console::Write("Zero");`
	`}`
	`else`
	`{`
	`    Console::Write("Positive");`
	`}`

Test a single expression against a finite set of constant values.	Use the *switch* keyword followed by an integral expression enclosed in parentheses. Define case branches for each value you want to test against, and define a default branch for all other values. Use the *break* statement to close a branch. For example:
	`int dayNumber; // 0=Sun, 1=Mon, etc.`
	`…`
	`switch (dayNumber)`
	`{`
	`case 0:`
	`case 6:`
	`    Console::Write("Weekend");`
	`    break;`
	`default:`
	`    Console::Write("Weekday");`
	`    break;`
	`}`

Perform iteration by using the *while* loop.	Use the *while* keyword followed by a test enclosed in parentheses. For example:
	`int n = 10;`
	`while (n >= 0)`
	`{`
	`    Console::WriteLine(n);`
	`    n--;`
	`}`

To	Do this

Perform iteration by using the *for* loop.	Use the *for* keyword followed by a pair of parentheses. Within the parentheses, define an initialization expression, followed by a test expression, followed by an update expression. Use semicolons to separate these expressions. For example:
	
	`for (int n = 10; n >= 0; n--)`
	`{`
	`    Console::WriteLine(n);`
	`}`

Perform iteration by using the *do-while* loop.	Use the *do* keyword, followed by the loop body, followed by the *while* keyword and the test condition. Terminate the loop with a semicolon. For example:
	
	`int n;`
	`do`
	`{`
	`    String^ input = Console::ReadLine();`
	`    n = Convert::ToInt32(input);`
	`} while (n > 100);`

Terminate a loop prematurely.	Use the *break* statement inside any loop. For example:
	
	`for (int n = 0; n < 1000; n++)`
	`{`
	`    int square = n * n;`
	`    if (square > 3500)`
	`    {`
	`        break;`
	`    }`
	`    Console::WriteLine(square);`
	`}`

Abandon a loop iteration and continue with the next iteration.	Use the *continue* statement inside any loop. For example:
	
	`for (int n = 0; n < 1000; n++)`
	`{`
	`    int square = n * n;`
	`    if (square % 2 == 0)`
	`    {`
	`        continue;`
	`    }`
	`    Console::WriteLine(square);`
	`}`

Index

Symbols

+ (addition operator), 30
& (ampersand character), 175
& (AND operator), 296
&& (AND operator), 31
= (assignment operator), 26
* (asterisk) symbol, 438
^ (caret) symbol, 391
~ (complement operator), 32
%d descriptor, 422
/ (division operator), 30
. (dot operator), 20
:: (double-colon syntax), 79
= (equal sign), 236, 254
#include directive, 439
#include statements, 107
__int8 type, 24
__int16 type, 24
__int32 type, 24
__int64 type, 24
<< (left-shift operator), 32
<= (less-than-or-equal-to) condition, 199
% (modulus operator), 30
* (multiplication operator), 30
! (NOT operator), 31
-= operator, 252, 254
-> operator, 83
+ operator, 252
+= operator, 251, 254
-> (pointer operator), 28
#pragma once directive, 125
#pragma once line, 240
>> (right-shift operator), 32
:: (scope resolution operator), 269
[ ] (square brackets), 267
- (subtraction operator), 30
<> syntax, 207
%x descriptor, 422

A

Abs function, 169
Abstract attribute, 469
abstract classes
    and sealed classes, 137
    overview, 130–131
Account class, 14
    in bank example, 238–240
AddAccount method, 240
Add function, 221
addition operator (+), 30
Add method
    in bank example, 240–241
Add New Item dialog box, 107
Add New Reference button, 461
add_OnFirstEvent method, 256
addresses, WCF, 355
Add Service dialog box, 364
AddServiceEndpoint, 363
ADO.NET
    assemblies, 336
    connected application, creating
        connecting to database, 337–341
        creating and executing command, 340–341
        executing command that modifies data, 341–342
        executing queries and processing results, 342–343
        overview, 336–337
    data providers, 334–335
    disconnected application, creating, 344–345
    disconnected operation using DataSet, 345–350
    namespaces, 335

overview, 334–336
quick reference, 350, 368
Age property, 232
aggregate initializer, 202
algorithms, 226
All Apps charm, 378
ampersand character (&), 175
AND operator (&), 296
AND operator (&&), 31
Animal class, 16
AnsiClass attribute, 469
API (application programming interface), Windows, 265
AppBarButton style, 427
app bars
AppBar control, 383
in calculator example, 425–428
AppendAllLines method, 292
AppendAllText method, 292
AppendChild method, 324, 326
AppendText method, 292, 293
App.g.cpp, 380
App.g.h, 380
application programming interface (API), Windows, 265
Application UI tab, 412
ArgumentException, 222, 232, 239
ArithmeticButtons_Click method, 404, 406
ArithmeticException, 184
arithmetic operators, 30–31
overloading, 161–162
arity, 161
array keyword, 207
Array::Reverse method, 217
arrays
managed arrays
and reference types, 208–210
initializing, 208
multidimensional, 211
overview, 207–208
using for each loop with, 210–211
native
dynamic allocation of, 203–205
initializing, 202
multidimensional, 202–203
overview, 197–199
passing to functions, 200–202
overview, 28
System::Array class
basic operations using, 213–215
copying elements, 215
overview, 212
searching, 216–217
sorting, 217–218
using enumerators with, 218–219
Array::Sort method, 217
AsReadOnly method, 212
assemblies
ADO.NET, 336
.NET, 266
AssemblyCompanyAttribute, 267, 268
AssemblyInfo.cpp file
predefined attributes, 457–458
assembly linker, 6
assembly manifest, 266
Assembly property, 468
AssemblyQualifiedName property, 468
Assets folder, 379
assigning variables, 26–27
assignment conversions, 26
assignment operator (=), 26, 30
asterisk (*) symbol, 438
attached properties, 386–387
AttributeCount property, 307, 315
Attribute node type, 309
attributes
and metadata
overview, 453–454
using ILDASM, 454–457
defining custom
creating, 463–467
design criteria for attribute classes, 463–464
overview, 461–463
properties for, 463–464
predefined attributes
AssemblyInfo.cpp file, 457–458
classes for, 458–461
overview, 457
using reflection to obtain
accessing custom attribute data, 470–472
accessing standard attributes, 469
overview, 467–468
Type class, 467–469
Attributes property, 291, 323, 326, 468
AttributeTargets enumeration, 464
AttributeUsage attribute, 462, 464
AttributeUsageAttribute class, 458, 462
AutoClass attribute, 469
auto_gcroot type, 440
auto-implemented properties, 233

automatic layout, 449
Average function, 53

B

backing variable, 233
Balance property, 241
BankAccount class, 123, 124, 130, 132, 136
bank example
  adding Account class, 238–239
  Bank class
    adding accounts to, 240–243
    implementing, 236–238
  overview, 236
base addresses, 355
BaseButtons_Click method, 418
base classes, 126–129
BaseStream property, 298, 302, 318
BaseType property, 468
base variable, 419
BasicHttpBinding, 355, 358
behavior of Windows Store apps, 373
behaviors, WCF, 358–359
Berkeley Sockets protocol, 275
BestFitMapping field, 447
binary I/O
  BinaryReader class, 299–304
  BinaryWriter class, 298
  overview, 298
binary operator, 162
BinaryReader class, 274, 282, 299–303
BinarySearch method, 212
BinaryWriter class, 274, 282, 298
binding, WCF, 355
bitwise operators, 32–33
blocking, 283
Boolean type, 271
Boolean value type, 145
bool type, 24
Border control, 383
BorderThickness property, 399
BottomAppBar element, 428
boxing 443
  unboxing, 443–444
boxing process, 171
break keyword, 73
breakpoints, 47
BufferedStream class, 282
buffer overrun, 200
Button control, 383
Button element, 376
Byte type, 271
Byte value type, 144

C

Cached File Updater contract, Windows 8, 429
calculator example
  adding tile, 412–415
  app bars, 425–428
  arithmetic buttons, 403–404
  getting number from button, 404–405, 407–408
  handling different number bases
    adding buttons for, 417–418
    changing base, 418–421
    converting string in display, 421–425
  handling number input, 401–402
  laying out number buttons, 398–401
  overview, 397–398
  performing calculations, 408–409
  remembering operations, 406
  sharing in
    contracts and charms, 428–429
    DataPackage class, 430
    handling requests, 431–432
    implementing, 429–430
    overview, 428
  testing, 410–412
Calendar Assistant application, 61
CallingConvention field, 447, 448
calling functions, 45–47
CallMe method, 261
CanDebit method, 123, 133, 134
Canvas control, 388
Capacity property, 220
caret (^) symbol, 391
casting process, 26
cast operator, overview, 33–34
catch block, 347
  handling exceptions using, 180–182, 189
C++/CLI
  defined, 3
  Hello World example, 4
  identifiers in, overview, 5–6
  keywords in, overview, 5–6
  main function in, overview, 4–5
CCWs (COM Callable Wrappers), 483

CDATA node type, 309
charms
  in calculator example, 428–429
  in Windows Store apps, 374
CharSet field, 447, 448
Char type, 24, 271
Char value type, 145
CheckBox control, 384
CheckCharacters property, 310
CheckingAccount class, 15
ChildNodes property, 323, 326
Circle class, 236
Class attribute, 469
classes
  abstract classes, 130–131
  base classes, 126–129
  class-wide members
    data members, 88–89
    member functions, 90–91
    overview, 87–88
    static constructors, 92–93
  concrete classes, 130–131
  constants in
    class-wide constants, 93–94
    instance constants, 94–95
    overview, 93
  constructors
    defining, 84–86
    member initialization lists, 86–87
  creating objects, 83–84
  for custom attributes, 463–464
  derived classes, 129–130
  and finalizers, 106
  in header files, 79–80
  in source files, 81–82
  in object-oriented programming, 16
  object relationships
    creating LoyaltyScheme class, 95–96
    creating LoyaltyScheme objects, 97–100
    implementing LoyaltyScheme class, 96–97
    overview, 95–96
    testing example application, 100
  organizing, 78–79
  overriding member functions, 131–136
  for predefined attributes, 458–461
  protected access, 136–137
  sealed classes
    and abstract classes, 137
    overview, 137
  vs. structures, 149–150
  in Windows RT, 391–392
class keyword, 20
class library, .NET, 265
class members, 77
class-wide constants, 93–94
class-wide members
  data members, 88–89
  member functions, 90–91
  overview, 87–88
  static constructors, 92–93
ClearButton_Click method, 402
Clear method, 212
clearOnNextKey variable, 409
Clone method, 212, 324, 326
CloneNode method, 324, 326
Close method, 283, 287, 298, 299, 308, 318
CLR (Common Language Runtime), 20, 263–264, 336, 437, 454
CLS (Common Language Specification), 160, 265, 298
CLSCompliantAttribute class, 458
CLS-compliant operators, overloading, 166–167
code-behind files, 379, 382
code reuse, and inheritance, 122
collections
  List<T> class, overview, 219–221
  overview, 219
  SortedList<K,V> class, overview, 222–223
Collections interfaces, 273–274
Collections namespaces, 272–273
ColumnDefinition element, 386
Column property, 387
Combine method, 250
ComboBox control, 384
COM Callable Wrappers (CCWs), 483
COM (Component Object Model), 276
  overview, 475–476
  using from .NET code
    and RCWs, 476–477
    creating RCWs, 477–480
    handling errors, 480–481
    late binding to COM objects, 481–482
    overview, 476
  using .NET components as COM components, 483–485
CommandText property, 340
Comment node type, 309

Common Language Runtime (CLR), 20, 263–264, 336, 437, 454
Common Language Specification (CLS), 160, 298
Compare method, 222
CompareTo method, 218, 222
compiling source files, 9–10
complement operator (~), 32
Component Object Model (COM), 276. *See* COM
concrete classes, 130–131
ConditionalAttribute class, 458
ConfigurationManager class, 339
ConformanceLevel property, 310
connected application, ADO.NET
  connecting to database, 337–341
  creating and executing command, 340–341
  executing command that modifies data, 341–342
  executing queries and processing results, 342–343
  overview, 336–337
Connection property, 340
ConnectionStringSettings object, 339
connectionStrings section, 338
connectivity, WCF, 353
Console line, 4
Console::ReadLine function, 44
Console::Write function, 44
constants
  in classes
    class-wide constants, 93–94
    instance constants, 94–95
    overview, 93
  overview, 28–29
const_cast<> operator, 33
constructors
  defining, 84–86
  handling exceptions for, 184–185
  member initialization lists, 86–87
  for structures, 150
ContainsKey method, 223
Contains method, 241
ContainsValue method, 223
Content attribute, 376
content controls, 382
continue keyword, 73
contracts
  in calculator example, 428–429
  WCF, 356–358
  in Windows Store apps, 374
controls, in XAML, 382–383
Control templates, 381
conversion operator, 164
ConverterClass, 479
converting constructors, 164
ConvertOutputString function, 423
ConvertTextToInt function, 407, 421
Convert::ToInt32 function, 44
copy constructors, overview, 113–116
Copy method, 212, 215, 292
CopyTo method, 212, 293
Count property, 220
count variable, 289
CreateAttribute method, 324
CreateCDataSection method, 324
CreateComment method, 324
CreateDefaultAttribute method, 324
CreateDirectory method, 290
CreateDocumentType method, 324
CreateElement method, 324
CreateEntityReference method, 324
Create method, 291–293, 306, 308
CreateNavigator method, 324, 326
CreateNode method, 324
CreateProcessingInstruction method, 324
CreateSubdirectory method, 291
CreateText method, 292, 293
CreateTextNode method, 324
CreateWhitespace method, 324
CreateXmlDeclaration method, 324
CreationTime property, 291, 293
CreditCardAccount class, 78
CTS (Common Type System), 264
Cube function, 249
CurrentAccount class, 123, 126–127, 129–130
CurrentAccount.cpp project, 128
CurrentAccount header file, 129
Current property, 210, 218
custom attributes
  creating, 463–467
  design criteria for attribute classes, 463–464
  obtaining data using reflection, 470–472
  overview, 461–463
  properties for, 463–464

D

data adapter, 344
DataColumn class, 344
DataContract class, 357
data contracts, 356

data hiding, 14
data members, class-wide, 88–89
Data namespaces, 276–277
DataPackage class, in calculator example, 430
data providers, ADO.NET, 334–335
DataRow class, 344
DataSet class, disconnected operation using, 344–350
DataTransferManager, 431
data types, for variables, 23–24
Date structure, 150, 152
DateTime class, 234
DbConnection class, 336
DbDataAdapter class, 344
DbProviderFactory class, 346
DCOM (Distributed Component Object Model), 352
DebuggableAttribute class, 458
DebuggerHiddenAttribute class, 458
DebuggerStepThroughAttribute class, 458
debugging, stepping through application, 47–51
Debug toolbar, 49
declarative UI layout, 381
declaring variables
  multiple, 26
  overview, 25
decrement operators, overloading, 171–172
DefaultAttribute, 395
default branch, 66
default values, for function prototypes, 40
delegate keyword, 247, 250
delegates
  defining, 247
  implementing
    calling non-static member functions by using delegates, 249
    calling static member functions by using delegates, 248–249
    delegates that return result, 252–253
    overview, 247
    using multicast delegates, 249–252
  overview, 245–246
  purpose of, 246–247
DeleteCommand, 345
delete method
  for arrays, 204
  overview, 109
Delete method, 290, 291, 292, 293
Depth property, 307
deque type, STL/CLR, 226
derived classes, 129–130
destructors
  overview, 105–106
  using, 109–110
Diagnostics namespace, 274
Dialog class, 265
Dictionary<K,V> class, 219
directories, getting information about, 290–297
Directory class, 274, 282
DirectoryInfo class, 274, 282, 290–291
DirectoryName property, 293
Directory property, 293
DisplayDate function, 59
Dispose method, 283, 287, 298–299, 308
Distributed Component Object Model (DCOM), 352
distributed systems, WCF, 352
DivideByZeroException, 183
division operator (/), 30
DLL (Dynamic-Link Library), 192, 365–368, 444
DllImport attribute, 446
DllImportAttribute class, 447–448
DOB member, 151–152
Documentation attribute, 466
DocumentationAttribute class, 464, 472
DocumentElement property, 323
DocumentFragment node type, 309
Document node type, 309
Document Object Model (DOM), 307
DocumentType node type, 309
DocumentType property, 323
do keyword, 72
DOM (Document Object Model), 307
dot operator (.), 20
double-colon syntax (::), 79
Double type, 24, 271
Double value type, 144
do-while loops, overview, 71–73
DtdProcessing property, 310
duplex operation, 358
dynamic allocation, of arrays, 203–205
dynamic_cast, 170, 444
dynamic_cast<> operator, 33
dynamic invocation, 467
Dynamic-Link Library (DLL, 192, 444

E

for each loop
  using with arrays, 210–211
EarnPointsOnAmount function, 97

EF (Entity Framework), 276
E_INVALIDARG error, 480
Element node type, 309
elements in arrays, copying, 215
EnableBinaryButtons method, 420
EnableDecimalButtons method, 420
EnableHexButtons method, 420
encapsulation, in object-oriented programming, 14–15
Encoding property, 307
EndElement node type, 309
EndEntity node type, 309
EndPointAddress class, 362
endpoints, WCF, 353–354
EntityClient data provider, 334
Entity Framework (EF), 276
Entity node type, 309
EntityReference node type, 309
EntryPoint field, 447–448
EnumerateDirectories method, 290–291
EnumerateFiles method, 290–291
EnumerateFileSystemEntries method, 290–291
enumerations
  creating, 153–154
  memory usage, 156
  using in programs, 156
enumerators, using with arrays, 218–219
EOF property, 307
E_POINTER eror, 480
EqualsButton_Click method, 407, 425
Equals function, overloading, 169–171
equal sign (=), 236
Equals method, 74, 471
errNo field, 190
error handling, using COM components from .NET, 480–481
Error List window, 10
errors, in properties, 232
EventArgs object, 260
event handling, in XAML, 389
event keyword, 255
events
  event receiver, 256–258
  event source class, 254–256
  overview, 253–254
  quick reference, 262
  standard, 259–261
  System::EventHandler delegate and, 259–261
EvtRcv class, 257
EvtSrc class, 255
ExactSpelling field, 448
exceptions
  and safe_cast keyword, 191–192
  creating, 189–191
  Exception class properties, 182–183
  handling
    catch block, 189
    Exception class properties, 182–183
    exception hierarchy, 184
    finally block, 188
    try/catch blocks, 180–182
    with constructors, 184–185
  in mixed-language programming, 192–195
  nesting, 185–188
  overview, 175–178
  rethrowing, 185–188
  throwing, 178–180
  types of, 178
executable programs
  compiling source files, 6, 9–10
  creating project, 8–9
  running program, 7, 11
  source files for, 9
ExecuteNonQuery method, 337, 341
ExecuteReader method, 337, 342
ExecuteScalar method, 337, 340
Exists method, 212, 288, 290, 292
Exists property, 291, 293
explicit layout, 449
eXtensible Markup Language. *See* XML
Extensible Stylesheet Language Transformations (XSLT), 306
Extensible Stylesheet Language (XSL), 276
Extension property, 291

F

fall-through, using in switch statement, 67–68
fault contracts, 356
FieldOffsetAttribute class, 449
FIFO (first in, first out), 226
FileAccess enumeration, 286
FileAttributes class, 296
File class, 274, 282, 288
FileInfo class, 274, 282
FileMode enumeration, 286
File Picker contract, Windows 8, 429
files. *See also* binary I/O; *See also* text I/O
  getting information about, 290–297
  quick reference, 303–304

FileShare enumeration, 286
FileStream class, 274, 282, 286–287
file structure, for Windows Store apps, 379–380
FileSystemInfo class, 274, 282
FileSystemWatcher class, 274, 282
FillBuffer method, 299
finalAmount variable, 50
finalizers
  overview, 106
  using, 108–109
finally block, 347
  handling exceptions using, 188
FindAll method, 212
FindLast method, 212
Find method, 212
FirstChild property, 323, 326
FirstEventHandler delegate, 255
first in, first out (FIFO), 226
FlagsAttribute, 395
FlagsAttribute class, 458
FlipView control, 384
floating-point types, 272
floating-point values, 169
float type, 24
flow control statements
  if statement
    multiway tests, 62–64
    nested tests, 64–65
    one-way tests, 57–61
    overview, 57
    two-way tests, 61–62
  loop statements
    do-while loops, 71–73
    for loops, 70–71
    overview, 68
    unconditional jumps in, 73–74
    while loops, 68–70
  switch statement
    overview, 65–67
    using fall-through in, 67–68
FlushAsync method, 283
Flush method, 283, 298, 318
FontSize property, 399
for-each loop, 68
ForEach method, 212
Foreground property, 403
for loops, overview, 70–71
Format member, 154
Formatting property, 318, 320
forms, 370
FromBinary function, 422
FullName property, 291, 293, 468
fully qualified name, 269
func function, 180
function header, 41
Function keyword, 38
functions
  calling, 45–47
  function bodies
    defining, 41–42
    overview, 41
    parameters in, 42–43
    return type, 43–45
  function prototypes
    declaring, 38–39
    default values for, 40
    defined, 38
    parameters in, 39
    return type, 39–40
  global scope, 51–53
  local scope, 51–53
  non-static member functions, calling by using delegates, 249
  overloading, 53–55
  overriding, 131–136
  passing arrays to, 200–202
  static member functions, calling by using delegates, 248–249

G

GAC (Global Assembly Cache), 484
garbage collector, 103–104
GCHandle::Alloc method, 438
GCHandle type, and unmanaged code, 438–441
gcnew operator, 27, 28, 110, 143, 147, 208
gcroot variable, 440
GDI32.dll, 445
generations, 104
generic keyword, 206, 392
generics, in Windows RT, 392
generic types
  and templates
    overview, 224
    STL/CLR library, 224–227
  overview, 205–206
Geometry.cpp file, 117
GetAccountNumber function, 82
GetAttribute method, 308
GetAttributes method, 292, 296

GetConstructor method, 468
GetConstructors method, 468
GetCreationTime method, 290, 292
GetCurrentDirectory method, 290
GetCustomAttribute method, 470
GetCustomAttributes method, 470, 471
get_date function, 229
GetDay function, 59
GetDirectories function, 297
GetDirectories method, 290, 291
GetDirectoryRoot method, 290
GetElementById method, 324
GetElementsByTagName method, 324
GetEnumerator method, 212, 218, 324, 326
GetEvent method, 468
GetEvents method, 468
GetField method, 468
GetFields method, 468
GetFiles method, 290, 291
GetFileSystemEntries method, 290
GetFileSystemInfos method, 291
get function, 161
GetHashCode method, 171, 471
GetInterestRate function, 87
GetInterfaceMap method, 468
GetInterface method, 468
GetInterfaces method, 468
GetInvocationList function, 253
GetLastAccessTime method, 290, 292
GetLastWriteTime method, 290, 292
GetLength method, 212, 214
GetLogicalDrives method, 290
GetLowerBound method, 212, 214
GetMember method, 468
GetMembers method, 468
get method, 234
GetMethod method, 468
GetMethods method, 468
GetMonth function, 59
GetNamespaceOfPrefix method, 326
GetNumberOfAccounts function, 90
GetParent method, 290
GetPrefixOfNamespace method, 326
GetProperties method, 469
GetProperty method, 469
GetSystemPowerStatus function, 449, 450
getter, 231
GetTypeFromCLSID method, 481
GetType method, 469, 470, 471
GetUpperBound method, 213, 214
getVal function, 162, 164
getValue function, 440
GetValue method, 213
GetYear function, 58
Global Assembly Cache (GAC), 484
global scope, overview, 51–53
global variables, 52
green and blue stacks, 372–373
Grid control, 375–376
GridView control, 384

H

handles
  to objects, 118–119
  overview, 27–28
handling exceptions
  catch block, 189
  with constructors, 184–185
  Exception class properties, 182–183
  exception hierarchy, 184
  finally block, 188
  try/catch blocks, 180–182
hardware, and Windows Store apps, 374
HasAttributes property, 307
HasChildNodes property, 323, 326
hashcode, 171
HashSet<T> class, 219, 273
HasSecurity attribute, 469
HasValue property, 307
header files, classes in, 79–80
Hello World example, 4
hierarchy
  for exceptions, 184
  for inheritance, 123–124
HttpGetEnabled property, 364
HttpRequest class, 277
HttpResponse class, 277
HTTP transport, 354

I

IChannel handle, 362
ICollection<T> interface, 273
IComparable interface, 218
IComparer<T> interface, 273
IConnectionPointContainer interface, 477
IConnectionPoint interface, 477
IDE (integrated development environment), 11

identifiers, overview, 5–6
IDictionary<K,V> interface, 273
IDispatchEx interface, 477
IDispatch interface, 477
IEnumerable<T> interface, 273
IEnumerator interface, 210
IEnumerator<T> interface, 273
IEnumVARIANT interface, 477
IErrorInfo interface, 477
if statement
  multiway tests, 62–64
  nested tests, 64–65
  one-way tests, 57–61
  overview, 57
  two-way tests, 61–62
IgnoreComments property, 310
IgnoreProcessingInstructions property, 310
IgnoreWhitespace property, 310
ILDASM, 454–457
ILDASM tool, 264
IL Disassembler tool, 264
IL (Intermediate Language), 375
IList<T> interface, 273
IMathService contract, 361
IMetadataExchange contract, 363
Import attribute, 469
ImportNode method, 324
#include directive, 439
include guard, 360
#include statements, 79, 96
increment operators, overloading, 171–172
Indentation property, 318
IndentChar property, 318
indexed properties
  bank example
    creating Account class properties, 239–240
    implementing to retrieve accounts, 241–244
  defined, 230
  overview, 236
indexing, 207
IndexOfKey method, 223
IndexOf method, 213, 216
IndexOfValue method, 223
inheritance
  abstract classes, 130–131
  and code reuse, 122
  base classes, 126–129
  concrete classes, 130–131
  derived classes, 129–130
  designing hierarchy for, 123–124
  interfaces, 138–139
  in object-oriented programming, 15
  overriding member functions, 131–136
  overview, 121–122
  properties and, 235
  protected access, 136–137
  sealed classes
    and abstract classes, 137
    overview, 137
  substitutability, 123–124
  terminology, 122
InitializeComponent method, 406
Initialize method, 213
inline functions, 19
InnerText property, 323, 326
InnerXml property, 323, 326
input/output. *See* I/O
input variable, 44
InsertAfter method, 324, 326
InsertBefore method, 324, 326
InsertCommand, 345
Insert function, 221
instance constants, 94–95
instance members, 77
Int16 type, 271
Int16 value type, 144
Int32 type, 271
Int64 type, 271
Int64 value type, 144
integrated development environment (IDE), 11
Interface attribute, 469
interfaces, properties in, 235
interior pointers, 441
Intermediate Language (IL), 375
inter-process communication (IPC), 353
IntPtr type, 271
IntPtr value type, 145
IntPtr::Zero argument, 447
introspection, 467
int type, 18, 24
IntVal class, 161, 163
InvalidCastException, 191
InvalidOperationException, 219
invocation list, 250
InvokeMember method, 469, 482
Invoke method, 248
IOException class, 274, 282

I/O (input/output)
  binary I/O, 298
    BinaryReader class, 299–303
    BinaryWriter class, 298
  text I/O
    FileStream class, 286–287
    overview, 283
    TextReader, 287–290
    TextWriter, 283–285
IO namespace, 274
IPC (inter-process communication), 353
IPC transport, 354
IProvideClassInfo interface, 477
IReadOnlyCollection<T> interface, 273
IsAbstract property, 468
IsArray property, 468
IsByRef property, 468
IsClass property, 468
IsEmptyElement property, 307
ISet<T> interface, 273
IsFixedSize property, 212
IsInterface property, 468
IsNotPublic property, 468
IsPublic property, 468
IsReadOnly property, 212, 323, 326
IsStartElement method, 308
IsSynchronized property, 212
IsValueType property, 468
Item property, 307, 323, 326
items controls, 382
iterator, 225, 226
IUnknown interface, 477

J

JIT (Just-In-Time) compiler, 264
Just-In-Time (JIT) compiler, 264

K

Kernel32.dll, 445
KeyRoutedEventArgs, 389
KeyValuePair class, 223
keywords, overview, 5–6

L

LastAccessTime property, 291, 293
LastChild property, 323, 326
LastIndexOf method, 213, 216
last in, first out (LIFO), 226
LastWriteTime property, 291, 293
late binding, to COM objects, 481–482
layout
  in calculator example, 398–401
  in XAML, 384–388
left-shift operator (<<), 32
Length property, 212, 213, 293
less-than-or-equal-to (<=) condition, 199
lifetimes, of objects, 103–105
LIFO (last in, first out), 226
LimitReached event, 260
LineNumberOffset property, 310
LinePositionOffset property, 310
LinkedList<T> class, 219, 273
Linq class, 276
ListBox control, 384
ListBoxItems control, 381
List class, 240, 241
List<T> class, 273
  overview, 219–221
list type, STL/CLR, 226
ListView control, 384
literal constant, 28
literal keyword, 93
Live tiles, 415
Load method, 325
LoadXml method, 325
LocalName property, 307, 323, 326
local scope, overview, 51–53
location element, 331
LogAttribute class, 462
logical operators
  overloading, 167–169
  overview, 31–32
LongLength property, 212
long long type, 24
long type, 24
LookupNamespace method, 308
LookupPrefix method, 318
loop statements
  do-while loops, 71–73
  for loops, 70–71
  overview, 68
  unconditional jumps in, 73–74
  while loops, 68–70
LoyaltyScheme class example
  creating, 95–96
  creating objects, 97–100

implementing class, 96–97
testing application, 100–101

M

main function, 41, 248
overview, 4–5
main method, 108
MainPage class, 406
MainPage.g.cpp, 380
MainPage.g.h, 380
MakePurchase function, 80, 98
MakeRepayment function, 80
managed arrays
and reference types, 208–209
initializing, 208
multidimensional, 211
overview, 207
using for each loop with, 210–211
managed code
vs. unmanaged code
GCHandle type, 438–441
mixed classes, 437–438
overview, 437
Map type, 394
map type, STL/CLR, 226
MapView type, 394
Margin property, 376
markup extensions, 382
MarshalAsAttribute class, 458
marshaling, 356
Math::Abs function, 169
MathServiceClient class, 368
MaxCharactersInDocument property, 310
MaximumRowsOrColums attribute, 387
MBCS (Multi-Byte Character Set), 405
MClass object, 440
mc variable, 440
member functions, class-wide, 90–91
member initialization lists, in constructors, 86–87
MemoryStream class, 274, 282
memory usage, for enumerations, 156
MEPs (message exchange patterns), 357–358
MessageBox function, 445, 446
Message property, Exception class, 182
metadata
adding to WCF services, 363–365
and attributes
overview, 453–454
using ILDASM, 454–457
.NET, 266–268
in Windows RT, 390
MEX (Metadata Exchange) addresses, 355
MFC (Microsoft Foundation Classes), 370, 405
Microsoft Intermediate Language (MSIL), 229, 375
Microsoft Intermediate Language (MSIL) file, 81
Microsoft-specific data types, 24
mixed classes, and unmanaged code, 437–438
mixed-language programming, exceptions in, 192–195
mm class, 115
Module property, 468
modulus operator (%), 30
Move method, 290, 292
MoveNext method, 210
MoveToAttribute method, 308
MoveToContentAsync method, 308
MoveToContent method, 308
MoveToElement method, 308, 315
MoveToFirstAttribute method, 308
MoveTo method, 291, 293
MoveToNextAttribute method, 308, 315
MSIL (Microsoft Intermediate Language), 229, 264, 375
MSIL (Microsoft Intermediate Language) file, 81
MSMQ transport, 354
Multi-Byte Character Set (MBCS), 405
multicast delegates, 249–252
multidimensional arrays
managed arrays, 211
native arrays, 202–203
multimap type, STL/CLR, 226
multiplication operator (*), 30
multiset type, STL/CLR, 226

N

named parameters, 463
named pipes, 353
Name property, 291, 293, 307, 323, 326
Namespace property, 468
namespaces
ADO.NET, 335
.NET
Collections interfaces, 273–274
Collections namespaces, 272–273
Data namespaces, 276–277
Diagnostics namespace, 274
IO namespace, 274
Net namespaces, 275

overview, 268–269
ServiceModel namespaces, 275
System namespace, 270–273
using in C++ applications, 270–271
Web namespaces, 277–278
Windows namespaces, 275
Xml namespaces, 276
Namespaces property, 318
NamespaceURI property, 307, 323
naming, of variables, 25–26
NaN (not a number), 272
native arrays
dynamic allocation of, 203–205
initializing, 202
multidimensional, 202–203
overview, 197–199
passing to functions, 200–202
negative infinity, 272
nesting
exceptions, 185–188
if statements, 64–65
.NET
using COM components from
and RCWs, 476–477
creating RCWs, 477–480
handling errors, 480–481
late binding to COM objects, 481–482
overview, 476
using .NET components as COM components, 483–485
.NET Framework
assemblies, 266
class library, 265
CLR (Common Language Runtime), 263–264
CLS (Common Language Specification), 265
CTS (Common Type System), 264
metadata, 266–268
MSIL (Microsoft Intermediate Language), 264
namespaces
Collections interfaces, 273–274
Collections namespaces, 272–273
Data namespaces, 276–277
Diagnostics namespace, 274
IO namespace, 274
Net namespaces, 275
overview, 268–269
ServiceModel namespaces, 275
System namespace, 270–273
using in C++ applications, 270–271
Web namespaces, 277–278
Windows namespaces, 275
Xml namespaces, 276
overview, 263
quick reference, 278
XML and
NET XML namespaces, 306
overview, 305–306
XML processing classes, 306–307
NetMsmqBinding, 355, 358
NetNamedPipeBinding, 355, 358
Net namespaces, 275
NetTcpBinding, 355, 358
new operator, 203
NextSibling property, 323, 326
NodeChanged event, 325
NodeChanging event, 325
NodeInserted event, 325
NodeInserting event, 325
nodelist, 329
NodeRemoved event, 325
NodeRemoving event, 325
NodeType property, 308–309, 324, 326
None node type, 309
NonSerializedAttribute class, 458
non-static member functions, calling by using delegates, 249
Normalize method, 326
normal pointers, 246
not a number (NaN), 272
Notation node type, 309
NotifyDelegate, 250
NOT operator (!), 31
NotPublic attribute, 469
nullptr keyword, 184
nullptr value, 98
number bases
in calculator example
adding buttons for, 417–418
changing base, 418–421
converting string in display, 421–425
NumericOp function, 247

O

Object Linking and Embedding (OLE), 370
object-oriented programming
advantages of, 16–17
classes in, 16
defined, 13–14

encapsulation in, 14–15
example of, 17–22
inheritance in, 15
objects in, 16
polymorphism in, 15–16
objects
and stack semantics
creating objects with, 111–113
overview, 116–118
copy constructors, 113–116
creating, 83–84
destructors
overview, 105–106
using, 109–110
finalizers
overview, 106
using, 108–109
handles to, 118–119
lifetimes of, 103–105
in object-oriented programming, 16
relationships for
creating LoyaltyScheme class, 95–96
creating LoyaltyScheme objects, 97–100
implementing LoyaltyScheme class, 96–97
overview, 95–96
testing example application, 100–101
traditional C++ creation and destruction, 110–111
obj pointer, 444
Observer class, 261
Obsolete attribute, 460
ObsoleteAttribute class, 458
ODBC data provider, 334
OleDb data provider, 334
OLE (Object Linking and Embedding), 370
one-way messaging, 358
OnNavigatedFrom function, 432
OnNavigatedTo function, 421, 432
op_Addition operator, 166
op_AddressOf operator, 166
op_BitwiseAnd operator, 166
op_BitwiseOr operator, 166
op_Comma operator, 166
op_Decrement operator, 166
op_Division operator, 166
Open method, 292–293
OpenRead method, 292–293
OpenText method, 292–293
OpenWrite method, 292–293
op_Equality operator, 166
operation contracts, 356
operator overloading
and reference types, 172–173
arithmetic operators, 161–162
best practices, 173–174
CLS-compliant operators, 166–167
decrement operators, 171–172
increment operators, 171–172
logical operators
Equals function, 169–170
overview, 167–169
overview, 159
restrictions on, 160
rules for, 161
static operator overloads, 163–166
types needing, 160
operators
arithmetic operators, 30–31
assignment operators, 30
bitwise operators, 32–33
cast operator, 33–34
defined, 30
logical operators, 31–32
precedence of, 34
relational operators, 31–32
ternary operator, 32–33
op_ExclusiveOr operator, 166
op_GreaterThan operator, 167
op_GreaterThanOrEqual operator, 167
op_Increment operator, 167
op_Inequality operator, 167
op_LeftShift operator, 167
op_LessThan operator, 167
op_LessThanOrEqual operator, 167
op_LogicalAnd operator, 167
op_LogicalNot operator, 166
op_LogicalOr operator, 167
op_Modulus operator, 167
op_Multiply operator, 167
op_OnesComplement operator, 166
op_PointerDereference operator, 166
op_RightShift operator, 167
op_Subtraction operator, 167
op_UnaryNegation operator, 166
op_UnaryPlus operator, 166
OracleClient data provider, 334
Orientation property, 384
OR operator, 31–32
OuterXml property, 324, 326
overloaded [ ] operator, 230

overloading functions, 53–55
overriding member functions, 131–136
OwnerDocument property, 324, 326

P

Package.appxmanifest file, 379, 412
Page class, 377
Page element, 375
ParamArrayAttribute class, 458
parameters
  in function bodies, 42–43
  in function prototypes, 39
  names for, 39
ParentNode property, 324, 326
Parent property, 291
parsing XML using, XmlReaderSettings class, 310–314
partial classes, 391
partial keyword, 392
passing structured data, 449–452
Path class, 274, 282
PeekChar method, 299
Peek method, 287
pf function pointer, 246
pinning pointers, 441–442
P/Invoke (Platform Invoke)
  calling functions in Win32 API
    DllImportAttribute class, 447–448
    overview, 444–447
    passing structured data, 449–452
Platform::Collections namespace, 394
Platform Invoke (P/Invoke). *See* P/Invoke
Platform::Metadata namespace, 394
Platform namespaces, in Windows RT, 394
Play To contract, Windows 8, 429
PNG (Portable Network Graphics) files, 379, 414
pointer operator (->), 28
pointers
  interior pointers, 441
  overview, 27–28
  pinning pointers, 441–442
polymorphism
  in object-oriented programming, 15–16
pop_back function, 225
Portable Network Graphics (PNG) files, 379, 414
positional parameters, 463
positive infinity, 272
post-decrement, 171
postfix increment operator expression, 31
post-increment, 171
#pragma once directive, 125
precedence, of operators, 34
precompiled headers, 379
pre-decrement, 171
predefined attributes
  AssemblyInfo.cpp file, 457–458
  classes for, 458–461
  obtaining attribute data using, 469
  overview, 457
prefix increment operator expression, 31
Prefix property, 308, 324, 326
pre-increment, 171
PrependChild method, 326
PreserveSig field, 448
PreserveWhitespace property, 324
PreviousSibling property, 323, 326
printArea function, 236
PrintStatement function, 78
private auto ansi, 456
private class, 136
private keyword, 19
ProcessChildNodes function, 330
processFile function, 297
ProcessingInstruction node type, 309
projections, 390
Project Properties dialog box, 338
projects, creating, 8–9
properties
  for custom attributes, 463–464
  for Exception class, 182–183
  indexed
    bank example, 236–244
    overview, 236
  overview, 229–230
  quick reference, 244
  scalar properties
    auto-implemented properties, 233
    errors in properties, 232
    inheritance and, 235
    in interfaces, 235–236
    overview, 231–232
    read-only and write-only properties, 233–235
  of value types, 145
Properties property, 430
Properties tab, 402
property keyword, 231
protected access, 136–137
ProviderName property, 339

proxy, accessing WCF services using, 365–368
Public attribute, 469
public class, 136
public keyword, 18
push_back function, 225

Q

Queue<T> class, 219, 273
queue type, STL/CLR, 226
QuoteChar property, 318

R

RaiseOne function, 259
raise_OnFirstEvent method, 256
RaiseTwo function, 259
Rank property, 212, 213
rateFraction variable, 50
RC (Release Candidate) version, 7
RCWs (Runtime Callable Wrappers)
  creating, 477–480
  overview, 476–477
Read7BitEncodedInt method, 299
ReadAsync method, 287, 308
ReadAttributeValue method, 308
ReadBlockAsync method, 287
ReadBlock method, 287
ReadBoolean method, 299
ReadByte method, 299
ReadBytes method, 299
ReadChar method, 299
ReadChars method, 299
ReadContentAsAsync method, 308
ReadContentAsInt method, 308
ReadContentAs method, 308
ReadContentAsString method, 308
ReadDecimal method, 299
ReadDouble method, 299
ReadElementContentAsInt method, 308
ReadElementContentAs method, 308
ReadElementString method, 308
ReadEndElement method, 309
Read function, 313
ReadInnerXml method, 308
ReadInt16 method, 299
ReadInt32 method, 299
ReadInt64 method, 299
ReadLine method, 287, 289
Read method, 287, 299, 308
ReadNode method, 325
read-only properties, 233–234, 239
ReadOuterXml method, 308
ReadSByte method, 299
ReadSingle method, 299
ReadStartElement method, 309
ReadState property, 308
ReadString method, 299, 309
ReadToDescendant method, 309
ReadToEndAsync method, 287
ReadToEnd method, 287
ReadToFollowing method, 309
ReadToNextSibling method, 309
ReadUInt16 method, 299
ReadUInt32 method, 299
ReadUInt64 method, 299
RedeemPoints function, 97
refactoring, 407
reference counted objects, 391
Reference Manager dialog box, 193
reference types, 20
  and managed arrays, 208–209
  and operator overloading, 172–173
ref keyword, 184
reflection
  obtaining attribute data using
    accessing custom attribute data, 470–472
    accessing standard attributes, 469
    overview, 467–468
    Type class, 467–469
ref new keyword, 391
Refresh method, 293
reinterpret_cast<> operator, 34
relational operators, overview, 31–32
relationships, object
  LoyaltyScheme class example
    creating, 95–96
    creating objects, 97–100
    implementing class, 96–97
    testing application, 100–101
  overview, 95–96
Release Candidate (RC) version, 7
remembering operations, in calculator example, 406
Remote Method Invocation (RMI), 352
remoting, 264
RemoveAccount function, 241
RemoveAll method, 325, 326
RemoveByIndex method, 223
RemoveChild method, 325–326

Remove function, 221
RemoveHandler function, 259
Remove method
  in bank example, 240–241
  manipulating invocation lists using, 250
remove_OnFirstEvent method, 256
RemoveRange function, 221
ReplaceChild method, 325, 327
Replace method, 292, 293
reserved words, 5
Reset function, 409
Reset method, 210
Resize method, 213
resource dictionaries, 381
restrictions, on operator overloading, 160
rethrowing exceptions, 185–188
return keyword, 4, 41
return type
  for function bodies, 43–45
  for function prototypes, 39–40
Reverse method, 213
rightOperand variable, 408
right-shift operator (>>), 32
RMI (Remote Method Invocation), 352
Root property, 291
RoutedEventArgs class, 389
RowDefinition element, 386
running programs, 7, 11
Runtime Callable Wrappers (RCWs). *See* RCWs
RuntimeWrappedException, 192

S

safe_cast, 444
safe_cast keyword, and exceptions, 191–192
safe_cast<> operator, 33
Save method, 325
SavingsAccount class, 15, 123, 126–127, 130
SavingsAccount.cpp project, 129
SavingsAccount header file, 129
SAX (Simple API for XML) API, 307
SayHello method, 456
SByte type, 271
SByte value type, 144
scalar properties
  auto-implemented, 233
  defined, 230
  errors in, 232
  inheritance and, 235
  in interfaces, 235
  read-only and write-only, 233–234
Schema class, 276
SchemaInfo property, 308
Schemas property, 310, 324
scope
  global scope, 51–53
  local scope, 51–53
scope resolution operator (::), 269
ScrollViewerl control, 384
Sealed attribute, 469
sealed classes
  and abstract classes, 137
  overview, 137
Search contract, Windows 8, 428
searching, arrays, 216–217
SecondEventHandler delegate, 257
security permissions, 266
Seek method, 298, 302
SeekOrigin enumeration, 302
seek pointer, 302
SEH (Structured Exception Handling), 178
SelectCommand, 345
SelectCommand property, 347
Selected Components pane, 193
SelectNodes method, 325, 327
SelectSingleNode method, 325, 327
sequential layout, 449
Serializable attribute, 469
SerializableAttribute class, 458
Serialization class, 276
ServiceContract attribute, 356
service contracts, 356
ServiceModel namespaces, 275
services
  WCF, 352
    accessing by using proxy, 365–368
    adding metadata to, 363–365
    overview, 359–362
    writing service client, 361–362
SetAttributes method, 292
SetBitmap method, 430
SetCreationTime method, 290, 292
SetCreditLimit function, 80
SetCurrentDirectory method, 291
SetCursorToArrow line, 112
SetData method, 430
SetDataProvider method, 430
set_date function, 229
SetHtmlFormat method, 430

SetInterestRate function, 87
SetLastAccessTime method, 291–292
SetLastError field, 448
SetLastWriteTime method, 291–292
SetName function, 20
SetRtf method, 430
SetStorageItems method, 431
setter, 231
SetText method, 431
Settings contract, Windows 8, 429
Settings property, 308, 318
set type, STL/CLR, 226
SetUri method, 431
SetValue method, 213, 214
Shape class, 235
Share contract, Windows 8, 429
sharing
  in calculator example
    contracts and charms, 428–429
    DataPackage class, 430
    handling requests, 431–432
    implementing, 429–430
    overview, 428
short-circuit evaluation, 32, 60
short type, 24
SignificantWhitespace node type, 309
Simple API for XML (SAX) API, 307
Simple Mail Transfer Protocol (SMTP), 277
Simple Object Access Protocol (SOAP), 277, 306
simplex messaging, 358
single-byte string type, 405
Single type, 271
Single value type, 144
sizeof operator, 201
SkipAsync method, 309
Skip method, 309
SMTP (Simple Mail Transfer Protocol), 277
SOAP (Simple Object Access Protocol), 277, 306
SortedList<K,V> class, 273
  overview, 219, 222–223
SortedSet<T> class, 273
sorting, arrays, 217–218
Sort method, 213
Source property, Exception class, 182
SqlClient data provider, 334
SqlServerCe data provider, 334
square brackets [ ], 267
square function, 248
StackPanel control, 384
stack semantics, and objects
  creating objects with, 111–113
  overview, 116–118
Stack<T> class, 219, 273
StackTrace property, Exception class, 182
stack type, STL/CLR, 226
StandardStyles.xaml file, 380, 426
Start Without Debugging option, 47, 401
static_cast<double> operator, 33
static_cast<> operator, 33
static constructors, class-wide, 92–93
static keyword, 90
static member functions, calling by using delegates, 248–249
static operator overloads, overloading, 163–166
stdafx.h file, 107
stepping through application, 47–51
STL/CLR library
  concepts behind, 225–227
  overview, 224–225
Stop Debugging option, Debug menu, 165
Stream class, 274, 282, 284, 302
StreamReader class, 274, 282
streams, 302
StreamWriter class, 274, 282, 283
String class, 18, 44
  overview, 29–30
String^ parameter, 442
StringReader class, 274, 282
strings, in Windows RT, 392–393
StringWriter class, 274, 282
struct keyword, 146
StructLayoutAttribute class, 449, 458
structured data, passing, 449–452
Structured Exception Handling (SEH), 178
structures
  constructors for, 150
  copying, 152–153
  creating, 146–148
  overview, 146
  using within another, 150–152
  vs. classes, 149–150
styles, 381
Sub keyword, 38
substitutability, 123–124
subtraction operator (-), 30
Supports method, 327
switch statement, 313
  overview, 65–67
  using fall-through in, 67–68

swscanf_s function, 404, 409
symbolic constant, 28
Synchronized method, 283, 287
SyncRoot property, 212
syntax, for XAML, 381–382
System::ApplicationException class, 179
System::ArgumentException class, 179
System::ArithmeticException class, 179
System::Array class
  basic operations using, 213–214
  copying elements, 215
  overview, 212
  searching, 216–217
  sorting, 217–218
  using enumerators with, 218–219
System::Collection namespace, 218
System::Collections::Generic namespace, 219
System::Configuration assembly, 345
System::Data::Common namespace, 335
System::Data::EntityClient namespace, 276, 335
System::Data::Linq namespace, 335
System::Data namespace, 335
System::Data::Odbc namespace, 276, 335
System::Data::OleDb namespace, 276, 335
System::Data::OracleClient namespace, 276, 335
System::Data::Services namespace, 335
System::Data::Spatial namespace, 335
System::Data::SqlClient namespace, 276, 335
System::Data::SqlTypes namespace, 335
System::Delegate class, 247
System::Diagnostics namespace, 458
System::DivideByZeroException error, 177
System::Enum class, 153
System::EventHandler delegate, events and, 259–261
System::Exception class, 177, 179
System::GC::Collect static method, 105
System::IndexOutOfRangeException class, 179
System::InvalidCastException class, 179
System::IO namespace, 281–282
System::MemberAccessException class, 179
System::MulticastDelegate class, 247, 249
System namespace
  basic types, 271
  floating-point types, 272
  overview, 270–271
System::NotSupportedException class, 179
System::NullReferenceException class, 179
System::Object class, 169
System::OutOfMemoryException class, 179
SYSTEM_POWER_STATUS structure, 449
System::Reflection namespace, 470
System::Runtime::InteropServices namespace, 437
System::Runtime::Serialization namespace, 303
System::ServiceModel::AddressAccessDenied
    Exception, 361
System::ServiceModel assembly, 362
<system.ServiceModel> element, 367
System::String class, 209
System::SystemException class, 179
System::TypeLoadException class, 179
System::ValueType class, 145, 171
System::Web::Mail namespace, 277
System::Web namespace, 277
System::Web::Security namespace, 277
System::Web::Services namespace, 277
System::Web::UI::HtmlControls namespace, 278
System::Web::UI namespace, 277
System::Xml::Linq namespace, 306
System::Xml namespace, 306
System::Xml::Schema namespace, 306, 316
System::Xml::Serialization namespace, 306
System::Xml::XPath namespace, 306
System::Xml::Xsl namespace, 306

T

TCP/IP transport, 354
TempConverterLib component, 478
TempConverter project, 478
templates, and generic types
  overview, 224
  STL/CLR library
    concepts behind, 225–227
    overview, 224–225
ternary operator, overview, 32–33
TestAtts class, 466, 471
testing, calculator example, 410–412
Test method, 193
text I/O
  FileStream class, 286–287
  overview, 283
  TextReader, 287–290
  TextWriter, 283–286
Text node type, 309
TextReader class, 274, 282, 287–290
TextWriter class, 274, 283–285
throwing exceptions, 178–180
throw keyword, 177
ThrowOnUnmappableCharacter field, 448

tiles, Start Page, calculator example, 412–415
ToBinary function, 423
Tooltip control, 384
ToString function, 134
ToString method, 291, 293, 469, 471
tracking handles, 27
tracking reference, 114
triggers, 381
TrimToSize function, 222
TrimToSize method, 221
TrueForAll method, 213
try block, 347
try/catch blocks, handling exceptions using, 180–182
TryGetValue method, 223
type casting, operator for, 33–34
Type class, obtaining attribute data using, 467–469
typedefs, overview, 29
TypedEventHandler, 432
type-safe, 121

U

UClass object, 440
UInt16 type, 271
UInt16 value type, 144
UInt32 type, 271
UInt32 value type, 144
UInt64 type, 271
UInt64 value type, 144
UIntPtr type, 271
UI (user interface)
    libraries for Windows applications, 372
    model for Windows Store apps, 374
UI (user interface) framework, 275
UlnPtr value type, 145
UML (Unified Modeling Language), 123
unboxing, 443–444
unconditional jumps, in loop statements, 73–74
UnderlyingSystemType property, 468
UnicodeClass attribute, 469
Unified Modeling Language (UML), 123
unmanaged code, 264
    boxing, 443
    interior pointers, 441
    pinning pointers, 441–442
    unboxing, 443–444
    using P/Invoke to call functions in Win32 API
        DllImportAttribute class, 447–448
        overview, 444–447
        passing structured data, 449–452
    vs. managed code
        GCHandle type, 438–440
        mixed classes, 437–438
        overview, 437
UpdateCommand, 345
User32.dll, 445
user interface (UI) framework, 275
#using directive, 270

V

ValHandler class, 317
Validate method, 325
ValidationFlags property, 310
ValidationType property, 310
value keyword, 20, 146, 161
Value property, 308, 324, 326
ValueType property, 308
value types
    and reference types
        overview, 143–144
    enumerations
        creating, 153–154
        memory usage, 156
        using in programs, 156
    properties of, 145
    purpose of, 144–145
    structures
        constructors for, 150
        copying, 152–153
        creating, 146–148
        overview, 146
        using within another, 150–152
        vs. classes, 149–150
variables
    arrays, 28
    assigning values to, 26–27
    constants, 28–29
    data types for, 23–24
    declaring
        multiple, 26
        overview, 25
    defined, 23
    handles, 27–28
    naming of, 25–26
    pointers, 27–28
    String class, 29–30
    typedefs, 29

VariableSizedWrapGrid control, 387
VectorIterator class, 394
Vector type, 394
vector type, STL/CLR, 226
VectorViewIterator class, 394
VectorView type, 394
Vehicle class, 122
versioning, 266
VirtualizingStackPanel control, 385
void keyword, 38

W

W3C DOM, 322
WCF (Windows Communication Foundation)
  addresses, 355
  behaviors, 358–359
  binding, 355
  connectivity, 353
  contracts, 356–358
  defined, 275
  distributed systems, 352
  endpoints, 353–354
  MEPs (message exchange patterns), 357–358
  overview, 351
  services, 352
    accessing by using proxy, 365–368
    adding metadata to, 363–365
    overview, 359–362
    writing service client, 361–362
wchar_t* pointer, 405
wchar_t type, 24
wcstol (Wide Character String To Long) function, 422
Web namespaces, 277–278
web service, 277
Web Service Definition Language (WSDL), 355
Web Service Description Language (WSDL), 306
WeekDay class, 154
while loops, overview, 68–70
white space, 5
Whitespace node type, 309
Wide Character String To Long (wcstol) function, 422
wide string type, 405
Win32 API, 369–370
  calling functions using P/Invoke
    DllImportAttribute class, 447–448
    overview, 444–447
    passing structured data, 449–452
Windows::ApplicationModel.DataTransfer namespace, 430
Windows::ApplicationModel namespaces, 393
Windows applications
  Microsoft Foundation Classes, 370
  Win32 API, 369–370
  Windows Forms, 370–371
  Windows Presentation Foundation, 371
Windows Communication Foundation (WCF), 275
Windows::Data namespaces, 393
Windows::Devices namespaces, 393
Windows Forms, 370–371
Windows::Foundation::Collections namespaces, 393
Windows::Foundation namespaces, 393
Windows::Globalization namespaces, 393
Windows::Graphics namespaces, 393
Windows::Management namespaces, 393
Windows::Media namespaces, 393
WindowsMessageBox function, 448
Windows namespaces, 275
Windows::Networking namespaces, 393
Windows Presentation Foundation, 371
Windows Presentation Foundation (WPF), 275
Windows RT (WinRT). *See* WinRT
Windows Runtime Library (WRL), 390
Windows::Security namespaces, 393
Windows::Storage namespaces, 393
Windows Store apps
  and Windows applications
    Microsoft Foundation Classes, 370
    Win32 API, 369–370
    Windows Forms, 370–371
    Windows Presentation Foundation, 371
  calculator example
    adding tile, 412–415
    app bars, 425–428
    arithmetic buttons, 403–404
    getting number from button, 404–405, 407–408
    handling different number bases, 416–425
    handling number input, 401–402
    laying out number buttons, 398–401
    overview, 397–398
    performing calculations, 408–409
    remembering operations, 406
    sharing in, 428–432
    testing, 410–412
  choosing UI library, 372
  creating, 375–379
  file structure for, 379–380

main features
app behavior, 373
contracts and charms, 374
hardware usage, 374
overview, 373
UI model, 374
WinRT APIs, 374
overview, 372–373
Windows RT
classes, 391–392
generics, 392
metadata, 390
overview, 389–390
Platform namespaces, 394–395
strings, 392–393
Windows namespaces, 393
XAML
controls, 382–383
defined, 380–381
event handling, 389
layout controls, 384–388
syntax, 381–382
Windows::System namespaces, 394
Windows::UI namespaces, 394
Windows::UI::XAML namespaces, 394
Windows::Web namespaces, 394
WinRT (Windows RT)
APIs, 374
classes, 391–392
generics, 392
metadata, 390
overview, 389–390
Platform namespaces, 394
strings, 392–393
Windows namespaces, 393
WPF (Windows Presentation Foundation), 275
Write7BitEncodedInt method, 298
WriteAllLines method, 292
WriteAllText method, 292
WriteAsync method, 283
WriteAttributes method, 318
WriteAttributeString method, 318
WriteBase64 method, 318
WriteBinHex method, 318
WriteCData method, 319
WriteCharEntity method, 319
WriteChars method, 319
WriteComment method, 319
WriteContentTo method, 325, 327
WriteDocType method, 319
WriteElementString method, 319
WriteEndAttribute method, 319
WriteEndDocument method, 319
WriteEndElement method, 319
WriteEntityRef method, 319
WriteFullEndElement method, 319
WriteLineAsync method, 283
WriteLine method, 283
WriteLine statement, 108, 111, 162
Write method, 283, 298
WriteName method, 319
WriteNode method, 319
write-only properties, 233–234
WriteProcessingInstruction method, 319
WriteQualifiedName method, 319
WriteRaw method, 319
WriteStartAttribute method, 319
WriteStartDocument method, 319
WriteStartElement method, 319
Write statement, 108
WriteState property, 318
WriteString method, 319
WriteTo method, 325, 327
WriteValue method, 319
WriteWhitespace method, 319
WriteXml method, 348
WRL (Windows Runtime Library), 390
WSDL (Web Service Definition Language), 355
WSDL (Web Service Description Language), 306
WSDualHttpBinding, 355, 358
WSHttpBinding, 355, 358, 360, 362

X

XAML (Extensible Application Markup Language)
controls, 382–383
defined, 380–381
event handling, 389
layout controls, 384–388
project files, 379
syntax, 381–382
XamlTypeInfo.g.h, 380
Xml class, 276
XmlDeclaration node type, 309
XmlDocument class, 322
XML (eXtensible Markup Language)
NET and
NET XML namespaces, 306
overview, 305–306
XML processing classes, 306–307

  - parsing using XmlReader
    - creating XmlReaders, 309–310
    - handling attributes, 314–315
    - overview, 307–309
    - verifying well-formed XML, 314
    - with validation, 315–317
    - XmlReaderSettings class, 310–314
  - quick reference, 332
  - writing using XmlTextWriter, 318–322
  - XmlDocument class
    - overview, 322
    - W3C DOM and, 322
  - XmlNode class, 325–332
- XmlLang property, 308, 318
- Xml namespaces, 276
- XmlNode class, 325–332
- XmlNodeType enumeration, 309
- XmlReader class
  - parsing XML using
    - creating XmlReaders, 309–310
    - handling attributes, 314–315
    - overview, 307–309
    - verifying well-formed XML, 314
    - with validation, 315–317
    - XmlReaderSettings class, 310–314
- XmlResolver property, 310
- XmlSpace property, 318
- XmlTextReader class, 307
- XmlTextWriter class
  - creating object, 348
  - writing XML using, 318–322
- XmlValidatingReader class, 307
- XmlWriter class, 138
- x:Name attribute, 381
- XPath class, 276
- Xsl class, 276
- XSL (Extensible Stylesheet Language) processor, 276
- XSLT (Extensible Stylesheet Language Transformations), 306

Z

- ZIndex property, 388

About the author

JULIAN TEMPLEMAN is a professional consultant, trainer, and writer. He has been writing code for nearly 40 years, has been using and teaching C++ for nearly 20 of those, and has been involved with .NET since its first alpha release in 1998. He is the author or coauthor of 10 programming books, including *COM Programming with Microsoft .NET*. He currently runs a training and consultancy company in London with his wife, specializing in C++, Java and .NET programming, and software development methods.