Jesse Liberty
Siddhartha Rao
Bradley Jones

The Sixth Edition of *Sams Teach Yourself C++ in 21 Days*

More than 250,000 sold!

Sams **Teach Yourself**

C++

in **One Hour** a Day

SAMS

Sams Teach Yourself C++ in One Hour a Day

Copyright © 2009 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-32941-8

ISBN-10: 0-672-32941-7

Library of Congress Cataloging-in-Publication Data

Liberty, Jesse.
C++ in one hour a day / Jesse Liberty, Siddhartha Rao, Bradley Jones. — 6th ed.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-672-32941-8 (pbk.)
1. C++ (Computer program language) I. Rao, Siddhartha. II. Jones, Bradley. III. Title.
QA76.73.C153L528 2008
005.13'3—dc22
2008024283

Printed in the United States of America

First Printing July 2008

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

This Book Is Safari Enabled

The Safari® Enabled icon on the cover of your favorite technology book means the book is available through Safari Bookshelf. When you buy this book, you get free access to the online edition for 45 days.

Safari Bookshelf is an electronic reference library that lets you easily search thousands of technical books, find code samples, download chapters, and access technical information whenever and wherever you need it.

To gain 45-day Safari Enabled access to this book:

- Go to http://www.informit.com/onlineedition
- Complete the brief registration form
- Enter the coupon code ETR3-REFQ-5UBU-SLQ5-TYNC

If you have difficulty registering on Safari Bookshelf or accessing the online edition, please email customer-service@safaribooksonline.com.

Acquisitions Editor
Mark Taber

Development Editor
Songlin Qiu

Managing Editor
Patrick Kanouse

Project Editor
Seth Kerney

Copy Editor
Mike Henry

Indexer
WordWise Publishing Services, LLC

Proofreader
Kathy Ruiz

Technical Editors
Jon Upchurch
Dr. Mark S. Merry

Publishing Coordinator
Vanessa Evans

Book Designer
Gary Adair

Introduction

This book is designed to help you teach yourself how to program with C++. Just as you can learn to walk one step at a time, you can learn to program in C++ one hour at a time. Each lesson in this book has been designed so that you can read the entire lesson in just an hour a day. It lays emphasis on the practical usage of the language, and helps you get up-to-speed with concepts that are most important in writing C++ applications for real-world usage.

By focusing for just an hour a day at a time, you'll learn about such fundamentals as managing input and output, loops and arrays, object-oriented programming, templates, using the standard template library, and creating C++ applications—all in well-structured and easy-to-follow lessons. Lessons provide sample listings—complete with sample output and an analysis of the code—to illustrate the topics of the day.

To help you become more proficient, each lesson ends with a set of common questions and answers, a quiz, and exercises. You can check your progress by examining the quiz and exercise answers provided in Appendix D, "Answers."

Who Should Read This Book

You don't need any previous experience in programming to learn C++ with this book. This book starts you from the beginning and teaches you both the language and the concepts involved with programming C++. You'll find the numerous examples of syntax and detailed analysis of code an excellent guide as you begin your journey into this rewarding environment. Whether you are just beginning or already have some experience programming, you will find that this book's clear organization makes learning C++ fast and easy.

Organization of This Book

This is a book that appeals as much to a beginner in the language as it does to someone who wishes to understand C++ again, but from a more practical perspective. It is hence divided into five parts:

- Part I, "The Basics," introduces C++, and its syntactical details. This is very useful for absolute beginners who would first like to understand the basics of programming in C++.

- Part II, “Fundamentals of Object-Oriented Programming and C++,” introduces the object-oriented features of C++—those that set it apart from its predecessor C. This section lays the foundation for a more practical view of the language and one of its most powerful utilities, the standard template library.
- Part III, “Learning the Standard Template Library (STL),” gives you a close look at how C++ is used in real-life practical applications where quality of your application can be vastly improved by using readily available, standard-compliant constructs.
- Part IV, “More STL,” introduces you to algorithms such as sort and other STL constructs that help streamline your application and increase its reliability.
- Part V, “Advanced C++ Concepts,” discusses details and features of the programming language that not every application built using it needs to have, yet, knowing them can help in error analysis or in writing better code.

Conventions Used in This Book

Within the lessons, you’ll find the following elements that provide additional information:

TIP

These boxes highlight information that can make your C++ programming more efficient and effective.

NOTE

These boxes provide additional information related to material you just read.

FAQ

What do FAQs do?

Answer: These Frequently Asked Questions provide greater insight into the use of the language and clarify potential areas of confusion.

CAUTION

These focus your attention on problems or side effects that can occur in specific situations.

These boxes provide clear definitions of essential terms.

DO	DON'T
DO use the "Do/Don't" boxes to find a quick summary of a fundamental principle in a lesson.	**DON'T** overlook the useful information offered in these boxes.

This book uses various typefaces to help you distinguish C++ code from regular English. Actual C++ code is typeset in a special `monospace` font. Placeholders—words or characters temporarily used to represent the real words or characters you would type in code—are typeset in *`italic monospace`*. New or important terms are typeset in *italic*.

Sample Code for This Book

The code samples in this book are available online for download from the publisher's website.

LESSON 2
The Anatomy of a C++ Program

C++ programs consist of classes, functions, variables, and other component parts. Most of this book is devoted to explaining these parts in depth, but to get a sense of how a program fits together, you must see a complete working program.

In this lesson, you will learn

- The parts of a C++ program
- How the parts work together
- What a function is and what it does

A Simple Program

Even the simple program `HELLO.cpp` from Lesson 1, “Getting Started,” had many interesting parts. This section reviews this program in more detail. Listing 2.1 reproduces the original version of `HELLO.cpp` for your convenience.

LISTING 2.1 `HELLO.cpp` Demonstrates the Parts of a C++ Program

```
1:  #include <iostream>
2:
3:  int main()
4:  {
5:      std::cout << "Hello World!\n";
6:      return 0;
7:  }
```

Output

```
Hello World!
```

Analysis

On the first line, the file `iostream` is included into the current file. Here’s how that works: The first character is the `#` symbol, which is a signal to a program called the *preprocessor*. Each time you start your compiler, the preprocessor is run first. The preprocessor reads through your source code, looking for lines that begin with the pound symbol (`#`) and acts on those lines before the compiler runs. The preprocessor is discussed in further detail in Lesson 15, “An Introduction to Macros and Templates,” and in Lesson 29, “Tapping Further into the Preprocessor.”

The command `#include` is a preprocessor instruction that says, “What follows is a filename. Find that file, read it, and place it right here.” The angle brackets around the filename tell the preprocessor to look in all the usual places for this file. If your compiler is set up correctly, the angle brackets cause the preprocessor to look for the file `iostream` in the directory that holds all the include files for your compiler. The file `iostream` (input-output-stream) is used by `cout`, which assists with writing to the console. The effect of line 1 is to include the file `iostream` into this program as if you had typed it in yourself.

NOTE

The preprocessor runs before your compiler each time the compiler is invoked. The preprocessor translates any line that begins with a pound symbol (#) into a special command, getting your code file ready for the compiler.

NOTE

Not all compilers are consistent in their support for `#includes` that omit the file extension. If you get error messages, you might need to change the include search path for your compiler or add the extension to the `#include`.

The actual program starts with the function named `main()`. Every C++ program has a `main()` function. A function is a block of code that performs one or more actions. Usually, functions are invoked or called by other functions, but `main()` is special. When your program starts, `main()` is called automatically.

`main()`, like all functions, must state what kind of value it returns. The return value type for `main()` in `HELLO.cpp` is `int`, which means that this function returns an integer to the operating system when it completes. In this case, it returns the integer value `0`. A value may be returned to the operating system to indicate success or failure, or using a failure code to describe a cause of failure. This may be of importance in situations where an application is launched by another. The application that launches can use this "exit code" to make decisions pertaining to success or failure in the execution of the application that was launched.

CAUTION

Some compilers let you declare `main()` to return `void`. This is no longer legal C++, and you should not get into bad habits. Have `main()` return `int`, and simply return `0` as the last line in `main()`.

NOTE

Some operating systems enable you to test the value returned by a program. The informal convention is to return `0` to indicate that the program ended normally.

All functions begin with an opening brace (`{`) and end with a closing brace (`}`). Everything between the opening and closing braces is considered a part of the function.

The meat and potatoes of this program is in the usage of `std::cout`. The object `cout` is used to print a message to the screen. You'll learn about objects in general in Lesson 10, "Classes and Objects," and `cout` and `cin` in detail in Lesson 27, "Working with Streams." These two objects, `cin` and `cout`, are used in C++ to handle input (for example, from the keyboard) and output (for example, to the console), respectively.

`cout` is an object provided by the standard library. A *library* is a collection of classes. The standard library is the standard collection that comes with every ANSI-compliant compiler.

You designate to the compiler that the `cout` object you want to use is part of the standard library by using the namespace specifier `std`. Because you might have objects with the same name from more than one vendor, C++ divides the world into namespaces. A namespace is a way to say, "When I say `cout`, I mean the `cout` that is part of the standard namespace, not some other namespace." You say that to the compiler by putting the characters `std` followed by two colons before the `cout`.

Here's how `cout` is used: Type the word `cout`, followed by the output redirection operator (<<). Whatever follows the output redirection operator is written to the console. If you want a string of characters written, be certain to enclose them in double quotes (`"`), as visible in Listing 2.1.

NOTE

You should note that the redirection operator is two greater-than signs with no spaces between them.

A text string is a series of printable characters. The final two characters, `\n`, tell `cout` to put a new line after the words `Hello World!`

The `main()` function ends with the closing brace (}).

A Brief Look at `cout`

In Lesson 27, you will see how to use `cout` to print data to the screen. For now, you can use `cout` without fully understanding how it works. To print a value to the screen, write the word `cout`, followed by the insertion operator (<<), which you create by typing the less-than character (<) twice. Even though this is two characters, C++ treats it as one.

Follow the insertion character with your data. Listing 2.2 illustrates how this is used. Type in the example exactly as written, except substitute your own name where you see Jesse Liberty (unless your name *is* Jesse Liberty).

LISTING 2.2 Using `cout`

```
// Listing 2.2 using std::cout
#include <iostream>
int main()
{
    std::cout << "Hello there.\n";
```

LISTING 2.2 Continued

```
 6:      std::cout << "Here is 5: " << 5 << "\n";
 7:      std::cout << "The manipulator std::endl ";
 8:      std::cout << "writes a new line to the screen.";
 9:      std::cout <<  std::endl;
10:      std::cout << "Here is a very big number:\t" << 70000;
11:      std::cout << std::endl;
12:      std::cout << "Here is the sum of 8 and 5:\t";
13:      std::cout << 8+5 << std::endl;
14:      std::cout << "Here's a fraction:\t\t";
15:      std::cout << (float) 5/8 << std::endl;
16:      std::cout << "And a very very big number:\t";
17:      std::cout << (double) 7000 * 7000 << std::endl;
18:      std::cout << "Don't forget to replace Jesse Liberty ";
19:      std::cout << "with your name...\n";
20:      std::cout << "Jesse Liberty is a C++ programmer!\n";
21:      return 0;
22:  }
```

Output ▼

```
Hello there.
Here is 5: 5
The manipulator endl writes a new line to the screen.
Here is a very big number:      70000
Here is the sum of 8 and 5:     13
Here's a fraction:              0.625
And a very very big number:     4.9e+007
Don't forget to replace Jesse Liberty with your name...
Jesse Liberty is a C++ programmer!
```

CAUTION

Some compilers have a bug that requires that you put parentheses around the addition before passing it to `cout`. Thus, line 13 would change to

```
cout << (8+5) << std::endl;
```

Analysis ▼

The statement `#include <iostream>` causes the `iostream` file to be added to your source code. This is required if you use `cout` and its related functions.

The program starts with the the simplest use of `cout` by printing a string; that is, a series of characters. The symbol `\n` is a special formatting character. It tells `cout` to print a newline character to the screen; it is pronounced “slash-n” or “new line.”

Three values are passed to `cout` in this line:

```
    std::cout << "Here is 5: " << 5 << "\n";
```

In here, each value is separated by the insertion operator (<<). The first value is the string `"Here is 5: "`. Note the space after the colon. The space is part of the string. Next, the value `5` is passed to the insertion operator and then the newline character (always in double quotes or single quotes) is passed. This causes the line

```
Here is 5: 5
```

to be printed to the console. Because no newline character is present after the first string, the next value is printed immediately afterward. This is called concatenating the two values.

Note the usage of the manipulator `std::endl`. The purpose of `endl` is to write a new line to the console, thus presenting an alternative to '\n'. Note that `endl` is also provided by the standard library; thus, `std::` is added in front of it just as `std::` was added for `cout`.

NOTE

`endl` stands for ***end l***ine and is end-ell rather than end-one. It is commonly pronounced "end-ell."

The use of `endl` is preferable to the use of `\n` because `endl` is adapted to the operating system in use, whereas `\n` might not be the complete newline character required on a particular operating system or platform.

The formatting character `\t` inserts a tab character. Other lines in the sample demonstrate how `cout` can display integers, decimal equivalents, and so on. The terms `(float)` and `(double)` tell `cout` that the number is to be displayed as a floating-point value. All this will be explained in Lesson 3, " Using Variables, Declaring Constants," when data types are discussed.

You should have substituted your name for `Jesse Liberty`. If you do this, the output should confirm that you are indeed a C++ programmer. It must be true, because the computer said so!

Using the Standard Namespace

You'll notice that the use of `std::` in front of both `cout` and `endl` becomes rather distracting after a while. Although using the namespace designation is good form, it is tedious to type. The ANSI standard allows two solutions to this minor problem.

The first is to tell the compiler, at the beginning of the code listing, that you'll be using the standard library `cout` and `endl`, as shown on lines 5 and 6 of Listing 2.3.

LISTING 2.3 Using the `using` Keyword

```
 1:  // Listing 2.3 - using the using keyword
 2:  #include <iostream>
 3:  int main()
 4:  {
 5:      using std::cout;  // Note this declaration
 6:      using std::endl;
 7:
 8:      cout << "Hello there.\n";
 9:      cout << "Here is 5: " << 5 << "\n";
10:      cout << "The manipulator endl ";
11:      cout << "writes a new line to the screen.";
12:      cout <<  endl;
13:      cout << "Here is a very big number:\t" << 70000;
14:      cout <<  endl;
15:      cout << "Here is the sum of 8 and 5:\t";
16:      cout << 8+5 << endl;
17:      cout << "Here's a fraction:\t\t";
18:      cout << (float) 5/8 << endl;
19:      cout << "And a very very big number:\t";
20:      cout << (double) 7000 * 7000 << endl;
21:      cout << "Don't forget to replace Jesse Liberty ";
22:      cout << "with your name...\n";
23:      cout << "Jesse Liberty is a C++ programmer!\n";
24:      return 0;
25:  }
```

Output ▼

```
Hello there.
Here is 5: 5
The manipulator endl writes a new line to the screen.
Here is a very big number:      70000
Here is the sum of 8 and 5:     13
Here's a fraction:              0.625
And a very very big number:     4.9e+007
Don't forget to replace Jesse Liberty with your name...
Jesse Liberty is a C++ programmer!
```

Analysis ▼

You will note that the output is identical to the previous listing. The only difference between Listing 2.3 and Listing 2.2 is that on lines 5 and 6, additional statements inform the compiler that two objects from the standard library will be used. This is done with the keyword `using`. After this has been done, you no longer need to qualify the `cout` and `endl` objects.

The second way to avoid the inconvenience of writing `std::` in front of `cout` and `endl` is to simply tell the compiler that your listing will be using the entire standard namespace; that is, any object not otherwise designated can be assumed to be from the standard namespace. In this case, rather than writing `using std::cout;`, you would simply write `using namespace std;`, as shown in Listing 2.4.

LISTING 2.4 Using the `namespace` Keyword

```
// Listing 2.4 - using namespace std
#include <iostream>
int main()
{
    using namespace std;  // Note this declaration

    cout << "Hello there.\n";
    cout << "Here is 5: " << 5 << "\n";
    cout << "The manipulator endl ";
    cout << "writes a new line to the screen.";
    cout <<  endl;
    cout << "Here is a very big number:\t" << 70000;
    cout <<  endl;
    cout << "Here is the sum of 8 and 5:\t";
    cout << 8+5 << endl;
    cout << "Here's a fraction:\t\t";
    cout << (float) 5/8 << endl;
    cout << "And a very very big number:\t";
    cout << (double) 7000 * 7000 << endl;
    cout << "Don't forget to replace Jesse Liberty ";
    cout << "with your name...\n";
    cout << "Jesse Liberty is a C++ programmer!\n";
    return 0;
}
```

Analysis ▼

Again, the output is identical to the earlier versions of this program. The advantage to writing `using namespace std;` is that you do not have to specifically designate the objects you're actually using (for example, `cout` and `endl;`). The disadvantage is that you run the risk of inadvertently using objects from the wrong library.

Purists prefer to write `std::` in front of each instance of `cout` or `endl`. The lazy prefer to write `using namespace std;` and be done with it. In this book, most often the individual items being used are declared, but from time to time each of the other styles are presented just for fun.

Commenting Your Programs

When you are writing a program, your intent is always clear and self-evident to you. Funny thing, though—a month later, when you return to the program, it can be quite confusing and unclear. No one is ever certain how the confusion creeps into a program, but it nearly always does.

To fight the onset of bafflement, and to help others understand your code, you need to use comments. Comments are text that is ignored by the compiler, but that can inform the reader of what you are doing at any particular point in your program.

Types of Comments

C++ comments come in two flavors: single-line comments and multiline comments. Single-line comments are accomplished using a double slash (`//`) . The double slash tells the compiler to ignore everything that follows, until the end of the line.

Multiline comments are started by using a forward slash followed by an asterisk (`/*`). This “slash-star” comment mark tells the compiler to ignore everything that follows until it finds a “star-slash” (`*/`) comment mark. These marks can be on the same line or they can have one or more lines between them; however, every `/*` must be matched with a closing `*/`.

Many C++ programmers use the double-slash, single-line comments most of the time and reserve multiline comments for blocking out large blocks of a program. You can include single-line comments within a block commented out by the multiline comment marks; everything, including the double-slash comments, is ignored between the multiline comment marks.

NOTE

The multiline comment style has been referred to as *C-style* because it was introduced and used in the C programming language. Single-line comments were originally a part of C++ and not a part of C; thus, they have been referred to as *C++-style*. The current standards for both C and C++ now include both styles of comments.

Using Comments

Some people recommend writing comments at the top of each function, explaining what the function does and what values it returns. Functions should be named so that little ambiguity exists about what they do, and confusing and obscure bits of code should be redesigned and rewritten so as to be self-evident. Comments should not be used as an excuse for obscurity in your code.

This is not to suggest that comments ought never be used, only that they should not be relied upon to clarify obscure code; instead, fix the code. In short, you should write your code well, and use comments to supplement understanding. Listing 2.5 demonstrates the use of comments, showing that they do not affect the processing of the program or its output.

LISTING 2.5 Demonstrates Comments

```
 1:  #include <iostream>
 2:
 3:  int main()
 4:  {
 5:      using std::cout;
 6:
 7:      /* this is a comment
 8:      and it extends until the closing
 9:      star-slash comment mark */
10:      cout << "Hello World!\n";
11:      // this comment ends at the end of the line
12:      cout << "That comment ended!\n";
13:
14:      // double-slash comments can be alone on a line
15:      /* as can slash-star comments */
16:      return 0;
17:  }
```

Output ▼

```
Hello World!
That comment ended!
```

Analysis ▼

The comment on lines 7–9 is completely ignored by the compiler, as are the comments on lines 11, 14, and 15. The comment on line 11 ended with the end of the line. The comments on lines 7 and 15 required a closing comment mark.

NOTE

A third style of comment is supported by some C++ compilers. These comments are referred to as *document comments* and are indicated using three forward slashes (`///`). The compilers that support this style of comment allow you to generate documentation about the program from these comments. Because these are not currently a part of the C++ standard, they are not covered here.

A Final Word of Caution About Comments

Comments that state the obvious are less than useful. In fact, they can be counterproductive because the code might change and the programmer might neglect to update the comment. What is obvious to one person might be obscure to another, however, so judgment is required when adding comments. The bottom line is that comments should not say *what* is happening, they should say *why* it is happening.

Functions

Although `main()` is a function, it is an unusual one. To be useful, a function must be called, or *invoked*, during the course of your program. `main()` is invoked by the operating system.

A program is executed line-by-line in the order it appears in your source code until a function is reached. Then the program branches off to execute the function. When the function finishes, it returns control to the line of code immediately following the call to the function.

A good analogy for this is sharpening your pencil. If you are drawing a picture and your pencil point breaks, you might stop drawing, go sharpen the pencil, and then return to what you were doing. When a program needs a service performed, it can call a function to perform the service and then pick up where it left off when the function is finished running. Listing 2.6 demonstrates this idea.

NOTE

Functions are covered in more detail in Lesson 6, “Organizing Code with Functions.” The types that can be returned from a function are covered in more detail in Lesson 3, “Using Variables, Declaring Constants.” The information provided in the current lesson is to present you with an overview because functions will be used in almost all of your C++ programs.

LISTING 2.6 Demonstrating a Call to a Function

```
 1:  #include <iostream>
 2:
 3:  // function Demonstration Function
 4:  // prints out a useful message
 5:  void DemonstrationFunction()
 6:  {
 7:      std::cout << "In Demonstration Function\n";
 8:  }
 9:
10:  // function main - prints out a message, then
11:  // calls DemonstrationFunction, then prints out
12:  // a second message.
13:  int main()
14:  {
15:      std::cout << "In main\n" ;
16:      DemonstrationFunction();
17:      std::cout << "Back in main\n";
18:      return 0;
19:  }
```

Output ▼

```
In main
In Demonstration Function
Back in main
```

Analysis ▼

The function `DemonstrationFunction()` is defined on lines 6–8. When it is called, it prints a message to the console screen and then returns.

Line 13 is the beginning of the actual program. On line 15, `main()` prints out a message saying it is in `main()`. After printing the message, line 16 calls `DemonstrationFunction()`. This call causes the flow of the program to go to the `DemonstrationFunction()` function on line 5. Any commands in `DemonstrationFunction()` are then executed. In this case, the entire function consists of the code on line 7, which prints another message. When `DemonstrationFunction()` completes (line 8), the program flow returns to from where it was called. In this case, the program returns to line 17, where `main()` prints its final line.

Using Functions

Functions either return a value or they return `void`, meaning they do not return anything. A function that adds two integers might return the sum, and thus would be defined to return an integer value. A function that just prints a message has nothing to return and would be declared to return `void`.

Functions consist of a header and a body. The header consists, in turn, of the return type, the function name, and the parameters to that function. The parameters to a function enable values to be passed into the function. Thus, if the function were to add two numbers, the numbers would be the parameters to the function. Here's an example of a typical function header that declares a function named `Sum` that receives two integer values (`first` and `second`) and also returns an integer value:

```
int Sum( int first, int second)
```

A *parameter* is a declaration of what type of value will be passed in; the actual value passed in when the function is called is referred to as an argument. Many programmers use the terms *parameters* and *arguments* as synonyms. Others are careful about the technical distinction. The distinction between these two terms is not critical to your programming C++, so you shouldn't worry if the words get interchanged.

The body of a function consists of an opening brace, zero or more statements, and a closing brace. The statements constitute the workings of the function.

A function might return a value using a `return` statement. The value returned must be of the type declared in the function header. In addition, this statement causes the function to exit. If you don't put a `return` statement into your function, it automatically returns `void` (nothing) at the end of the function. If a function is supposed to return a value but does not contain a `return` statement, some compilers produce a warning or error message.

Listing 2.7 demonstrates a function that takes two integer parameters and returns an integer value. Don't worry about the syntax or the specifics of how to work with integer values (for example, `int first`) for now; that is covered in detail in Lesson 3.

LISTING 2.7 `FUNC.cpp` Demonstrates a Simple Function

```
#include <iostream>
int Add (int first, int second)
{
    std::cout << "Add() received "<< first << " and "<< second <<
➥"\n";
    return (first + second);
}

int main()
{
    using std::cout;
    using std::cin;


```

LISTING 2.7 Continued

```
14:      cout << "I'm in main()!\n";
15:      int a, b, c;
16:      cout << "Enter two numbers: ";
17:      cin >> a;
18:      cin >> b;
19:      cout << "\nCalling Add()\n";
20:      c=Add(a,b);
21:      cout << "\nBack in main().\n";
22:      cout << "c was set to " << c;
23:      cout << "\nExiting...\n\n";
24:      return 0;
25:  }
```

Output ▼

```
I'm in main()!
Enter two numbers: 3 5

Calling Add()
In Add(), received 3 and 5

Back in main().
c was set to 8
Exiting...
```

Analysis ▼

The function `Add()` is defined on line 2. It takes two integer parameters and returns an integer value. The program itself begins on line 8. The program prompts the user for two numbers (line 16). The user types each number, separated by a space, and then presses the Enter key. The numbers the user enters are placed in the variables `a` and `b` on lines 17 and 18. On line 20, the `main()` function passes the two numbers typed in by the user as arguments to the `Add()` function.

Processing branches to the `Add()` function, which starts on line 2. The values from `a` and `b` are received as parameters `first` and `second`, respectively. These values are printed and then added. The result of adding the two numbers is returned on line 5, at which point the function returns to the function that called it—`main()`, in this case.

On lines 17 and 18, the `cin` object is used to obtain a number for the variables `a` and `b`. Throughout the rest of the program, `cout` is used to write to the console. Variables and other aspects of this program are explored in depth in the next lesson.

Methods Versus Functions

A function by any other name is still just a function. It is worth noting here that different programming languages and different programming methodologies might refer to functions using a different term. One of the more common words used is *method*. Method is simply another term for functions that are part of a class.

Summary

The difficulty in learning a complex subject, such as programming, is that so much of what you learn depends on everything else there is to learn. Today's lesson introduced the basic parts of a simple C++ program.

Q&A

Q What does `#include` do?

A This is a directive to the preprocessor that runs when you call your compiler. This specific directive causes the file in the `<>` named after the word `#include` to be read in as if it were typed in at that location in your source code.

Q What is the difference between `//` comments and `/*` style comments?

A The double-slash comments (`//`) expire at the end of the line. Slash-star (`/*`) comments are in effect until a closing comment mark (`*/`). The double-slash comments are also referred to as *single-line comments*, and the slash-star comments are often referred to as *multiline comments*. Remember, not even the end of the function terminates a slash-star comment; you must put in the closing comment mark or you will receive a compile-time error.

Q What differentiates a good comment from a bad comment?

A A good comment tells the reader *why* this particular code is doing whatever it is doing or explains what a section of code is about to do. A bad comment restates what a particular line of code is doing. Lines of code should be written so that they speak for themselves. A well-written line of code should tell you what it is doing without needing a comment.

Workshop

The Workshop provides quiz questions to help you solidify your understanding of the material covered and exercises to provide you with experience in using what you've learned. Try to answer the quiz and exercise questions before checking the answers in Appendix D, and be certain that you understand the answers before continuing to the next lesson.

Quiz

1. What is the difference between the compiler and the preprocessor?
2. Why is the function `main()` special?
3. What are the two types of comments and how do they differ?
4. Can comments be nested?
5. Can comments be longer than one line?

Exercises

1. Write a program that writes `I love C++` to the console.
2. Write the smallest program that can be compiled, linked, and run.
3. **BUG BUSTERS:** Enter this program and compile it. Why does it fail? How can you fix it?

   ```
   #include <iostream>
   main()
   {
        std::cout << Is there a bug here?";
   }
   ```

4. Fix the bug in Exercise 3 and recompile, link, and run it.
5. Modify Listing 2.7 to include a subtract function. Name this function `Subtract()` and use it in the same way that the `Add()` function was called. You should also pass the same values that were passed to the `Add()` function.

Index

SYMBOLS

! (logical NOT) operator, 119

!= (not equal) operator, 106

” (quotation marks), 733

# (pound symbol), 28

#define preprocessor directives, 428

#includes, 29

% (modulus) operator, 99

%d conversion specifier, 672

%f conversion specifier, 672

%l conversion specifier, 672

%ld conversion specifier, 672

%s conversion specifier, 672

& (address of) operator, 204-206, 233-234

& (AND) operator, 232, 255-256, 744

&& (logical AND) operator, 118

() (parentheses)

- macro syntax, 430-431
- nesting, 104

* (indirection) operator, 208-209, 255

+ (addition) operator, 397-399

++ (increment) operator, 101, 387-390

- postfix, 102
- prefix, 101-103

+= (self-assigned addition operator), 100

- (decrement) operator, 101-103, 387-390

- (subtraction) operator, 98-99, 397-399

. (dot) operator, 277

/* comment notation, 35

// comment notation, 35

0 (null character), 651

= (assignment) operator, 53, 97

= (equal sign), 77

= (less than or equals) operator, 106

= 0 notation, 373

== (equals) operator, 106

?[:] (conditional operator), 121-123

A

abstract classes, 381

abstract data types (ADTs), 368-373

- declaring, 373
- deriving from other ADTs, 378-381
- pure virtual functions, 372

abstraction

- hierarchies of, 377-380
- in programming, 159

access control keywords, 277

access labels, 754-755

accessing

arrays, 79

class members, 271

data members, 273-274

private, 272, 275

public, 273-274

derived objects, 307-308

elements, 72-74, 480-482

memory addresses, 212-213

methods from base classes, 328

STL string classes, 461-463

accessor methods, 275-276

adaptive containers, 601. *See also* containers

queues, 602-603

instantiating, 607

member functions, 607-610

priority queues, 610-614

stacks, 602

instantiating, 603-604

member functions, 604-606

adaptive function objects, 554

Add() function, 40

adding

constructors and destructors, 281

inheritance to two lists, 350

addition (+) operator, 100, 397-399

address of (&) operator, 204-206, 233-234

addresses

memory addresses, 204, 209-210

determining, 204-206

examining, 212-213

retrieving, 208-209

storing in pointers, 206-207

target addresses

references, 233

returning, 233-236

ADTs (abstract data types), 368-373

declaring, 373

deriving from other ADTs, 378-381

pure virtual functions, 372

\a escape code, 62

aggregation, 337-338

algorithms

containers, 573

STL, 569

classification of, 570-573

copy and remove operations, 585-588

counting and finding elements, 573-576

initializing elements, 578-581

inserting elements, 595-597

overview of, 570

partitioning ranges, 593-595

processing elements, 581-583

replacing elements, 589-590

searching ranges, 576-578

sorting collections, 591-593

transforming ranges, 583-585

aliases, creating with typedef, 55-56

aligning braces ({ }), 750

allocating

memory, 218-219

pointers, 220

ambiguity resolution (multiple inheritance), 358-359

American National Standards Institute (ANSI), 14-15

ampersands (&), 744

address of operator, 204-206, 233-234

logical AND (&&) operator, 118

reference operator, 232, 255-256

AND bitwise operator, 744

AND operators, logical (&&), 118

ANSI (American National Standards Institute), 14-15

appending files, 678-679

applications. *See* programs

applying

const cast, 422-423

conventional pointers, 630

dynamic cast, 419-421

function objects, 554

binary functions, 561-563

binary predicates, 563-565

unary functions, 554-559

unary predicates, 559-561

private inheritance, 335-337

reinterpret cast, 421-422

static cast, 418

STL string classes, 459

accessing, 461-463

case conversion, 469-470

concatenation, 463-464

find member function, 464-466

instantiating, 459-461

reversing, 468-469

template-based implementation, 471

truncating, 467-468

templates, 440-441

Area() function, 133

argc (argument count), 683
arguments, 39, 129, 141-142
- command-line processing, 682-686
- defaults, 145-147
- passing
  - to base constructors, 311-315
  - by reference, 237-249
  - by value, 137-138, 238-239

argv (argument vector), 683
arithmetic operators, 97
- combining with assignment operators, 100
- modulus (%), 99
- subtraction (-), 98-99

arrays, 72
- bugs, 74
- char, 84-86
- characters, 458
- declaring, 72, 78-79
- defined, 72
- elements, 72-74, 79
- fence post errors, 76-77
- filling, 85-86
- initializing, 77-78
- integer arrays, 73
- memory, 84
- multidimensional, 80
  - declaring, 80
  - initializing, 81-82
- names, 213-215
- pointer arrays, 215
- pointer to, 213-215
- sizes, 78-79
- STL dynamic array class
  - accessing elements in vectors, 480-482
  - deleting elements from vectors, 482-484
  - inserting elements into vectors, 476-479
  - instantiating vectors, 474-475
  - need for, 473
  - size and capacity of vectors, 484-485
  - vectors, 474
- writing past the end of, 74-76

ASCII character sets, 47
assemblers, 722
assert() macro, 734-735, 755
- debugging functions, 735-737
- exceptions, 736
- source code, 734-735

assigning
- addresses to references, 235-236
- values to variables, 53-55, 271

assignment operators (=), 53, 97, 399-401
- combining with math operators, 100

associative containers, 448-449
asterisks (*)
- indirection operator, 208-209
- syntax, 255

auto_ptr smart pointers, 638-640

B

backslashes (\), 62
backspaces, 62
base 10 numbers, 763-764
- converting to base 2, 769-770
- converting to base 6, 766
- converting to base 7, 765-766
- converting to binary, 766-767

base 16 numbers, 768-772
base 2 numbers, 766-768
base 7 numbers, 765
base 8 numbers, 764
base classes, 302
- common, 359
- functions, 316-317
- methods
  - accessing, 328
  - calling, 320-321
  - hiding, 318-319

base constructors, 309
- overloading, 311-315
- passing arguments to, 311-315

base destructors, 309
base methods, calling, 320-321
bases classes, 359-363
\b escape code, 62
bidirectional iterators, 452
binary files, compared to text files, 680-682
binary functions, 554-563
binary numbers, 766-768
binary operators
- addition/subtraction, 397-399
- assignment, 399-401
- types, 396

binary predicates, 554, 563-565
binding, dynamic, 326
bits, 767
- clearing, 745-746
- fields, 746-750
- flags, 617
  - bitset class, 618-623
  - vector bool, 623-625
- flipping, 746
- setting, 745
- twiddling, 744-750

bitset class, 618-621, 623
bitwise operators, 744
- AND, 744
- complement, 745
- exclusive OR, 745
- OR, 745

blocks, 94-95
- catch, 693, 703-706
- try, 693-696

body (function), 39

bool data type, 47, 105

braces ({ }), 29, 94
- aligning, 750
- nested if statements, 115-118

branching
- programs, 162
- relational operators, 108-109

break statements, 173-175

breaking while loops, 173

breakpoints, 721

buffers, 645-646
- copying strings to, 87-88
- implementing, 647
- uninitialized, 85

bugs, 18, 690. *See also* debugging; troubleshooting
- arrays, 74
- debugging, 721
  - assemblers, 722
  - breakpoints, 721
  - examining memory, 722
  - watch points, 721
- fence post errors, 76-77
- stray pointers, 226

built-in functions, 128-131

built-in text editors, 24

bulletproof programs, 690

bytes, 767

C

c filename extension, 16-17

C language, 14, 35

C# language, 50

calling
- constructors, 355-358
- functions, 37-38, 153-162
- methods, 320-321

cannot find file error messages, 20

capabilities classes (inheritance), 368

capacity of vectors, 484-485

capitalization style guidelines, 753

caret (^), 745

carriage return escape characters (\r), 62

case conversion, STL string classes, 469-470

case values (switch statements), 192

case-sensitivity, variable names, 49

casting down, inheritance, 347, 350

casting operators, 417
- const cast, 422-423
- defined, 416
- dynamic cast, 419-421
- need for, 416
- reinterpret cast, 421-422
- static cast, 418
- troubleshooting, 423-424
- unpopular styles, 417

Cat class
- accessor functions, 286
- Cat object, initializing, 283, 285
- data members, accessing, 273-274
- declaring, 269, 289
- implementing, 290
- initializing, 283-285
- methods
  - accessor methods, 276
  - GetAge(), 280
  - GetWeight(), 289
  - implementing, 278, 280
  - Meow(), 276, 281
  - SetAge(), 280-281

CAT.cpp (code listing), 290

Cat.h (code listing), 289

catch blocks, 693
- multiple specifications, 703-706
- placing, 702

catching
- exceptions, 702-703
- multiple exceptions, 703-706

Celsius, converting to Fahrenheit, 135

cerr object, 648

char arrays, 84-86

char data type, 47

char variables, 45, 59-60
- character encoding, 60-61
- escape characters, 61-62
- sizes, 59

characters, 59-60
- arrays, declaring, 458
- ASCII character sets, 47
- encoding, 60-61
- escape characters, 61-62
- fill characters, setting, 667-668
- null, 84, 651
- reference parameters (get() method), 656-657
- sizes, 59
- STL string classes, accessing, 461-463

cin object, 648-650
- input
  - multiple input, 651-654
  - strings, 651
- member functions, 654-657
- methods
  - get(), 655-659
  - getline(), 659-660
  - ignore(), 660-662
  - peek(), 662-663
  - putback(), 662-663
- return values, 654

class keyword, 269

class statement, 277-278

classes, 268, 277
- abstract, 381
- base, 316-317. *See also* base classes
- bitset, 618-623
- Cat
  - declaring, 269, 289
  - implementing, 290
- comparing
  - to objects, 270-271
  - to structures, 295
- container, 513
  - advantages of set and multiset, 529
  - deleting elements, 519-528
  - inserting elements, 515-517
  - instantiating set objects, 514-515
  - map and multimap, 533
  - searching elements, 517-518
- data members, 268
  - accessing, 271-274
  - other classes as, 291-295
  - private, 272-273, 306-307
  - protected, 306-307
  - public, 272-274
- declaring, 269, 288
- defined, 268
- defining, 755
- derived, 302-306, 368-372
- Dog, 304-306
- exceptions, creating, 698-702
- fstream, 647
- inheritance
  - casting down, 347-350
  - limitations, 344-346
  - percolating shared functions, 346-347
  - relationships among, 303
- invariants, 737-742
- ios, 647
- iostream, 647
- istream, 647
- member functions, 269
- methods
  - constants, 286-287
  - defining, 272
  - implementing, 278-281
  - inline, 288-290
  - public accessor methods, 275-276
- mixins, 368
- naming conventions, 269-270
- objects
  - defining, 270, 277
  - values, 271
- ofstream, 675
- point, declaring, 291-292
- polymorphism, 13
- Rectangle, declaring, 292-294
- security, 277
- Shape, 369-371
- shared bases classes, 359-363
- STLs
  - accessing, 461-463, 480-482
  - applying, 459
  - case conversion, 469-470
  - concatenation, 463-464
  - deleting elements from vectors, 482-484
  - deque, 486-488
  - find member function, 464-466
  - inserting elements into vectors, 476-479
  - instantiating, 459-461, 474-475
  - need for, 458-459, 473
  - reversing, 468-469
  - size and capacity of vectors, 484-485
  - strings, 457
  - template-based implementation, 471
  - truncating, 467-468
  - vectors, 474
- String, 89-90
- subclasses, 291-295
- templates, 436-439
- writing to files, 680-682

classification of STL algorithms, 570
- mutating, 571-573
- nonmutating, 570-571

clearing bits, 745-746

clog object, 648

code listings
- abstract data types
  - deriving from other ADTs, 378-380
  - example, 373
- accessing STL strings, 461-463
- address of (&) operator, 205
- arrays
  - characters
  - filling with, 86
  - consts and enums, 78
  - filling, 85-86
  - integer array, 73
  - pointer relationships, 214
  - writing past the end of, 75
- bit fields, 747-748
- bitset class
  - instantiating, 618-619
  - member methods, 620-623
  - operators, 619-620
- Calendar class, 388-389
- Cat.cpp, 290
- Cat.h, 289
- cin object
  - example, 649-650
  - multiple input, 651-653

How can we make this index more useful? Email us at indexes@samspublishing.com

classes
  accessor methods, 276
  declaring, 291-292
  public data members, 274
command-line arguments, 683-685
comments example, 36
compiler errors, 23
compiling with symbols, 721
const pointers, passing, 249-251
constant integers, 66
cout object, 30-32
  adjusting width of output, 666-667
data slicing when passing by value, 328-330
dynamic casting, 419-421
else keyword, 111
enumerated constants, 65
exceptions
  class hierarchies, 706-709
  multiple, 703-705
  sample program, 691
  templates, 717-719
  throwing, 697-701
expressions, evaluating, 96
files
  appending, 678-679
  opening for input/output, 676-677
  writing a class to, 681-682
fill() method, 668
for loop, 182
free store, creating/deleting objects, 223
FUNC.cpp, 39-40
functions
  binary, 561-563
  binary predicates, 563-565
  calling, 38
  declaring, 132-133
  default parameters, 146
  inline, 152-153
  objects holding state, 557-558
  polymorphism, 148-150
  unary, 555-556
  unary predicates, 559-560
get() methods
  character arrays, 658
  character reference parameters, 656-657
  with no parameters, 655-656
getline() method, 659
getting data out of an exception object, 709
HELLO.cpp, 20, 28
if statements
  braces ({}), 115-117
  else keyword, 111-112
  nesting, 113-114
ignore() method, 661
inheritance
  calling multiple constructors, 355-357
  casting down, 348-349
  common base classes, 359-362
  constructors and destructors called, 309-311
  derived objects, 307-308
  If Horses Could Fly example (limitations of inheritance), 345-346
  overloaded constructors, 311-314
  simple inheritance, 304-305
  virtual, 364-366
inline functions, 433
Invariants() method, 737-742
linked lists
  deleting elements, 497-500
  inserting elements, 493-497
  instantiating, 492-493
  reversing elements, 500-501
  sorting elements, 502-511
loops
  break and continue statements, 173-174
  do...while, 178-179
  Fibonacci series application, 190-191
  forever loop, 196-198
  null statements in for loops, 184-186
  simple while loop, 170
  while loops reexamined, 180-181
macros
  assert(), 734-735
  parentheses, 430-431
map and multimap
  customizing sort predicates, 543-547
  deleting elements, 540-543
  inserting elements, 535-538
  instantiating, 535
  searching elements, 538-540
memory leaks, 258
methods
  calling base method from overridden methods, 320-321
  calling multiple virtual functions, 324-325
  constructors and destructors, 283-285
  hiding, 318-319
  implementing, 279-280

overriding, 316-317
virtual, 323
virtual copy constructors, 331-333
multiple return statements, 143-144
namespace keyword, 34
operator() function, 411-412
operators
assignment, 400-405
binary, 397-399
branching based on relational operators, 108-109
conditional, 122
conversion, 394
overloading, 406-408
prefix and postfix, 102
subscript, 409-410
subtraction and integer overflow, 98
passing
objects by reference, 247-248
by reference in exceptions, 713-716
by reference using pointers, 239-240
by value, 138, 238
peek() and putback() methods, 662-663
pointers
allocating and deleting, 220
applying, 220
data manipulation, 210-211
examining memory addresses, 212-213
stray, creating, 224
postfix increment operators, 390
printf() method, 672-673
printing
characters based on numbers, 60-61
values in DEBUG mode, 742-743
private inheritance, 335-337
pure virtual functions, 374-377
put() method, 664
queues
instantiating, 607
member functions, 607-610
priority, 610-614
Rect.cpp, 292-293
recursion, 155-156
references
addresses, 234
assigning to, 235-236
creating, 232-233
to nonexistent objects, 257
passing to objects, 252-253
returning values with, 245
relational operators, 108
returning values with pointers, 243
set and multiset
deleting elements, 519-528
inserting elements, 515-517
instantiating, 514-515
searching elements, 517-518
setf() method, 669-670
Shape classes, 369-371
signed integers, wrapping around, 58-59
simple smart Pointer class, 392
smart pointers
auto_ptr, 638-640
Copy on Write, 635
deep copy, 633-635
destructive copy, 636-637
implementing, 631-632
reference counting, 635-636
reference-linked, 636
types of, 632-633
solving the nth Fibonacci number, 191
stacks
instantiating, 603-604
member functions, 604-606
std[::]auto_ptr, 391
STL algorithms
copy and remove operations, 585-588
counting and finding elements, 573-576
initializing elements, 578-581
inserting elements, 595-597
partitioning ranges, 593-595
processing elements, 581-583
replacing elements, 589-590
searching ranges, 576-578
sorting collections, 591-593
transforming ranges, 583-585
STL deque class, 486-488
STL string classes
case conversion, 469-470
concatenation, 463-464
find member function, 464-466
reversing, 468-469

template-based implementation, 471
truncating, 467-468
STL string instantiation, 460-461
strcpy() method, 87
String class, 89
strncpy() method, 88
swap() rewritten with references, 241-242
switch statement, 193
templates
classes, 439
connecting, 453-454
typedef keyword, 55-56
unsigned integers, wrapping around, 57
using keyword, 33
variables, 54
global and local, 139-140
local, 134-135
scope, 136-137
sizes, 52
vectors
accessing elements in, 480-482
bool, 623-625
deleting elements from, 482-484
inserting elements into, 476-479
instantiating, 474-475
size and capacity of, 484-485
virtual functions in exceptions, 713-716
write() method, 665

code rot, 720

code space, 159

collections
elements, inserting, 595-597
ranges, searching, 576-578
sorting, 591-593

combining
math operators with assignment operators, 100
references and pointers, 255

command-line processing, 682-686

commands
DOS, redirect output ([<]), 648
preprocessor
#define, 428
#ifndef, 730

comments, 35
/* (C-style), 35
// (C++-style), 35
applying, 36-37
readability, 754
troubleshooting, 37

common base classes, 359

comparing private inheritance to aggregation, 337-338

comparison operators, overloading, 401-405

compile time, 25
errors, 287

compilers, 22, 428, 728
compiling with symbols, 721
errors, 23-24
intermediate files, saving, 728
macros, assert(), 734-735
overview, 9
troubleshooting, 23

compiling
errors, 23-24
Hello World program, 20-22
source code, 17-18
with symbols, 721

complement operator, 745

composition, 337-338

compound statements, 94-95

concatenation, 733
STL string classes, 463-464
strings, 733
values, 32

condition states, 675

conditional operator (?[:]), 121-123

connecting STL classes, 453-454

const cast, applying, 422-423

const methods, 286
advantages, 287
declaring, 286
implementing, 286

const pointers, 227
declaring, 227
passing, 249-252

const statement, 63-64, 286, 755

constants, 62. *See also* variables
changing, 64
defining
const statement, 63-64
#define statement, 63
enumerated, 64-65
example, 65-66
syntax, 64
values, 64-65
literals, 62
substitutions, 729
symbolic, 62-63

constructors, 281
base, passing arguments to, 311-315
copy constructors, virtual, 331-334
defaults, 282-285
inheritance, 309-311
multiple, calling, 355-358
overloading derived classes, 311, 315

consts in arrays, 78-79

container classes, 513
- advantages of set and multiset, 529
- elements
  - deleting, 519-528
  - inserting, 515-517
  - searching, 517-518
- map and multimap, 533
  - customizing sort predicates, 543-547
  - deleting elements, 540-543
  - inserting elements, 535-538
  - instantiating, 535
  - searching elements, 538-540
- set, instantiating, 514-515

containers
- adaptive, 601
  - instantiating queues, 607
  - instantiating stacks, 603-604
  - priority queues, 610-614
  - queue member functions, 607-610
  - queues, 602-603
  - stack member functions, 604-606
  - stacks, 602
- algorithms, 573
- classes. *See* container classes
- elements, initializing, 578-581
- STL, 448
  - associative, 448-449
  - selecting, 449-451
  - sequential, 448

continue statements, 173-175

conventional pointers, applying, 630

conventions, naming, 49-50

Convert() function, 135

converting
- base 10 to base 6, 766
- base 10 to base 7, 765-766
- base 10 to binary, 766-767
- decimals to binary, 769-770
- Fahrenheit/Celsius, 135
- operators, programming, 394-395
- specifiers, 671-672

copy constructors, virtual, 331-334

copy function, 585-588

Copy on Write (COW) smart pointers, 635

copying
- algorithms, 572
- STL string classes, 459-461
- strings, 87-88

counting
- algorithms, 570
- elements, 573-576
- numbers, 176-177
- references, 635-636
- variables, 188

cout object, 30-32, 648, 663
- example, 30-31
- fill characters, 667-668
- flags, 668-671
- methods
  - fill(), 667-668
  - flush(), 663
  - put(), 664-665
  - setf(), 668-671
  - width(), 666-667
  - write(), 665-666
- output width, 666-667
- passing values to, 32

COW (Copy on Write) smart pointers, 635

cp filename extension, 16

cpp filename extension, 16-17, 287

CPUs, registers, 159

customizing
- sorting, 543-547
- unary operators, 394-395

cycles, development, 18

D

dangling pointers, 224-226

data members
- accessing, 271-274
- classes, 291-295
- private, 272-273, 306-307
- protected, 306-307
- public, 272-274
- security, 277

data slicing, 328-330

data types, 47
- abstract, 368-372
  - declaring, 373
  - deriving from other ADTs, 378-381
  - example, 372-373
  - pure virtual functions, 372
- bool, 105
- creating, 267

deallocating memory, 219-221

DEBUG mode, 431, 742-749

debugging, 18, 721. *See also* troubleshooting
- assemblers, 722
- assert() macro, 735-737
- breakpoints, 721
- examining memory, 722
- inclusion guards, 731-732
- printing interim values, 431, 742-749
- watch points, 721

dec flag, 669

decimal format, 763

decimal numbers, 763-764
 converting to base 6, 766
 converting to base 7, 765-766
 converting to binary, 766-770
declaring
 abstract data types, 373
 arrays, 72, 78-79
  multidimensional, 80
  two-dimensional, 83
 character arrays, 458
 classes, 269, 288
  Cat, 269, 289
  derived classes, 304-306
  Point, 291-292
  Rectangle, 292
 constant pointers, 227
 functions, 130
  inline, 151-153, 432-433
 methods
  const, 286
  file locations, 287-288
 multiple inheritance, 354
 pointers, 206, 216
 references, 232-233
 structures, 295-296
 templates, 436
  with default parameters, 438
  with multiple parameters, 437-438
 virtual inheritance, 367
decrement (—) operator, 101-103, 387-390
deep copy smart pointers, 633-635
default constructors, 282-285
default destructors, 282-285
default parameters
 functions, 145-147
 templates, declaring, 438
default statement, 194
deference operators, 391
#define statements
 constant substitutions, 729
 tests, 729-730
define statement, 63
defining
 classes, 755
 constant substitutions, 729
 constants
  const statement, 63-64
  #define statement, 63
 functions, 131-133, 272, 288
 macros, 429-430
 methods, 272, 287-288
 objects, 270, 277
 string substitutions, 428, 728
 templates, 434-435
 variables, 45-50
  case-sensitivity, 49
  local, 134
  multiple variables, 53
  reserved words, 50
delete statement, 219-221
deleting
 duplicates, 591-593
 elements
  from vectors, 482-484
  linked lists, 497-500
  map and multimap, 540-543
  set or multiset, 519-528
 objects from free stores, 222-224
 pointers, 220
DemonstrationFunction() function, 38
deques, STL deque class, 486-488
dereference operator (*), 208-209
dereferencing pointers, 216
derivation
 ADTs from other ADTs, 378-381
 inheritance, 302-303
 syntax, 304-306
derived classes, 302, 368-372
 constructors, overloading, 311-315
 data members, accessing, 307-308
 declaring, 304-306
derived types, 13
designating standard namespaces, std[::] notation, 32
designing programs, 15-16
destructive copy smart pointers, 636-637
destructors, 282
 defaults, 282-285
 inheritance, 309-311
 virtual, 330-331
determining memory addresses, 204-206
development
 cycles, 18
 environments, 16
displaying trailing zeros, 668
division integers, 99
do...while loops, 177-178
 example, 178-179
 syntax, 179-180
documents, inserting comments, 37
Dog class
 constructors, 309-311
 declaring, 304-306
 destructors, 309-311
DOS commands, redirect output ([<]), 648
dot operator (.), 277
DoTaskOne() function, 198
double data type, 47

double quote (") escape characters, 62

Double() function, 153

Doubler() function, 144

duplicates, deleting, 591-593

dynamic binding, 326

dynamic cast, applying, 419-421

E

editors, text, 16, 24

elements
- of arrays, 72-74, 79
- characters, accessing, 461-463
- collections, searching, 576-578
- counting, 573-576
- finding, 573-576
- initializing, 578-581
- inserting, 595-597
- linked lists
  - deleting, 497-500
  - inserting, 493-497
  - reversing, 500-501
  - sorting, 502-511
- processing, 581-583
- replacing, 589-590
- set or multiset
  - deleting, 519-528
  - inserting, 515-517
  - searching, 517-518

else (#else) precompiler command, 730-731

else keyword, 111

empty for loops, 185-187

emptying buffers, 646

encapsulation, 12-13, 644

endl object, 32

endless loops
- exiting, 196
- switch statement, 195-198
- while (true), 176-177

enum keyword, 64

enumerated constants, 64-65
- example, 65-66
- syntax, 64
- values, 64-65

enumerations in arrays, 78-79

environments, 16

equal signs (=), 77
- assignment operator, 53, 97

equality operators, 402

equals (==) operator, 106

erase() function, 497-500, 519-528, 540-543

errors. *See also* troubleshooting
- compile errors, 23-24
- compile-time, 287
- error messages, cannot find file, 20
- fence post errors, 76-77
- messages, cannot find file, 20
- referencing
  - nonexistent objects, 257
- stray pointers, 226
- warning messages, 25

escape characters, 61-62

eternal loops
- exiting, 196
- switch statement, 195-198
- while (true), 176-177

evaluating
- expressions, 96
- logical operators, 119

examining memory, 212-213, 722

exceptions, 691-694
- assert() macro, 736
- catching, 702-706
- classes
  - creating, 698-702
  - hierarchies, 706-709
- compiler support, 694
- data
  - passing by reference, 713-716
  - reading, 709
- multiple, 703-706
- naming, 709-716
- programming tips, 719
- sample program, 691-692
- templates, 716-719
- throwing, 696-702
- try...catch blocks, 693-696
- virtual functions, 713-716

exclamation points (!), logical NOT operators, 119

exclusive OR bitwise operator, 745

executable files, creating, 18

executing functions, 133

exiting loops
- break statement, 173
- endless loops, 196

expressions, 95. *See also* operators
- branching switch statements, 194
- evaluating, 96
- nesting parentheses, 104
- truth, 105-107

extensions, 14

extraction operator ([<<]), 649

F

Factor() function
- pointers, 242-244
- references, 244-246

Fahrenheit, converting to Celsius, 135

false/true operations, 120-121

fence post errors, 76-77

\f escape code, 62

fib() function, 191

Fibonacci series

- recursion, 154-158
- solving with iteration, 190-192

fields, bit fields, 746-750

filename extensions

- .c, 16-17
- .cp, 16
- .cpp, 16-17, 287
- .h, 288
- .hp, 288
- .hpp, 288
- .obj, 18

files. *See also* specific file names

- appending, 678-679
- binary, compared to text files, 680-682
- executable files, creating, 18
- iostream, 28
- object files, 17
- opening for input/output, 675-677
- source files, 16
- text, compared to binary files, 680-682
- writing classes to, 681-682

fill characters, setting, 667-668

fill() method, 667-668

filling

- arrays, 85-86
- buffers, 646

find function, 517-518, 538-540

find member function, STL string classes, 464-466

finding. *See* searching

fixed flag, 669

flags

- bit, 617
  - bitset class, 618-623
  - vector bool, 623-625
- setting, 669-671
- state, 666

flipping bits, 746

float data type, 47

floating-point variables, 47

flush() method, 663

flushing

- buffers, 646
- output, 663

Fly() method, 344

for loops, 181-183

- empty loops, 185-187
- example, 182
- initialization, 181
- multiple initialization, 183-184
- nesting, 187-188
- null statements, 184-187
- scope, 189
- syntax, 182

for statements, 180-181

forever loops

- exiting, 196
- switch statement, 195-198
- while (true), 176-177

form feeds, escape characters, 62

formatting

- output
  - flags, 668-671
  - width, 666-667
- special printing characters, 61-62
- variable sizes, 45-46

forward iterators, 452

for_each algorithm, 581-583

free store

- advantages, 218
- memory, 217
  - allocating, 218-219
  - restoring, 219-221
- objects
  - creating, 222
  - deleting, 222-224

fstream classes, 647

FUNC.cpp file, 39-40

FunctionOne() function, 248

functions, 39, 128

- accessor functions, 275-276
- Add(), 40
- Area(), 133
- arguments, 39, 129, 141-142
  - defaults, 145-147
  - passing by reference, 237-249
  - passing by value, 137-138, 238-239
- binary, 554
- body, 39
- built-in, 128
- cin object, 654-657
- compared to macros, 432
- const member functions
  - advantages, 287
  - declaring, 286
  - implementing, 286
- const methods, 286
- Convert(), 135
- copy, 585-588
- declaring, 130
  - file locations, 287-288
- defining, 11, 131-133, 272, 288
- DemonstrationFunction(), 38
- DoTaskOne(), 198
- Doubler(), 144
- erase, 497-500

erase(), 519-523, 525-528, 540, 543
executing, 133
Factor()
    pointers, 242-244
    references, 244-246
fib(), 191
find, 517-518, 538-540
FUNC.cpp example, 39-40
FunctionOne(), 248
FunctionTwo(), 249
GetAge(), 280
GetArea(), 294
GetInt(), 257
GetUpperLeft(), 294
GetWeight(), 289
headers, 39
inheritance
    casting down, 347, 350
    percolating shared functions, 346-347
inline, 151, 153, 288-290, 432-433
    disadvantages, 151
    example, 153
invoking, 37-38, 158, 162
main(), 29, 128
menu(), 198
Meow(), 276, 280
myFunc(), 137
objects, 553-554
    applying, 554-565
    overview of, 554
operator(), 411-412
operators, 386. *See also* operators
overloading, 147-150, 318
overriding, 316-318
parameters, 39, 129, 141-142
polymorphism, 13, 147-150
pop_back, 483
printf(), 671-673
prototypes, 130-131
    defined, 129
    return types, 131
queues, 607-610
    priority, 611-614
recursion, 153-154, 158
    Fibonacci series example, 156-158
recursive, stop conditions, 154-155
remove, 585-588
return values, 38-39, 129, 142, 144
    multiple, 142-144
returning multiple values
    pointers, 242-244
    references, 244-246
reverse(), 500-501
SetAge(), 280-281
sizes, 141
sort(), 502-511
stacks, 604-606
statements, 141
strcpy(), 88
strncpy(), 88
swap(), 138
    pointers, 239-240
    references, 240-242
syntax, 29
unary, 554-559
vector bool, 624-625
virtual, pure, 372-377

FunctionTwo() function, 249

G

get() method, 86, 655
    characters
        arrays, 658-659
        reference parameters, 656-657
    with no parameters, 655-656
    overloading, 660

GetAge() function, 280
GetArea() function, 294
GetInt() function, 257
getline() method, 659-660
GetUpperLeft() function, 294
GetWeight() function, 289
global variables, 139
    example, 140
    limitations, 140
goto statement, 168
    loops
        disadvantages, 169
        example, 168-169
    syntax, 169
GUI (graphical user interface), 644

H

h filename extension, 288
handling exceptions. *See* exceptions
hardware, CPU registers, 159
headers (function), 39
heaps, 217. *See also* smart pointers
Hello World program, 19
    compiling, 20-22
    creating, 22-23
    source code, 20, 28
    testing, 22-23
Hello.cpp file, 20, 28
hex flag, 669
hexadecimal numbers, 62, 768-772
hiding
    compared to overriding, 320
    methods, 318-319
hierarchies
    exceptions, 706-709
    of abstraction, 377-380
history of C++, 8, 14

hp filename extension, 288
hpp filename extension, 288
Hungarian notation, 50

I

I/O objects, 647-648
- cerr, 648
- cin, 648-650
  - get() method, 655-659
  - getline() method, 659-660
  - ignore() method, 660-662
  - multiple input, 651-654
  - peek() method, 662-663
  - putback() method, 662-663
  - strings, 651
- clog, 648
- cout, 648, 663
  - fill characters, 667-668
  - fill() method, 667-668
  - flags, 668-671
  - flush() method, 663
  - output width, 666-667
  - put() method, 664-665
  - setf() method, 668-671
  - width() method, 666-667
  - write() method, 665-666
- streams. *See* streams

IDE (Integrated Development Environment), 728
identifiers, naming, 752-753
If Horses Could Fly (code listing), 345-346
if statements, 107, 109
- branching, 108-109
- else keyword, 111
- indentation styles, 110-111
- nesting, 113
  - braces ({ }), 115-118
  - example, 114-117
- semicolon notation, 109-110
- syntax, 112-113

ifndef (#ifndef) command, 730
ignore() method, 660-662
implementing
- buffers, 647
- classes, Cat, 290
- methods, 278-281
  - const methods, 286
  - inline, 288-290
- pure virtual functions, 374-377
- smart pointers, 631-632
- streams, 647
- swap() function
  - pointers, 239-240
  - references, 240-242

include files, 755
include statement, 28
inclusion guards, 731-732
increment (++) operators, 101, 387-390
- postfix, 102
- prefix, 101-103

indenting
- code, 750-752
- if statements, 110-111

indirect recursion, 153
indirection (*) operator, 207-209, 255
inequality operators, 402
inheritance, 13, 302
- adding to two lists, 350
- casting down, 347, 350
- classes, relationships among, 303
- constructors, 309-311
  - arguments, 311-315
  - overloading, 311-315
- derivation, 302-306
- destructors, 309, 311
- functions, 346-347
- limitations, 344, 346
- mixins, 368
- multiple, 351, 353
  - ambiguity resolution, 358-359
  - constructors, 355-358
  - declaring, 354
  - limitations, 367
  - objects, 354-355
  - shared base classes, 359-363
  - virtual methods, 354
- private, 335
  - applying, 335-337
  - comparing to aggregation, 337-338
- virtual, 363-366
  - declaring, 367
  - example, 365-366
- virtual methods, 322-328
  - copy constructors, 331-334
  - destructors, 330-331
  - invoking multiple, 324-326
  - memory costs, 334
  - slicing, 328-330
  - v-pointers, 326-327
  - v-tables, 326-327

initializing
- algorithms, 571
- arrays, 77-82
- elements, 578-581
- for loops, 181-184
- objects
  - Cat, 283-285
  - constructor methods, 281-282
- pointers, 206, 216
- references, 233
- statements, 182
- variables, 53

inline functions, 151-153, 432-433
- disadvantages, 151
- example, 153

inline implementation (methods), 288-290

inline statement, 151, 288-289

input
- handling
  - cin object, 649
  - extractions, 649-650, 654
  - multiple input, 651-654
  - peek() method, 662-663
  - putback() method, 662-663
  - single character input, 655
  - strings, 651, 657-660
- ignoring, 660-662
- iterators, 452
- streams, 644. *See also* streams

inserting
- elements, 595-597
  - linked lists, 493-497
  - map and multimap, 535-538
  - set or multiset, 515-517
- quotation marks ("), 733
- into vectors, 476-479

insertion operator (), 30

instantiating
- bitset classes, 618-619
- linked lists, 492-493
- map objects, 535
- queues, 607, 610
- set objects, 514-515
- stacks, 603-604
- STL string classes, 459-461
- templates, 437
- vector bool, 623-624
- vectors, 474-475

int data type, 47

integers
- arrays, 73
- division operations, 99
- integer overflow, 98
- long, 46, 56-57
- short, 46, 56-57
- signed/unsigned, 46, 57-59
- sizes, 45-46

Integrated Development Environment (IDE), 728

interfaces, GUIs, 644

interim values, printing, 431, 742-749

intermediate files (compiler), 728

internal flag, 669

International Standards Organization (ISO), 14

interpreters, 9

invariants, 737-742

Invariants() method, 737-742

invoking
- functions, 37-38, 158-162
- methods, 320-321

ios class, 647, 677

iostream class, 647

iostream file, 28

iostream library, 644

ISO (International Standards Organization), 14

istream class, 647

iteration, loops, 168
- do...while, 178-180
- for, 181-187
- goto keyword, 168-169
- nesting, 187-188
- scope, 189
- while, 169-181

iterators, STL, 451-452

J

jumps, 168

K

KB (kilobytes), 768

keywords, 50-51, 296, 773-774
- class, 269, 277-278
- const, 286
- delete, 219-221
- else, 111
- enum, 64
- goto, 168
  - disadvantages, 169
  - example, 168-169
  - syntax, 169
- inline, 151, 288-289
- namespace, 34-35
- new, 218-219
- protected, 306
- public, 280
- return, 142, 144
- struct, 295-296
- typedef, 55-56
- using, 33-34

kilobytes (KB), 768

kludges, 178

L

l-values, 97

labels, defined, 168

leaks
- creating
  - delete statement, 219
  - pointer reassignments, 221
- memory, 219, 258-259

left flags, 669

libraries
defined, 18
iostream, 644
smart pointers, 640
standard, 21
streams, 644
linked lists, STL, 491
characteristics of, 492
deleting elements, 497-500
inserting elements, 493-497
instantiating, 492-493
reversing elements, 500-501
sorting elements, 502-511
linkers, 9
linking references, 636
lists, 491. *See also* code listings; linked lists
literals, 62
local variables, 134-135
defining, 134
example, 134-135
persistence, 217
scope, 134-137
logic errors, 76-77
logical operators, 118
AND (&&), 118
NOT (!), 119
OR (||), 119
order of evaluation, 119
precedence, 120
long data type, 56-57
long int data type, 47
long integers, 46
loops, 168
do...while
example, 178-179
syntax, 179-180
endless
exiting, 196
switch statement, 195-198
while(true), 176-177
exiting, 173-175
Fibonacci series application, 190-192
for, 181-183
empty loops, 185-187
example, 182
initialization, 181
multiple initialization, 183-184
nesting, 187-188
null statements, 184-187
scope, 189
syntax, 182
goto keyword, 168
disadvantages, 169
example, 168-169
syntax, 169
overview of, 190-192
returning to top of, 173-175
while, 169
break statement, 173, 175
complex loops, 171-172
continue statement, 173-175
exiting, 173-175
returning to top of, 173-175
simple example, 169-170
skipping body of, 177-178
starting conditions, 180-181
syntax, 171
while (true), 176-177

M

macros, 429-430
assert(), 734-735, 755
debugging functions, 735-736
exceptions, 736
limitations, 736-737
source code, 734-735
comparing
to functions, 432
to templates, 432
defining, 430
disadvantages, 432
parentheses (), 430-431
predefined, 733
syntax, 429-430
when to use, 442
main() function, 29, 128
Managed Extensions to C++, 14
managing memory, 629. *See also* smart pointers
map, 533
elements
deleting, 540, 543
inserting, 535-538
searching, 538-540
instantiating, 535
sorting, 543, 546-547
mathematical operators, 97
combining with assignment operator, 100
modulus (%), 99
subtraction (-), 98-99
member functions
cin, 654-657
queues, 607-614
stacks, 604-606
member methods, 620-623
member variables, 268
memory, 159
addresses, 204, 209-210
determining, 204-206
examining, 212-213
retrieving, 208-209
storing in pointers, 206-207
arrays, 84
code space, 159

examining, 722
free store, 217
advantages, 218
memory allocation, 218-219
objects, 222-224
restoring, 219-221
leaks, 258-259
delete statement, 219
pointer reassignments, 221
pointers, 204-206
advantages, 216
allocating, 220
const, 227
data manipulation, 210-211
declaring, 206, 216
deleting, 220
dereferencing, 208-209, 216
indirection, 207
initializing, 206, 216
memory leaks, 221
naming, 206
null, 206
reassigning, 221
RTTI, 347
stray/dangling, 224-227
stomping on, 226
this, 224
wild, 206
RAM, 44, 159-161
registers, 159
smart pointers. *See* smart pointers
stack, 160-161, 217
clearing, 217
pulling data from, 161-162
pushing data onto, 160-162
storing data in, 44
variables, sizing, 51-53
virtual methods, 334
menu() function, 198
Meow() function, 276, 280
methods, 268
base classes, accessing, 328
base methods, calling, 320-321
bitset class, 620-623
constructors, 281
calling multiple, 355-358
defaults, 282-285
defining, 272
destructors, 282-285
file locations, declaring, 287-288
fill(), 667-668
flush(), 663
Fly(), 344
get(), 86, 655
character arrays, 658-659
character reference parameters, 656-657
with no parameters, 655-656
overloading, 660
GetAge(), 280
GetArea(), 294
getline(), 659-660
GetUpperLeft(), 294
hiding, 318-319
ignore(), 660-662
implementing, 278-281
inline, 288-290
Invariants(), 737-742
overloading, 318
overriding, 316-318
peek(), 662-663
printf(), 671-673
public accessor methods, 275-276
push back, 476
put(), 664-665
putback(), 662-663
SetAge(), 280-281
setf(), 668-669, 671
strcpy(), 87-88
strncpy(), 87-88
virtual, 322-328
calling multiple, 324-326
copy constructors, 331-334
destructors, 330-331
memory costs, 334
slicing, 328-330
v-pointers, 326-327
v-tables, 326-327
width(), 666-667
write(), 665-666
minus signs (-), 101-103
mixins (capabilities classes), 368
modifying algorithms, 572
modulus (%) operator, 99
multidimensional arrays, 80
declaring, 80
initializing, 81-82
multiline comment styles, 35
multimap, 533
elements
deleting, 540, 543
inserting, 535-538
searching, 538-540
instantiating, 535
sorting, 543, 546-547
multiple base classes
ambiguity resolution, 358-359
constructors, 355-358
objects, 354-355
multiple exceptions, 703-706
multiple inheritance, 351-353
ambiguity resolution, 358-359
constructors, 355-358
declaring, 354

How can we make this index more useful? Email us at indexes@samspublishing.com

limitations, 367
objects, 354-355
shared base classes, 359-363
virtual inheritance, 363-367
virtual methods, 354
multiple initialization for loops, 183-184
multiple input (cin object), 651-654
multiple parameters, 437-438
multiple values
functions, 242-244
returning
pointers, 242-244
references, 244-246
multiple variables, 53
multiset, 513
advantages of, 529
elements
deleting, 519-528
inserting, 515-517
searching, 517-518
objects, instantiating, 514-515
mutating algorithms, 571-573
myFunc() function, 137

N

namespaces
designating
namespace keyword, 34-35
std[::] notation, 32
using keyword, 33-34
keywords, 34-35
naming
arrays, 213-215
classes, 269-270
conventions
capitalization, 753
identifiers, 752-753
spelling, 753
counting variables, 188
exceptions, 709-716
filename extensions
.c, 16-17
.cpp, 16-17, 287
.h, 288
.hp, 288
.hpp, 288
.obj, 18
pointers, 206
references, 232
variables, 48-50
case-sensitivity, 49
reserved words, 50-51, 773-774
NCITS (National Committee for Information Technology Standards) Standard, 15
need for casting, 416
\n escape code, 62
nesting
if statements, 113
braces ({ }), 115-118
example, 114-117
loops, 187-188
parentheses, 104
Net (.Net) platform, 756, 758
new operator, 254
new statement, 218-219
newline code (\n), 30-31
newline delimiter, 86
newline escape characters (\n), 62
newsgroups, 756
nonexistent objects, referencing, 256-257
nonmutating algorithms, 570-571
not equal operator (!=), 106
NOT operators, logical (!), 119
notation, Hungarian, 50
null character, 84, 651
null pointers, 206, 237
compared to stray pointers, 227
null references, 237
null statements for loops, 184-187
numbers
base 10, 763-764
converting to base 6, 766
converting to base 7, 765-766
base 7, 765
base 8, 764
binary, 767-768
advantages, 767
converting to, 766
counting while (true) loops, 176-177
Fibonacci series, 190-192
recursion, 154-158
hexadecimal, 768-772
nybbles, 767-768

O

object-oriented programming (OOP), 12, 266-267
data hiding, 12
encapsulation, 12-13
inheritance, 13
polymorphism, 13
functions, 147-150
objects, 17-18, 709. ***See also*** **exceptions**
Cat, initializing, 283-285
cin
ember functions, 654-657
return values, 654
compared to classes, 270-271
cout, 30-32
example, 30-31
passing values to, 32
defining, 270, 277
derived, accessing, 307-308

endl, 32
exceptions, naming, 709-716
free store objects
creating, 222
deleting, 222-224
functions, 553-554
applying, 554-565
overview of, 554
inheritance
casting down, 347-350
multiple, 354-355
initializing constructor methods, 281-282
passing
data slicing, 328-330
references to, 252-253
referencing
nonexistent objects, 257
objects on heap, 258-259
SimpleCat, 249
standard I/O objects, 647-648
cerr, 648
cin, 648-663
clog, 648
cout, 648, 663-671
states, 668
values, assigning, 271

oct flag, 669

octal notation, escape characters, 62

ofstream objects, 675
arguments, 677-678
condition states, 675
default behavior, 677-680
opening files, 675-677

obj filename extension, 18

OOP (object-oriented programming), 12, 266-267
data hiding, 12
encapsulation, 12-13
inheritance, 13
polymorphism, 13

opening files for input/output, 675-677

operator() function, 411-412

operators, 97
address of (&), 204-206, 233-234
assignment (=), 53, 97
binary
addition/subtraction, 397-399
assignment, 399-401
types, 396
bitset classes, 619-620
bitwise, 744
AND, 744
complement, 745
exclusive OR, 745
OR, 745
casting, 417
const cast, 422-423
defined, 416
dynamic cast, 419-421
need for, 416
reinterpret cast, 421-422
static cast, 418
troubleshooting, 423-424
unpopular styles, 417
comparison, overloading, 401-405
concatenation, 733
conditional (?[:]), 121-123
conversion, programming, 394-395
decrement (—), 101
postfix, 102
prefix, 101-103
dot (.), 277
equality, 402
extraction ([<<]), 649
increment (++), 101
postfix, 102
prefix, 101-103
indirection (*), 208-209
syntax, 255
inequality, 402
logical, 118
AND (&&), 118
NOT (!), 119
OR (||), 119
order of evaluation, 119
mathematical, 97
modulus (%), 99
self-assigned, 100
subtraction (-), 98-99
new, 254
overloading, 405-408
postfix increment, 390
precedence, 103, 120, 775-776
redirection, 648
redirection (), 30
reference (&), 232, 255-256
relational, 105-109
sizeof, 53
subscript, 409-411
symbols, 386
true/false operations, 120-121
types, 386
unary, 387
customizing, 394-395
programming deference, 391-393
programming increment/decrement, 387-390
types, 387
vector bool, 624-625

optimization
performance, 152
pointers, 630-631

OR operators
bitwise, 745
logical (||), 119

ostream class, 647

out-of-scope references, returning, 256-257
output, 663
- flushing, 663
- formatting, 673-674
  - fill characters, 667-668
  - flags, 668-671
  - width, 666-667
- iterators, 452
- output devices, writing to, 664-665
- redirection (/r) operator, 30
- streams, 644. *See also* streams

overloading
- binary operators, 396
- compared to overriding, 318
- comparison operators, 401-405
- constructors
  - derived classes, 311-315
- functions, 147-150
- operators, 405-408
  - that cannot be redefined, 412-413
  - unary, 387

overriding operators, 316-320

P

parameters
- command-line processing, 682-686
- defaults, 145-147
- get() method, 656
- macros, 429
- passing by value, 137-138
- templates, declaring, 437-438

parameters (functions), 39, 129, 141-142
parentheses ()
- macro syntax, 430-431
- nesting, 104

partitioning
- algorithms, 573
- RAM, 159-161
- ranges, 593-595

passing
- arguments
  - by reference, 237-249
  - by value, 238-239
  - to base constructors, 311-315
- const pointers, 249-252
- exceptions, 713-714, 716
- objects, data slicing, 328-330
- parameters by value, 137-138
- references to objects, 252-253
- values to cout, 32

peek() method, 662-663
percolating shared functions, 346-347
performance optimization, 152
periods (.), dot operator, 277
persistence, local variables, 217
pipe character (|), 119
piping, 648
plus signs (+), 101-103
Point class, declaring, 291-292
pointers, 204-210
- advantages, 216
- allocating, 220
- combining with references, 255
- compared to references, 254
- const, 227
  - declaring, 227
  - passing, 249-252
- current values, printing, 431, 742-749
- data manipulation, 210-211
- declaring, 206, 216
- deleting, 220
- dereferencing, 208-209, 216
- indirection, 207
- initializing, 206, 216
- memory addresses
  - assigning, 206-207
  - examining, 212-213
  - retrieving, 208-209
- memory leaks, 221
- naming, 206
- null, 206, 227, 237
- passing by reference, 239-240
- pointer arrays, 215
- reassigning, 221
- returning multiple values, 242-244
- RTTI, 347
- smart, 629
  - applying conventional, 630
  - auto_ptr, 638-640
  - Copy on Write, 635
  - deep copy, 633-635
  - destructive copy, 636-637
  - implementing, 631-632
  - libraries, 640
  - optimizing, 630-631
  - overview of, 630
  - reference counting, 635-636
  - reference-linked, 636
  - types, 632-633
- stomping on, 226
- stray/dangling, 224-226
  - cautions, 226
  - compared to null pointers, 227
  - creating, 225
- this, 224
- to arrays, 213-215
- v-pointers, 326-327
- wild, 206

polymorphism, 13, 322, 343
- functions, 147-150

pop_back function, 483
postfix increment operators, 390
postfix operators, 102
pound symbol (#), 28
precedence of operators, 103, 120, 775-776
predefined macros, 733
predicates, 554-559
  binary, 554, 563-565
  unary, 559-561
prefix operators, 101-103
preprocessors, 428
  class invariants, 737-742
  commands
    #define, 428, 728-730
    #else, 730-731
    #ifndef, 730
  inclusion guards, 731-732
  inline functions, 432-433
  interim values, printing, 431, 742-749
  macros, 429-430
    assert(), 734-737
    compared to functions, 432
    compared to templates, 432
    defining, 429-430
    parameters, 429
    parentheses (), 430-431
    predefined, 733
    syntax, 429-430
  string manipulation, 733
  substitutions
    constants, 729
    strings, 428, 728
  tests, 729-730
printf() function, 671-673
printing
  characters, 60-61
  interim values, 431, 742-749
  printf() function, 671-673
  to screens, 30-32
priority queues, 610-614
private classes, 272-274
private data members, 275, 306-307
  accessing, 272
  security, 277
private inheritance, 335
  aggregation, 337-338
  applying, 335-337
procedures, 11, 266
processing elements, 581-583
programming
  comments, 35
    /* (C-style), 35
    // (C++-style), 35
    applying, 36-37
    troubleshooting, 37
  conversion operators, 394-395
  development
    cycles, 18
    environments, 16
  executable files, creating, 18
  levels of abstraction, 159
  loops, 168. *See also* loops
  object files, 17-18
  object-oriented, 11-12
    data hiding, 12
    encapsulation, 12-13
    inheritance, 13
    polymorphism, 13
  OPP. *See* OPP
  programs
    branching, 162
    design, 15-16
    structure, 28-30
  resources, 756
  structured, 11-12
  style guidelines
    access labels, 754-755
    assert() macro, 755
    capitalization, 753
    class definitions, 755
    comments, 754
    const statement, 755
    identifier names, 752-753
    include files, 755
    readability of code, 752
    spelling, 753
  templates, 440-441
    connecting, 453-454
    STL containers, 448-451
    STL iterators, 451-452
  troubleshooting, 10
  unary operators, 387-393
programs
  branching, 162
  comments, 35
    /* (C-style), 35
    // (C++-style), 35
    applying, 36-37
    troubleshooting, 37
  compilers, 22
    overview, 9
    troubleshooting, 23
  compiling errors, 23-24
  debugging
    assert() macro, 735-736
    printing interim values, 431, 742-749
  defined, 10
  designing, 15-16
  Hello World, 19
    compiling, 20-22
    creating, 22-23
    source code, 20, 28
    testing, 22-23
  interpreters, 9
  linkers, 9

listings. *See* code listings
preprocessors, 28
structure of, 28, 30
include statements, 28
main() function, 29
pound (#) symbol, 28
properties, STL container classes, 449
protected data members, 306-307
protected keyword, 306
prototypes
defined, 129
functions, 130-131
return types, 131
public accessor methods, 275-276
public classes, 272-274
public keyword, 280
pulling data from stack, 162
pure virtual functions, 372-377
push back method, 476
pushing data onto stack, 162
put() method, 664-665
putback() method, 662-663

Q

question mark (?), 62
queues, 602-607
quotation marks ("), 733

R

r-values, 97
RAM (random access memory), 44, 159-161. *See also* memory
random access iterators, 452
ranges
elements, processing, 581-583
partitioning, 593-595
searching, 576-578
transforming, 583-585
values, replacing, 589-590
readability of code, 752
reading data in exceptions, 709
reassigning
pointers, 221
references, 235
Rect.cpp (code listing), 292-293
Rectangle class, declaring, 292-294
recursion
Fibonacci series example, 156-158
functions, 153-154, 158
stop conditions, 154-155
redirect output command ([<], 648
redirection, 20, 30, 648-649
reference operator (&), 232, 255-256
references, 232-233
combining with pointers, 255
compared to pointers, 254
const pointers, 249-252
counting, 635-636
creating, 232-233
errors
nonexistent objects, 256-257
referencing objects on heap, 258-259
initializing, 233
linking, 636
naming, 232
null, 237
objects
nonexistent objects, 257
objects on heap, 258-259
passing
by reference, 237-240, 246-249
to objects, 252-253
reassigning, 235
returning multiple values, 244-246
swap() function, 240-242
target addresses
assigning, 235-236
returning, 233-235
refilling buffers, 646
reinterpret cast, applying, 421-422
relational operators, 105-106
branching, 108-109
precedence, 120
relationships among classes, 303
removal algorithms, 572
remove function, 585-588
replacement algorithms, 572
replacing elements, 589-590
requirements, updating, 10
\r escape code, 62
reserved words, 50-51, 773-774
resources, 756
restoring memory to free space, 219-221
retrieving data in exceptions, 709
return statements, 39, 142-144
return types (functions), 131
return values
cin, 654
functions, 38-39, 129, 142-144
multiple, 142-144

returning
multiple values
pointers, 242-244
references, 244-246
out-of-scope references, 256-257
reusing source code, inheritance, 13
reversing
elements, linked lists, 500-501
STL string classes, 468-469
right flag, 669
rot, code, 720
rSomeRef statement, 232
RTTI (Run Time Type Identification), 347
run-time binding, 326
runtime type identification, 419-421

S

safety, template and type, 437
schematic representations, 204
scientific flag, 669
scope
for loops, 189
variables, 134-137
screens, printing to, 30-32
searching
algorithms, 570
elements, 573-576
map and multimap, 538-540
set or multiset, 517-518
ranges, 576-578
security classes, 277
selecting containers, 449, 451
self-assigned addition (+=) operator, 100
self-assigned operators, 100
semicolons (;), 94
in if statements, 109-110
sequential containers, 448
set, 513
advantages of, 529
elements
deleting, 519-528
inserting, 515-517
searching, 517-518
objects, instantiating, 514-515
SetAge() function, 280-281
setf() method, 668-671
setw manipulator, 669
Shape classes, 369-371
shared base classes, inheritance, 359-363
short data type, 56-57
short int data type, 47
short integers, 46
showbase flag, 669
showpoint flag, 669
signed integers, 46, 58-59
Simonyi, Charles, 50
SimpleCat object, 249
single character input, 655
single inheritance, troubleshooting, 344-346
single quote (‘), 62
sizeof operator, 53
sizes
of functions, 141
of variables, 45-48
sizing
arrays, 78-79
variables, 51-53
vectors, 484-485
slashes (/), 35
slicing virtual methods, 328-330
smart pointers, 629
auto_ptr, 638-640
conventional, applying, 630
Copy on Write, 635
deep copy, 633-635
destructive copy, 636-637
implementing, 631-632
libraries, 640
optimizing, 630-631
overview of, 630
reference counting, 635-636
reference-linked, 636
types of, 632-633
solving the nth Fibonacci number (listing), 191
sorting
algorithms, 572
collections, 591-593
elements, 502-511
map and multimap templates, 543-547
source code, 8. *See also* code listings
compiling, 17-18
reusing, 13
source files, 16
special printing characters, formatting, 61-62
specialization, templates, 437
stacks, 217, 602
clearing, 217
data
adding, 160-161
retrieving, 161
instantiating, 603-604
member functions, 604-606
memory, 160-161
pulling data from, 162
pushing data onto, 162
standard I/O objects, 647-648
cerr, 648
cin, 648-650
get() method, 655-659
getline() method, 659-660
ignore() method, 660-662

multiple input, 651-654
peek() method, 662-663
putback() method, 662-663
strings, 651
clog, 648
cout, 648, 663-671
fill characters, 667-668
flags, 668-671
flush() method, 663
output width, 666-667
put() method, 664-665
write() method, 665-666

standard libraries, 21

standard namespaces
namespace keyword, 34-35
std[:] notation, 32
using keyword, 33-34

Standard Template Library (STL), 440-441
adaptive containers, 601
instantiating queues, 607
instantiating stacks, 603-604
priority queues, 610-614
queue member functions, 607-610
queues, 602-603
stack member functions, 604-606
stacks, 602
algorithms, 569
classification of, 570-573
copy and remove operations, 585-588
counting and finding elements, 573-576
initializing elements, 578-581
inserting elements, 595-597
overview of, 570
partitioning ranges, 593-595
processing elements, 581-583
replacing elements, 589-590
searching ranges, 576-578
sorting collections, 591-593
transforming ranges, 583-585
bit flags, 617
bitset class, 618-623
vector bool, 623-625
connecting, 453-454
container classes, 513
advantages of set and multiset, 529
deleting elements, 519-528
inserting elements, 515-517
instantiating set objects, 514-515
searching elements, 517-518
containers, 448
associative, 448-449
selecting, 449-451
sequential, 448
iterators, 451-452
linked lists, 491
characteristics of, 492
deleting elements, 497-500
inserting elements, 493-497
instantiating, 492-493
reversing elements, 500-501
sorting elements, 502-511
map and multimap, 533
customizing sort predicates, 543-547
deleting elements, 540-543
inserting elements, 535-538
instantiating, 535
searching elements, 538-540

state flags, 666-668

statements, 94
blocks, 94-95
catch, 693
try, 693-696
break, 173-175
catch, 693-696, 703-706
class, 269, 277-278
compound, 94-95
const, 63-64, 286, 755
continue, 173-175
default, 194
#define
constant substitutions, 729
string substitutions, 428, 728
tests, 729-730
define, 63
delete, 219-221
do...while, 179-180
#else, 730-731
expressions, 95
evaluating, 96
nesting parentheses, 104
for loops, 180-181
in functions, 141
goto, 168
disadvantages, 169
example, 168-169
syntax, 169
if, 107, 109
branching, 108-109
else keyword, 111
indentation styles, 110-111
nesting, 113-118

semicolon notation, 109-110
syntax, 112-113
include, 28
initialization, 182
inline, 151, 288-289
new, 218-219
null for loops, 184-187
protected, 306
return, 39, 142-144
struct, 295-296
switch, 192-193
case values, 192
example, 193-194
forever loops, 195-198
guidelines, 198
syntax, 192-194
syntax, 94
try, 693-696
watch, 442
while, 169
complex loops, 171-172
simple example, 169-170
syntax, 171
whitespace, 94

states (objects), 668

static cast, applying, 418

STL (Standard Template Library), 440-441
adaptive containers, 601
instantiating queues, 607
instantiating stacks, 603-604
priority queues, 610-614
queue member functions, 607-610
queues, 602-603
stack member functions, 604-606
stacks, 602
algorithms, 569
classification of, 570-573
copy and remove operations, 585-588
counting and finding elements, 573-576
initializing elements, 578-581
inserting elements, 595-597
overview of, 570
partitioning ranges, 593-595
processing elements, 581-583
replacing elements, 589-590
searching ranges, 576-578
sorting collections, 591-593
transforming ranges, 583-585
bit flags, 617
bitset class, 618-623
vector bool, 623-625
connecting, 453-454
container classes, 513
advantages of set and multiset, 529
deleting elements, 519-528
inserting elements, 515-517
instantiating set objects, 514-515
searching elements, 517-518
containers, 448
associative, 448-449
selecting, 449-451
sequential, 448
iterators, 451-452
linked lists, 491
characteristics of, 492
deleting elements, 497-500
inserting elements, 493-497
instantiating, 492-493
reversing elements, 500-501
sorting elements, 502-511
map and multimap, 533
customizing sort predicates, 543-547
deleting elements, 540-543
inserting elements, 535-538
instantiating, 535
searching elements, 538-540

STL deque class, 486-488

STL dynamic array class
need for, 473
vectors, 474
accessing elements, 480-482
deleting elements, 482-484
inserting elements, 476-479
instantiating, 474-475
size and capacity, 484-485

STL string class, 457
accessing, 461-463
applying, 459
case conversion, 469-470
concatenation, 463-464
find member function, 464-466
instantiating, 459-461
need for, 458-459
reversing, 468-469

template-based implementation, 471
truncating, 467-468
stomping on pointers, 226
stop conditions, 154-155
storing
data in memory, 44
memory addresses in pointers, 206-207
stray pointers, 224-226
cautions, 226
compared to null pointers, 227
creating, 225
strcpy() function, 88
strcpy() method, 87-88
streambuf class, 647
streams, 643
buffers, 645-647
compared to printf() function, 671-673
encapsulation, 644
ofstream class, 675
condition states, 675
default behavior, 677-680
opening files, 675-677
overview, 644
redirection, 648-649
standard I/O objects, 647-648
cerr, 648
cin, 648-663
clog, 648
cout, 648, 663-671
String classes, 89-90
strings
char arrays, 84-86
concatenating, 733
copying
strcpy(), 87-88
strncpy(), 88
current values, 431, 742-749
defined, 30
null character, 651
placing in quotes, 733
STL string class, 457
accessing, 461-463
applying, 459
case conversion, 469-470
concatenation, 463-464
find member function, 464-466
instantiating, 459-461
need for, 458-459
reversing, 468-469
template-based implementation, 471
truncating, 467-468
stringizing, 733
substitutions, 428, 728
testing #define statements, 729-730
troubleshooting, 651-654
strncpy() function, 88
strncpy() method, 87-88
Stroustrup, Bjarne, 14
structs
keywords, 295-296
types, creating, 268
structured programming
disadvantages, 11-12
overview, 11
structures, 266
compared to classes, 295
declaring, 295-296
style guidelines, 750
access labels, 754-755
assert() macro, 755
braces, 750
capitalization, 753
class definitions, 755
comments, 754
const statement, 755
identifier names, 752-753
include files, 755
indents, 750-752
long lines, 751
readability of code, 752
spelling, 753
styles, unpopular casting, 417
subclasses, 291-295
subscript operators, 409-411
subtraction (-) operator, 98-99, 397-399
supersets, 302
swap() function, 138
pointers, 239-240
references, 240-242
switch statements, 192-193
case values, 192
example, 193-194
forever loops, 195-198
guidelines, 198
indenting, 751-752
syntax, 192-194
symbolic constants, 62-64
symbols, operators, 386

T

\t (tab code), 32
tables
v-tables
(virtual function tables), 326-327
tabs
escape characters (\t), 62
escape code (\t), 32
target addresses
assigning, 235-236
references, 233
returning, 233-235
temperatures, Fahrenheit/Celsius conversions, 135
templates
adaptive container, 601
instantiating queues, 607
instantiating stacks, 603-604

priority queues, 610-614
queue member functions, 607-610
queues, 602-603
stack member functions, 604-606
stacks, 602
algorithms, 569
classification of, 570-573
copy and remove operations, 585-588
counting and finding elements, 573-576
initializing elements, 578-581
inserting elements, 595-597
overview of, 570
partitioning ranges, 593-595
processing elements, 581-583
replacing elements, 589-590
searching ranges, 576-578
sorting collections, 591-593
transforming ranges, 583-585
applying, 440-441
bit flags, 617
bitset class, 618-623
vector bool, 623-625
classes, 436-439
compared to macros, 432
container classes, 513
advantages of set and multiset, 529
deleting elements, 519-528
inserting elements, 515-517
instantiating set, 514-515
map and multimap, 533
searching elements, 517-518
default parameters, declaring with, 438
defining, 434-435
exceptions, 716-719
instantiating, 437
linked lists, 491
characteristics of, 492
deleting elements, 497-500
inserting elements, 493-497
instantiating, 492-493
reversing elements, 500-501
sorting elements, 502-511
multiple parameters, declaring with, 437-438
overview of, 434
specialization, 437
STL. *See* STL
types
declaring, 436
safety, 437

ternary operator (?[:]), 121-123

\t escape code, 62

testing
Hello World programs, 22-23
strings, 729-730

text
built-in editors, 24
comments, 35. *See also* comments
comparing
to word processors, 24
to binary files, 680-682
strings, 30

this pointer, 224

throwing exceptions, 696-702

tilde (~), 282
complement operator, 745

trailing zeros, displaying, 668

transforming, 583-585

troubleshooting, 693
bugs, 690
casting operators, 423-424
code rot, 720
compile-time errors, 287
compilers, 23
conventional pointers, 630
exceptions, 691-692
catching, 702-706
class hierarchies, 706-709
hierarchies, 707
multiple, 703-706
programming tips, 719
sample program, 691-692
templates, 716-719
throwing, 696-702
try...catch blocks, 693-696
global variables, 140
inheritance
multiple, 367
single, 344-346
programming, 10
strings, 651-654
subtraction, 98-99

true/false operations, 120-121

truncating STL string classes, 467-468

truth, expressions, 105-107

try blocks, 693-696, 702

two-dimensional arrays, declaring, 83

type definition, 55-56

typedef keyword, 55-56

types, 47
of comments, 35
creating, 267
derived, 13

of smart pointers, 632-633
auto_ptr, 638-640
Copy on Write, 635
deep copy, 633-635
destructive copy, 636-637
libraries, 640
reference counting, 635-636
reference-linked, 636
of STL algorithms, 570
mutating, 571-573
nonmutating, 570-571
of variables, 47-53
operators, 386
binary, 396-401
programming deference, 391-393
programming unary increment/decrement, 387-390
subscript, 409-411
unary, 387
runtime identification, 419-421
templates
declaring, 436
safety, 437

U

unary functions, 554-559
unary operators, 387
customizing, 394-395
programming, 387-393
types, 387
unary predicates, 559-561
uninitialized buffers, 85
uninitialized character arrays, 85
unpopular casting styles, 417
unsigned int data types, 47
unsigned integers, 46, 57-58
unsigned long int data types, 47
unsigned short int data types, 47
updating requirements, 10
Uppercase flag, 669
Usenet newsgroups, 756
using keyword, 33-34

V

v-pointers, 326-327
v-tables, 326-327
values, 137-138
assigning to variables, 53-55, 271
cin object return values, 654
concatenating, 32
containers, 578-581
enumerated constants, 64
function return values, 38-39
l-values, 97
multiple
returning with pointers, 242-244
returning with references, 244-246
passing
by reference, 237-249
by value, 238-239
to cout, 32
r-values, 97
replacing, 589-590
variables
assigning, 53-55
values, 48
void, 142
variables, 64. *See also* constants
char, 45, 59-60
character encoding, 60-61
escape characters, 61-62
sizes, 59
counting variables, 188
current values, 431, 742-749
data members, 268
data types, 47, 267
defining, 45-53
example, 54
floating-point, 47
global, 139-140
initializing, 53
integers
long, 56-57
short, 56-57
signed, 58-59
signed/unsigned, 46
sizes, 45-46
unsigned, 57-58
local, 134-135
example, 134-135
persistence, 217
scope, 136-137
memory, storing data in, 44
names, 48-49
case-sensitivity, 49
reserved words, 50-51, 773-774
naming, 49-50
overview, 44
pointers, 204-206
advantages, 216
allocating, 220
const, 227
data manipulation, 210-211
declaring, 206, 216
deleting, 220
dereferencing, 208-209, 216
indirection, 207
initializing, 206, 216
memory addresses, 212-213
memory leaks, 221
naming, 206
null, 206
reassigning, 221

RTTI, 347
stomping on, 226
stray/dangling, 224-227
this, 224
wild, 206
pointers and, 209-210
scope, 134
sizes, 48
type definition, 55-56
values
assigning, 53-55, 271
defined, 48

vectors
characteristics of, 474
elements
accessing, 480-482
deleting, 482-484
inserting, 476-479
instantiating, 474-475, 623-624
operators, 624-625
size and capacity of, 484-485

vertical bars (|), 119

vertical tab escape characters (\v), 62

\v escape code, 62

viewing vectors, 474

virtual functions
exceptions, 713-714, 716
pointers, 326-327
pure, 372-377
tables, 326-327

virtual inheritance, 363, 366
declaring, 367
example, 365-366

virtual machine. *See* VM

virtual methods, 322-328
copy constructors, 331-334
destructors, 330-331
memory costs, 334
multiple, calling, 324-326
slicing, 328-330
v-pointers, 326-327
v-tables, 326-327

VM (virtual machine), 9

void value, 142

vptr (virtual function pointer), 326-327

W

warning messages, 25

watch points, 721

watch statements, 442

while (true) loops, 176-177

while loops, 169
break statement, 173-175
complex loops, 171-172
continue statement, 173, 175
do...while
example, 178-179
syntax, 179-180
exiting, 173-175
returning to top of, 173-175
simple example, 169-170
skipping body of, 177-178
starting conditions, 180-181
syntax, 171
while (true), 176-177

whitespace in statements, 94

width() method, 666-667

wild pointers, 206

word processors, 16
compared to text editors, 24

wrapping
signed integers, 58-59
unsigned integers, 57-58

write() method, 665-666

writing
classes to files, 680-682
to output devices, 664-665
past the end of arrays, 74-76