Disabling or otherwise ignoring warnings is an extraordinarily bad idea. It's a bit like unplugging the check engine light on your car's dashboard because it bothers you. Ignoring the problem doesn't make it go away. It doesn't mean that you can always fix the problem — for example, I chose to overlook the warnings in Chapter 31 about `strstream` being deprecated — but you need to at least understand the warning. What you don't know will hurt you.

If your compiler has a Syntax Check from Hell mode, enable it.

Adopt a Clear and Consistent Coding Style

Writing your C++ code in a clear and consistent style not only enhances the readability of your program, but also it results in fewer coding mistakes. This somewhat surprising state of affairs results from the fact that our brains have only a limited amount of computing power. When you read code that is clean and neat and that follows a style you're familiar with, you spend very little brain power parsing the syntax of the C++ statements. This leaves more brain CPU power to decode what the program is trying to do and not how it's doing it.

A good coding style lets you do the following with ease:

- Differentiate between class names, object names, and function names
- Understand what the class, function, or object is used for based on its name
- Differentiate preprocessor symbols from C++ symbols (that is, `#define` objects should stand out)
- Identify blocks of C++ code at the same level (this is the result of consistent indentation)

In addition, you need to establish a standard module header format that provides information about the functions or classes in the module, the author (presumably, that's you), the date, the version, and something about the modification history.

All programmers involved in a single project should use the same coding style. A program written in a patchwork of different coding styles is confusing and looks unprofessional.

www.it-ebooks.info

Comment the Code While You Write It

You can avoid errors if you comment your code while you write it rather than wait until everything works and then go back and add comments. I can understand not taking the time to write voluminous headers and function descriptions until later, but I have never understood not writing short comments as you are coding.

Have you ever had the experience of asking someone a question, and even as you got to the end of the question you knew the answer? Somehow formulating the question forced you to organize your thoughts sufficiently so that the answer became clear.

Writing comments is like that. Formulating comments forces you to take stock of what it is you're trying to do. Short comments are enlightening, both when you read them later and as you're writing them.

Write comments like you're talking to another, knowledgeable programmer. You can assume that the reader understands the basics of the program, so please don't explain how C++ works. There's no point in writing comments that explain how a `switch` statement works unless you're relying on some obscure point of the language (like the fall-through capability of the `switch` statement).

Single-Step Every Path in the Debugger at Least Once

It may seem like an obvious statement, but I'll say it anyway: As a programmer, it's important that you understand what your program is doing. It isn't sufficient that the program outputs the expected value. You need to understand everything your program is doing. Nothing gives you a better feel for what's going on under the hood than single-stepping the program with a good debugger (like the one that comes with Code::Blocks).

Beyond that, as you debug a program, you sometimes need raw material to figure out some bizarre behavior. Nothing gives you that material better than single-stepping through each function as it comes into service.

Finally, when a function is finished and ready to be added to the program, every logical path needs to be traveled at least once. Bugs are much easier to find when the function is examined by itself rather than after it has been thrown into the pot with the rest of the functions — and your attention has gone on to new programming challenges.

www.it-ebooks.info

Limit the Visibility

Limiting the visibility of class internals to the outside world is a cornerstone of object-oriented programming. The class should be responsible for its internal state — if something gets screwed up in the class, then it's the class programmer's fault. The application programmer should worry about solving the problem at hand.

Specifically, limited visibility means that data members should not be accessible outside of the class — that is, they should be marked as protected. In addition, member functions that the application software does not need to know about should also be marked protected. Don't expose any more of the class internals than necessary to get the job done.

A related rule is that public member functions should trust application code as little as possible, even if the class programmer and the application programmer are the same person. The class programmer should act like it's a fact that the application programmer is a felonious hacker; if your programmer is accessible over the Internet, all too often this assumption is true.

Keep Track of Heap Memory

Losing track of heap memory is the most common source of fatal errors in programs that have been released into the field and, at the same time, the hardest problem to track down and remove (because this class of error is so hard to find and remove, it's prevalent in programs that you buy). You may have to run a program for hours before problems start to arise (depending upon how big the memory leak is).

As a general rule, programmers should always allocate and release heap memory at the same "level." If a member function `MyClass::create()` allocates a block of heap memory and returns it to the caller, then there should be a member `MyClass::release()` that returns it to the heap. Specifically, `MyClass::create()` should not require the parent function to release the memory.

If at all possible, `MyClass` should keep track of such memory pointers on its own and delete them in the destructor.

Certainly, this doesn't avoid all memory problems, but it does reduce their prevalence somewhat.

www.it-ebooks.info

Zero Out Pointers after Deleting What They Point To

Sort of a corollary to the warning in the preceding section is to make sure that you zero out pointers after they are no longer valid. The reasons for this become clear with experience: you can often continue to use a memory block that has been returned to the heap and not even know it. A program might run fine 99 percent of the time, making it very difficult to find the 1 percent of cases where the block gets reallocated and the program doesn't work.

If you zero out pointers that are no longer valid and you attempt to use them to store a value (you can't store anything at or near location 0), your program will crash immediately. Crashing sounds bad, but it's not. The problem is there; it's merely a question of whether you find it or not before putting it into production.

It's like finding a tumor at an early stage in an x-ray. Finding a tumor early when it's easy to treat is a good thing. Given that the tumor is there either way, not finding it is much worse.

Use Exceptions to Handle Errors

The exception mechanism in C++ is designed to handle errors conveniently and efficiently. In general, you should throw an error indicator rather than return an error flag. The resulting code is easier to write, read, and maintain. Besides, other programmers have come to expect it, and you wouldn't want to disappoint them, would you?

Having said that, limit your use of exceptions to true errors. It is not necessary to throw an exception from a function that returns a "didn't work" indicator if this is a part of everyday life for that function. Consider a function `lcd()` that returns the least common denominator of its two arguments. That function will not return any values when presented with two mutually prime numbers. This is not an error and should not result in an exception.

Declare Destructors Virtual

Don't forget to create a destructor for your class if the constructor allocates resources such as heap memory that need to be returned when the object

www.it-ebooks.info

reaches its ultimate demise. This rule is pretty easy to teach. What's a little harder for students to remember is this: Having created a destructor, don't forget to declare it virtual.

"But," you say, "My class doesn't inherit from anything, and it's not subclassed by another class." Yes, but it *could* become a base class in the future. Unless you have some good reason for not declaring the destructor virtual, then do so when you first create the class. (See Chapter 29 for a detailed discussion of virtual destructors.)

Provide a Copy Constructor and Overloaded Assignment Operator

Here's another rule to live by: If your class needs a destructor, it almost surely needs a copy constructor and an overloaded assignment operator. If your constructor allocates resources such as heap memory, the default copy constructor and assignment operator will do nothing but create havoc by generating multiple pointers to the same resources. When the destructor for one of these objects is invoked, it will restore the assets. When the destructor for the other copy comes along, it will screw things up.

If you are too lazy or too confused or you just don't need a copy constructor and assignment operator, then declare "do nothing" versions, but make them protected so that application software doesn't try to invoke them by accident. See Chapter 30 for more details. (The 2009 C++ standard allows you to delete both the default copy constructor and assignment operator, but declaring them protected works almost as well.)

www.it-ebooks.info

Chapter 34

Ten Features Not Covered in This Book

In This Chapter

- The `goto` command
- The ternary operator
- Binary logic
- Enumerated types
- Namespaces
- Pure virtual functions
- The `string` class
- Multiple inheritance
- Templates and the Standard Template Library
- The 2009 C++ Standard

The C++ language contains so many features that covering every one in a single book — especially a book intended for beginning programmers — is impossible. Fortunately, you don't need to master all of the features of the language in order to write big, real-world programs.

Nevertheless, you may want to look ahead at some of the features that didn't make the cut for this beginner's book, just in case you see them in other people's programs.

The goto Command

This command goes all the way back to C, the progenitor of C++. In principle, using this command is easy. You can place `goto label;` anywhere you

www.it-ebooks.info

want. When C++ comes across this command, control passes immediately to the label, as demonstrated in this code snippet:

```
  for(;;)
  {
      if (conditional expression)
      {
          goto outahere;
      }
      // ...whatever you want...
  }
outahere:
  // ...program continues here...
```

In practice, however, goto introduces a lot of ways to screw up — many more than I can go into here. In any case, it didn't take long before programmers noticed that the two most common uses of the goto were to exit loops and to go to the next case within a loop. The C Standards Committee introduced break and continue and almost completely removed the need for the goto command. I can say that I've been programming in C and C++ for almost 20 years, and I've never had an application for a goto that I couldn't handle in some other way more clearly.

The Ternary Operator

The ternary operator is an operator unique to C and C++. It works as follows:

```
int n = (conditional) ? expression1 : expression2;
```

The ? operator first evaluates the conditional. If the condition is true, then the value of the expression is equal to the value of expression1; otherwise, it's equal to the value of expression2.

For example, you could implement a maximum() function as follows:

```
int max(int n1, int n2)
{
    return (n1 > n2) ? n1 : n2;
}
```

The ternary operator can be applied to any type of numeric but cannot be overloaded. The ternary operator is truly an expression and not a control statement like an if.

www.it-ebooks.info

Binary Logic

I chose to skip entirely the topic of binary arithmetic. Some readers will consider this scandalous. After all, how can you talk about programming without getting down to ones and zeros? It's not that I don't consider the topic worthwhile — it's just that I find explaining the topic properly takes many pages of text and leaves readers somewhat confused, when in practice it's rarely used. Google the topic once you feel comfortable with the basics of C++ programming.

Enumerated Types

This is a topic that barely missed the cut for inclusion in the book. The simple idea is that you can define constants and let C++ assign them values, as shown here:

```
enum Colors {BLACK, BLUE, GREEN, YELLOW, RED};
Colors myColor = BLACK;
```

The problem with enumerated types lies in the implementation: Rather than create a true type, C++ uses integers. In this case, `BLACK` is assigned the value `0`, `BLUE` is assigned `1`, `GREEN` `2`, and so on. This leads to special cases that make the topic not worth the trouble.

The 2009 Standard Library for C++ "fixed" this problem by creating true enumerated types, but it didn't do away with the integer version in order to retain compatibility with existing programs. The result is even more confusing than before.

Namespaces

It's possible to give different entities in two different libraries the same name. For example, the `grade()` function within the Student library probably assigns a grade, whereas the `grade()` function within the Civil Engineering library might set the slope on the side of a hill. To avoid this problem, C++ allows the programmer to place her code in a separate namespace. Thus, the grade within the `Student` namespace is different from the grade within `CivilEngineering`.

www.it-ebooks.info

The namespace is above and beyond the class name. The `grade()` member function of the class `BullDozer` in the `CivilEngineering` namespace has the extended name `CivilEngineering::BullDozer::grade()`.

All library objects and functions are in the namespace `std`. The statement at the beginning of the program template `using namespace std;` says if you don't see the specified object in the default namespace, then go look in `std`. Without this I would have to include the namespace explicitly, as in the following snippet:

```
std::cout << "Hello, world!" << std::endl;
```

Pure Virtual Functions

You saw how to declare functions virtual in Chapter 29. What I didn't mention there is that you don't have to define a function declared virtual. Such an undefined function is known as a *pure virtual member function*. However, things get complicated. For example, a class with one or more pure virtual functions is said to be abstract and cannot be used to create an object (see what I mean?). Tackle this subject after you feel comfortable with virtual functions and late binding.

The string Class

This is another topic that barely missed the cut. Most languages include a `string` class as an intrinsic type for handling strings of characters easily. In theory, the string class should do the same for C++. In practice, however, it's not that simple. Because `string` is not an intrinsic type, the error messages that the compiler generates when something goes wrong are more like those associated with user-defined classes. For a beginner, these messages can be very difficult to interpret.

It's actually worse than I'm describing here — `string` isn't even a class. It's an instance of a template class. The error messages can be breathtaking.

www.it-ebooks.info

Multiple Inheritance

I describe how to base one class on another using inheritance in Chapter 28. What I didn't mention there is that one class can actually extend more than one base class. This sounds simple but can get quite complicated when the two base classes contain member functions with the same name. Or worse, when both base classes are themselves subclasses of some common class. In fact, there are so many problems that arise that C++ is the only C-like language that supports multiple inheritance. Java and C#, both languages derived from C++, decided to drop support for multiple inheritance. I recommend that beginning programmers avoid the subject.

Templates and the Standard Template Library

The makers of C++ noticed how similar functions like the following are:

```
int max(int n1, int n2)
{
    if (n1 > n2)
    {
        return n1;
    }
    return n2;
}
double max(double n1, double n2)
{
    if (n1 > n2)
    {
        return n1;
    }
    return n2;
}
char max(char n1, char n2)
{
    if (n1 > n2)
    {
        return n1;
    }
    return n2;
}
```

www.it-ebooks.info

I can almost imagine the scene: "Wouldn't it be cool," one said to another, "if we could replace the type with a pseudo-type T that we could define at compile time?" Before you knew it, templates were a part of C++:

```
template <class T> T max(T t1, T t2)
{
    if (t1 > t2)
    {
        return t1;
    }
    return t2;
}
```

Now the programmer can create a `max(int, int)` by replacing `T` with `int` and compiling the result, create a `max(double, double)` by replacing `T` with `double`, and so forth. The Standards Committee even released an entire library of classes, known as the Standard Template Library (STL for short), based upon template classes.

For a beginner, however, the subject of template classes starts to get syntactically very complicated. In addition, the errors that the compiler generates when you get a template instantiation wrong are bewildering to an expert, never mind a beginner. This is definitely a topic that needs to wait until you feel comfortable with the basic language.

The 2009 C++ Standard

The C++ Standard was released and agreed to in the late 1990s. Things changed relatively little in the ensuing years, but the demand for additions to the language grew until finally the 2009 Standard was released for comment in late 2008. The problem with this standard is that it introduces a lot of new features for which there seems to be very little demand. The standard is more than 1,400 pages. (Admittedly, it includes a lot of very dry, very repetitive library definitions, but even so, that C++ is not a small language any more.) As of this writing (early 2010), no compilers implement the full 2009 standard.

www.it-ebooks.info

Appendix

About the CD

In This Appendix

- System requirements
- Using the CD with Windows, Linux, and Mac
- What you'll find on the CD
- Troubleshooting

This section describes the CD-ROM enclosed in the back of *Beginning Programs with C++ for Dummies*. All readers will appreciate the source code to the programs that appear in the book — using this code can save you a lot of typing. In addition, 32-bit Windows users will welcome the Code::Blocks C++ development environment coupled with the GNU C++ compiler ready to be installed on Windows 2000, Windows XP, Windows Vista, or Windows 7. (Macintosh, Linux, and 64-bit Windows users can download Code::Blocks from `www.codeblocks.org`.)

System Requirements

Make sure that your computer meets the minimum system requirements shown in the following list. If your computer doesn't match up to most of these requirements, you may have problems using the software and files on the CD. For the latest and greatest information, please refer to the ReadMe file located at the root of the CD-ROM.

- A PC running Microsoft Windows or Linux with kernel 2.4 or later
- A Macintosh running Apple OS X or later
- An Internet connection (only required for downloading versions of Code::Blocks for Macintosh or Linux)
- A CD-ROM drive

If you need more information on the basics, check out these books published by Wiley Publishing, Inc.: *PCs For Dummies* by Dan Gookin; *Macs For Dummies* by Edward C. Baig; *iMacs For Dummies* by Mark L. Chambers; and *Windows XP*

www.it-ebooks.info

For Dummies, Windows Vista For Dummies, and *Windows 7 For Dummies,* all by Andy Rathbone.

Using the CD

These steps will help you install the items from the CD to your hard drive:

1. **Insert the CD into your computer's CD-ROM drive.**

   The license agreement appears.

   Note to Windows users: The interface won't launch if you have autorun disabled. In that case, choose Start➪Run. (For Windows Vista, choose Start➪All Programs➪Accessories➪Run.) In the dialog box that appears, type ***D*****:\Start.exe**. (Replace *D* with the proper letter if your CD drive uses a different letter. If you don't know the letter, see how your CD drive is listed under My Computer.) Click OK.

   Note for Mac Users: When the CD icon appears on your desktop, double-click the icon to open the CD and double-click the Start icon.

   Note for Linux Users**:** The specifics of mounting and using CDs vary greatly between different versions of Linux. Please see the manual or help information for your specific system if you experience trouble using this CD.

2. **Read through the license agreement and then click the Accept button if you want to use the CD.**

   The CD interface appears. The interface allows you to browse the contents and install the programs with just a click of a button (or two).

3. **Copy the C++ source code onto your hard disk.**

   You can view the source code on the CD-ROM but you cannot build or execute programs there.

4. **Windows users will want to install the Code::Blocks environment.**

   Chapter 2 takes you through step-by-step instructions on how to install Code::Blocks and how to create your first program.

What You'll Find on the CD

The following sections are arranged by category and provide a summary of the software and other goodies you'll find on the CD. If you need help with installing the items provided on the CD, refer to the installation instructions in the preceding section.

www.it-ebooks.info

For each program listed, I provide the program platform (Windows or Mac) plus the type of software. The programs fall into one of the following categories:

- *Shareware programs* are fully functional, free, trial versions of copyrighted programs. If you like particular programs, register with their authors for a nominal fee and receive licenses, enhanced versions, and technical support.
- *Freeware programs* are free, copyrighted games, applications, and utilities. You can copy them to as many computers as you like — for free — but they offer no technical support.
- *GNU software* is governed by its own license, which is included inside the folder of the GNU software. There are no restrictions on distribution of GNU software. See the GNU license at the root of the CD for more details.
- *Trial, demo,* or *evaluation* versions of software are usually limited either by time or functionality (such as not letting you save a project after you create it).

CPP programs

For all environments. All the examples provided in this book are located in the Beginning_Programming-CPP directory on the CD and work with Macintosh, Linux, Unix, and Windows and later computers. These files contain the sample code from the book. Each example program is in its own folder. For example, the Conversion program is in:

```
Beginning_Programming-CPP\Conversion
```

For Windows. I have built a set of workspace and set of project files for Code::Blocks that allows you to recompile all the programs in the book with a single mouse click. The AllPrograms.workspace file is located in the Beginning_Programming-CPP folder. (See Chapter 2 for an explanation of Code::Blocks Project files.) You **must** copy the source code from the CD-ROM onto your hard disk before you use it.

Code::Blocks development environment

For Windows. Code::Blocks is an “open source, cross platform” freeware environment designed to work with a number of different compilers. The version included on the CD-ROM is bundled with the GNU gcc C++ compiler (version 4.4) for 32-bit versions of Windows (if you don’t know whether your Windows is 32-bit or not, it almost certainly is). Code::Blocks is supported by “The Code::Blocks Team/” You can find more information at `www.codeblocks.org`.

For non-Windows. You can download a version of Code::Blocks that works for your operating system at `www.codeblocks.org`. They’ve got versions of Code::Blocks for just about every environment short of the iPhone.

www.it-ebooks.info

Troubleshooting

I tried my best to compile programs that work on most computers with the minimum system requirements. Alas, your computer may differ, and some programs may not work properly for some reason.

I include Code::Blocks workspace and project files for the included C++ source. This allows you to recompile all the programs with literally a single click. However, these project files assume that the programs are installed in the directory `C:\\Beginning_Programming-CPP`. You'll have to set up your own project files if you decide to install the source code in a different directory.

Other common problems are that you don't have enough memory (RAM) for the programs you want to use, or you have other programs running that are affecting installation or running of a program. If you get an error message such as `Not enough memory` or `Setup cannot continue`, try one or more of the following suggestions and then try using the software again:

- **Turn off any antivirus software running on your computer.** Installation programs sometimes mimic virus activity and may make your computer incorrectly believe that it's being infected by a virus.
- **Close all running programs.** The more programs you have running, the less memory is available to other programs. Installation programs typically update files and programs; so if you keep other programs running, installation may not work properly.
- **Have your local computer store add more RAM to your computer.** This is, admittedly, a drastic and somewhat expensive step. However, adding more memory can really help the speed of your computer and allow more programs to run at the same time.

Customer Care

If you have trouble with the CD-ROM, please call Wiley Product Technical Support at 800-762-2974. Outside the United States, call 317-572-3993. You can also contact Wiley Product Technical Support at `http://support.wiley.com`. Wiley Publishing will provide technical support only for installation and other general quality control items. For technical support on the applications themselves, consult the program's vendor or the author at `www.stephendavis.com`.

To place additional orders or to request information about other Wiley products, please call 877-762-2974.

www.it-ebooks.info

Index

• Numbers •

0 (zero)
  importance in debugging, 247
  using with decimal point, 154
1GB, measurement of, 10

• Symbols •

+ (addition) operator, order of precedence, 52
&& (AND) logical operator, explained, 79
-> (arrow) operator, using, 270
%= (assignment) operator, order of precedence, 52
*= (assignment) operator, order of precedence, 52
+= (assignment) operator, order of precedence, 52
= (assignment) operator, order of precedence, 52
-= (assignment) operator, order of precedence, 52
* (asterisk), using with pointer variable, 188
\ (backslash)
  special symbol for, 66
  use with filenames, 66
\ \ (backslash), indicating in Windows/DOS, 365
| (binary OR) operator, using, 368
{} (braces)
  using, 72–73
  using with classes, 224–225
: (colon) syntax, using with variables, 320
, (comma) operator, using, 113–115
– (decrement) operator
  order of precedence, 52
  prefix and postfix versions, 55–56
  using, 54–55
/ (division) operator, order of precedence, 52
" (double quotes)
  special symbol for, 66
  using with characters, 65
== (equality) operator, explained, 70
= (equals sign), using with variables, 49–50
> (greater than) operator, meaning of, 70
>= (greater than or equal to) operator, meaning of, 70
++ (increment) operator
  order of precedence, 52
  prefix and postfix versions, 55–56
  using, 54–55
!= (inequality) operator, meaning of, 70
<< (left-shift) operator, role in stream I/O, 363
< (less than) operator, meaning of, 70
<= (less than or equal to) operator, meaning of, 70
% (modulo) operator, order of precedence, 52–53
* (multiplication) operator, order of precedence, 52
— (negative) operator
  order of precedence, 52
  using, 54–55
! (NOT) operator
  explained, 79
  using with streams, 377
|| (OR) logical operator, explained, 79
() (parentheses)
  combining for operators, 54
  using with pointers and arrays, 206
  using with pointers to objects, 270
(pound sign)
  use with binary operators, 56–57
  using with include files, 133
>> (right-shift) operator, role in stream I/O, 363
; (semicolon), missing, 40–41
' (single quotes)
  interpreting, 66
  special symbol for, 66
  using with characters, 65
— (subtraction) operator, order of precedence, 52
? (ternary operator), using, 406
~ (tilde), using with destructors, 297
_ (underscore), using with variable names, 48

www.it-ebooks.info

• A •

abstraction
  level of, 254–255
  relationship to classification, 256
access function, example of, 285
addition (+) operator, order of precedence, 52
algorithm
  converting into program, 13–16
  creating for "human computer," 11–12
AmbiguousBinding program, 346–347, 350–351
AND (&&) logical operator, explained, 79
applications. *See* programs
`ArgOutOfRange` class, creating, 392–394
arguments
  arrays of, 216–221
  defaulting to constructors, 311
  defining constructors with, 303–307
  passing by reference, 193–195
  passing by values, 190–193
  passing from Code::Blocks, 219–220
  passing through command line, 217–219
  passing through Windows, 220–221
  passing to functions, 123–126
  promoting, 163
arithmetic, calculation speeds, 155–156. *See also* mathematical operators
array length, specifying, 372–373
array pointers, using, 210–212
ArrayDemo program, 168–170
arrays. *See also* ASCIIZ array; parallel arrays; pointers; variables
  of arguments, 216–221
  ASCII-zero character array, 173–174
  computing size of, 198
  declaring, 166–167
  destroying to heap, 307
  fixed-size, 197
  indexing into, 167–168
  initializing, 171–172
  of objects, 226–227
  overview of, 165–166
  performing Bubble Sort on, 232–233
  versus pointers, 214
  of pointers, 215–216
  retaining scores' input in, 168–171
  tracking amount of data in, 174
  use of, 172
  using `displayAverage()` function with, 170
  using `for` loop with, 171
arrow syntax, using with pointers to objects, 270
ASCII character set, 60–62
ASCIIZ array. *See also* arrays
  declaring and initializing, 174–175
  string manipulation functions, 182–183
ASCIIZ string
  copying, 198
  terminating, 386
ASCII-zero character array, 173–174
assembler instructions, example from `Conversion` program, 18
assembly language, explained, 17
assignment operators
  as destructors, 358
  orders of precedence, 52
  overloading, 356–358, 362
  overloading to avoid bugs, 404
  in StudentAssignment program, 361
  using, 56–57
  using with classes, 226
asterisk (*), using with pointer variable, 188

• B •

backslash (\)
  special symbol for, 66
  use with filenames, 66
backslash (\ \), indicating in Windows/DOS, 365
backup file, creating, 375–376
`bad()` function, returning for `fstream` object, 369
base 8, explained, 65
base 16, coding characters in, 66
`Beginning_Programming-CPP` folder
  accessing, 33
  subfolders in, 35
bell, special symbol for, 66
binary data, writing, 368
binary logic, omission of, 407
binary mode, using, 369, 377
binary operators
  format of, 70
  order of precedence, 52

using # (pound sign) with, 56–57
using in expressions, 51–53
binary OR (|) operator, using, 368
binary versus text mode, 368
binding. *See* EarlyBinding program; LateBinding program
birth year, using nested `if` statement with, 76–78
bits, defined, 188
`bool` methods, using with `fstream` classes, 373–374
`bool` type, using in expressions, 75
`boolalpha` flag, explained, 378
braces ({})
using, 72–73
using with classes, 224–225
BranchDemo program, executing, 70–72
`break` statement
adding during debugging, 146
necessity of, 104
using, 84–85
using with loops, 102–104
breakpoints, setting in debugger, 240
Bubble Sort. *See also* class objects
performing, 232–233
using with class objects, 227
buffer
adding newline to, 380
avoiding overflow of, 372–373
bug avoidance. *See also* debugging; errors; exceptions
commenting code while writing, 401
declaring destructors virtual, 403–404
enabling error messages, 399–400
enabling warnings, 399–400
limiting visibility of class internals, 402
providing copy constructor, 404
providing overloaded assignment operator, 404
single-stepping paths in debugger, 401
tracking heap memory, 402
using exceptions to handle errors, 403
using good coding style, 400
zeroing out pointers, 403
Build step, performing, 38
byte, defined, 188

• C •

C++
as high level language, 19–20
managing compile time errors, 40
rules, 40
syntax, 37
uppercase versus lowercase, 37, 84
C++ programs, programs required for, 21. *See also* programs
C++ Standard, release of, 410
cache, flushing, 380
calculator, implementing via `switch` statement, 85–87
`calculator()` function
debugging, 148
displaying line numbers for, 146
testing, 141–143
CalculatorError program, error message in, 147. *See also* SwitchCalculator program
CalculatorError1 program, debugging, 139–141
CalculatorError2 program, 144–145
carriage return, special symbol for, 66
casts, using in character encoding, 64
`catch` keyword, using in exception mechanism, 387–389
`catch(...)` phrase, using, 391
.`cbp` extension, use in Code::Blocks, 27, 35
CD-ROM
Code::Blocks IDE, 413
CPP programs, 413
demo software, 413
evaluation software, 413
freeware programs, 413
GNU software, 413
shareware programs, 413
trial software, 413
troubleshooting, 414
using, 41, 412
Celsius to Fahrenheit conversion code, 18–19
`cerr` object
versus `cout`, 365
use of, 364–365
`char` keyword, using, 59

www.it-ebooks.info

char methods, using with fstream classes, 373–374
char type, number of bits in, 188
char variable
  contents of, 188
  explained, 159
  limitations of, 67
char versus int values, 64
character encoding
  example of, 63–65
  using casts in, 64
character variables, defining, 59. *See also* variables
characters. *See also* garbage characters
  applying true/false values to, 75
  encoding, 60–63
  encoding strings of, 65
  printable versus unprintable, 62–63
  using double quotes (") with, 65
  using single quotes (') with, 65
cin, extracting from, 364
class internals, limiting visibility of, 402
class members
  declaring as friends, 288
  naming, 262–263
class objects, initializing, 226, 289. *See also* Bubble Sort
classes
  assignment operators, 226
  creating instances of, 225
  declaring, 225
  definition, 224
  format of, 224
  inheriting from, 340
  naming rules for, 225
  versus objects, 225
  public keyword, 224–225
  struct keyword, 225
  use of open brace ({), 224–225
clear() function, calling for fail() flag, 369
COBOL (Common Business Oriented Language), 18–19
code, commenting while writing, 401
CodeBlocks, lengths of addresses in, 189
Code::Blocks IDE
  availability on CD-ROM, 413
  Beginning_Programming-CPP folder, 27, 32
  Build command, 30
  .cbp extension, 27, 35
  command line, 30
  Console Application, 27
  cout, 30
  creating folders, 28
  creating projects, 27–29
  downloading, 23, 413
  environment, 26
  "Hello, world!" message, 32
  HelloWorld project, 28–29
  installing, 23–25
  .layout extension, 27
  main.cpp file, 30
  organizing projects in, 35
  passing arguments from, 219–220
  Run command, 32
  selecting C++ as language choice, 27–28
  setting up subfolders, 29
  starting, 26
  Step Into command, 242
  terminated status of process, 30–31
  testing default project, 30–32
  types of programs created by, 26–27
  use of gcc compiler, 31
  using to create include files, 134–135
coding style
  avoiding errors in, 90–91, 400
  being consistent in, 90–91, 400
  variable naming conventions, 91
colon (:) syntax, using with variables, 320
comma (,) operator, using, 113–115
command line, passing arguments from, 217–219
commands, misspelling, 38–39
CommaOperator program, 113
comments, examples of, 43
Common Business Oriented Language (COBOL), 18–19
comparison operators
  binary format, 70
  using in BranchDemo program, 71
compile time errors, managing, 40
compiler
  function of, 22
  requirement of, 21
compound expressions
  conditional, 78–80
  decomposing, 53–54

www.it-ebooks.info

CompoundStudent program, 318–320
computer languages
  assembly language, 17
  C++, 19–20
  high-level, 18–19
  machine language, 17
ConcatenateError2 program, 245
ConcatenateHeap program, 199–201, 236–238
ConcatenateNString program, 181–182
`concatenateString()` function
  bug in, 181
  debugging, 245–249
  executing in debugger, 242
  pointer version of, 208–211
ConcatenateString program, 178–179
concatenating strings, 177–180
conditional expressions, compound, 78–80. *See also* expressions
Console Application
  defined, 23
  selecting in Code::Blocks, 27
`const` arguments, problems with, 163
`const char[]` type, using, 177
`const` descriptor, using with variables, 158
`const` keyword, using to declare variables, 160
`const` string, throwing pointer to, 391
`const` variable, initializing, 161
constant characters, modifying, 177
constant values, initializing data members to, 321
constants
  managing, 212–214
  for opening files, 366
  types of, 51, 160–161
constructors. *See also* copy constructors
  defaulting arguments to, 311
  defaults, 312–313
  defining with arguments, 303–307
  versus destructors, 297
  versus functions, 304
  initializing data members with, 315–318
  limitations on, 291
  for opening files for output, 367
  overloading, 307–311
  in StudentAssignment program, 358–360
  using, 289–291
  using defaults with data members, 314–315
`continue` command, using with loops, 102–104
`Conversion` program
  assembler instructions in, 18
  building, 38
  entering, 35–37
  finding error in, 92–93
  first version of, 92
  running, 42
  templates, 42–44
  test data for, 94
  without template, 44
Conversion project
  addition to HelloWorld project, 35
  creating, 33–34
ConversionError2 program, examining, 95
copies
  avoiding, 330–331
  deep versus shallow, 328
  shallow, 356
copy constructors. *See also* constructors
  creating, 327–330
  default, 324–325
  invoking, 330
  making shallow copies, 325–327
  overview of, 323–324
  providing to avoid bugs, 404
  in StudentAssignment program, 358–360
CopyFiles program, 375–376
copying objects, 323–327
counting, using floating point variables in, 155
coupling, reducing, 282, 285–286
`cout`
  versus `cerr`, 365
  declaration of, 364
CPP programs, availability on CD-ROM, 413
customer care, accessing, 414
CustomExceptionClass program, 392–395

• D •

data, grouping, 223–224
data members
  constructing, 294–296
  destructing, 300–301
  flagging with keyword `static`, 302
  initializing to constant values, 321

www.it-ebooks.info

data members *(continued)*
  initializing with constructors, 315–318
  initializing with default constructor, 314–315
data types, limit of number size in, 159–160
debug code, writing for calculator() function, 143
debug functions, creating, 143–144
debugger utility
  components of, 241
  Continue option, 247
  Debug window, 246–247
  executing concatenateString() function, 242
  features of, 236
  fixing bugs, 245–249
  Next Line option, 243–244, 246
  rebuilding executable, 239
  setting breakpoints, 240
  single-step commands, 243
  single-stepping paths in, 401
  starting, 239–241
  Tool Tips, 242
  turning on information, 239
  using to navigate through programs, 241–245
  Watches window, 242, 246
  while loop structure, 244
debugging. *See also* bug avoidance; errors; exceptions
  CalculatorError1 program, 139–141
  ConcatenateHeap, 236–238
  dividing programs into functions, 139–141
  importance of 0 (zero) in, 247
  outfitting functions for testing, 143–146
  performing unit level testing, 141–143
  unit testing, 146–148
dec
  flag, 378
  manipulator, 381
decimal point, using 0 with, 154
decrement (--) operator
  order of precedence, 52
  prefix and postfix versions, 55–56
  using, 54–55
deep copy, performing for source object, 328
DeepStudent program, 328–330
delete keyword
  using with memory heap, 198–199
  using with objects allocated off heap, 277
delete[] versus delete, 307
demotion, defined, 155
deposit() member function, defining, 261–262
destructors
  assignment operators as, 358
  versus constructors, 297
  declaring virtual, 352–354, 403–404
  invoking, 297
  in StudentAssignment program, 358–361
  for subclasses in inheritance, 342
  using with constructors and arguments, 304
  using with data members, 300–301
DisplayASCIIZ program, 175–176
division (/) operator, order of precedence, 52
DOS/Windows, indicating backslash in, 365
double limitations
  calculation speed, 155–156
  counting, 155
  loss of accuracy, 156–157
  memory consumption, 156
  range, 157
double quotes (")
  special symbol for, 66
  using with characters, 65
double variable
  assigning to int, 155
  digits of accuracy, 156–157
  explained, 159
  largest number stored by, 157
  memory consumption, 156
  promoting int to, 154
  size and range of, 160
  using to fix truncation, 153–154

• E •

EarlyBinding program, 344–345
else keyword, using, 73–74
endl object
  strstream version, 386
  using with output streams, 380
endl value, explained, 65
ends object, inserting, 386
enumerated types, problem with, 407
equality (==) operator, explained, 70
equals sign (=), using with variables, 49–50
error checking programs, 95–96

www.it-ebooks.info

error messages, enabling to avoid bugs, 399–400
error return, defined, 120
errors. *See also* bug avoidance; debugging; exceptions; run-time errors
  avoiding introduction of, 90–91
  build- versus compile-time, 89–90
  finding, 92–93
  forgetting to initialize variables, 50
  handling, 391
  handling via exceptions, 403
  identifying types of, 89–90
  missing semicolon, 40–41
  misspelled commands, 38–39
  run-time, 90, 93–96
  shallow copies, 327
example programs. *See* program examples
exception classes
  customizing, 392–395
  restrictions on, 395
exception mechanism
  `catch` keyword, 387–388
  `throw` keyword, 387–388
  `try` keyword, 387–388
exceptions. *See also* bug avoidance; debugging; errors
  catching, 391
  processing, 390–391
  rethrowing, 391
  throwing, 391–392
  using to handle errors, 403
executable file, finding path to, 218
expressions. *See also* conditional expressions; mixed mode expression, defined
  binary operators, 51–53
  compound, 53–54
  decomposing compound, 53–54
  defined, 14
  example, 51
  multiple operators in, 53–54
  operators, 51
  values and types of, 75
  versus variables, 47
extensions, unhiding, 36
extractor, `operator>>()` as, 364

• F •

`factorial()`, error encountered in, 392–394
Factorial program
  `for` loop in, 111–112
  parts of loop in, 109–110
  while loop in, 100–101
`factorial.cpp`, creating, 131–133
FactorialException program, 388–389
FactorialFunction program, 124–128
FactorialModule project
  building result, 137
  creating `factorial.cpp`, 131–133
  `#include` file, 133–134
  including #include files, 134–135
  `main.cpp`, 136
  steps for creation of, 130
`fail()` flag, setting and clearing, 369
`fail()` member function, using, 371–372
fields, sorting algorithms for, 232
FileCopy program, 375–376
filename extensions, unhiding, 36
files. *See also* input files; output files
  adding to contents of, 368
  open modes for, 367–368
  opening in binary mode, 368
  opening in text mode, 368
  reading from, 368
  saving, 37
  writing to, 368
`fill(char)`, calling, 379
`fixed` flag, explained, 378
fixed-size arrays, problem with, 197
`flags()` member function
  format flags for, 378–379
  using, 379
flat tire, changing, 11–17
`float` variable
  explained, 159
  size and range of, 160
floating point constant, assumption about, 154
floating point result, assigning to `int` variable, 155
floating point variables
  overflow of, 160
  types of, 153
  using for counting, 155

www.it-ebooks.info

flow, controlling via switch statement, 81–84
flush() member function, calling, 380
fmtflags methods, using with fstream classes, 374
fn() function
  calling in NamedStudent program, 306
  late-binding, 352
  making changes permanent, 192–193
  statements in, 192–193
  using in StudentDestructor program, 299–300
folders, creating for projects, 33
for loop. *See also* loops
  in arrays of pointers, 215–216
  in Factorial program, 111–112
  flow of, 110
  sections of, 112
  setup section, 112, 114
  using in ConcatenateHeap program, 201
  using in ConcatenateString program, 180
  using with arrays, 171
  using with main() program, 122–123
  versus while loop, 111, 115
format of input, controlling, 378–380
formatting member functions, span of, 380–381. *See also* member functions
friend declaration, using with functions, 286–288
fstream classes
  destructor for, 367
  explained, 365
  member functions of, 373–374
fstream objects
  constructing, 369
  errors related to, 369
  returning bad() function for, 369
function arguments, using, 123–126
function names, overloading, 126, 162
function overloading, explained, 126
function prototype declarations, defining, 127–128
FunctionDemo program, 121–123
functions. *See also* getter function, example of; member functions; setter function, defined
  call, 120
  calling with object pointers, 272–274
  calling with object values, 271–272
  versus constructors, 304
  declaration, 120
  definition, 120
  dividing programs into, 139–143
  elements of, 120
  flow of control, 120
  friend declaration, 286–288
  with multiple arguments, 125
  outfitting for testing, 143–146
  overloading, 307
  overview of, 118–119
  passing variables to, 161–164
  return, 120
  returning values to callers, 120–121
  similarities between, 409–410
  testing separately, 141–143
  writing and invoking, 119–120

• G •

garbage characters, displaying in debugging, 247–249. *See also* characters
gcc compiler, using with Code::Blocks, 31
get() method, using with input files, 375
getline() member function, using, 372–373, 375, 382–385
getter function, example of, 285. *See also* functions
goto command, using, 405–406
greater than (>) operator, meaning of, 70
greater than or equal to (>=) operator, meaning of, 70
GSInherit example, 338–340

• H •

HAS_A relationship, using in inheritance, 342
heap
  allocating memory from, 197
  allocating objects off, 278
  returning memory to, 198–199
heap memory, keeping track of, 402
HelloWorld project, creating in Code::Blocks, 28–29
hex
  flag, 378
  manipulator, 381
hexadecimal, coding characters in, 66

www.it-ebooks.info

high level languages, overview of, 18–19
"human computer"
  algorithm, 11–12
  processors, 16–17
  program, 13–16
  tire changing language, 12

• I •

IDE (Integrated Development Environment), 22–23
if clause, using with else, 74
IF statement in TCL (Tire Changing Language), 13–14
if statements
  encountering, 70
  format of, 69
  nesting, 75–78
  versus switch statements, 84
ifstream class
  constructor for, 365
  definition of, 365
  destructor for, 366
  using with input objects, 365
#include directives, inclusion in standard template, 137
#include files
  creating for FactorialModule, 133–134
  including, 134–135
include statements, examples of, 43
increment (++) operator
  order of precedence, 52
  prefix and postfix versions, 55–56
  using, 54–55
indentation, use of, 91
index, providing for arrays, 167–168
index operator, using on pointer variables, 207–208
inequality (!=) operator, meaning of, 70
inheritance. *See also* object-oriented programming, model for
  advantages of, 336–337
  from classes, 340
  defined, 337
  implementing, 337–342
  multiple, 409
  subclasses, 337
  terminology, 337
init() function
  in OverloadedStudent program, 310–311
  using with protected members, 289–291
input files, using get() method with, 375. *See also* files
input objects, creating, 365–366. *See also* objects
input/output (I/O), errors related to, 369
InputPerson program, 227–230
inserter, operator>>() as, 364
inserter operator, using with output objects, 366–367
int gcount() method, using with fstream classes, 374
int keyword, using with variables, 47–48
int variable
  assigning double to, 155
  assigning floating point result to, 155
  bytes taken up by, 191
  explained, 151, 158
  HAS_A relationship, 342
  limited range of, 152–153
  maximum value, 152–153
  memory consumption, 156
  promoting to double, 154
  round-off limitation, 151–152
  size and range of, 160
  standard size of, 159
int versus char values, 64
int width() methods, using with fstream classes, 374
integer constants, legal versus illegal, 50–51
integers
  interpreting, 60
  limited range of, 152–153
  round-off limitation, 151–152
  truncation of, 152
Integrated Development Environment (IDE), 22–23
intrinsic type, explained, 151
I/O (input/output), errors related to, 369
I/O manipulators, member functions for, 381
I/O stream format flags, 378
iomanip, including for manipulators, 381
ios_base: :, using, 377
iostream include
  declaration of, 364
  prototype declarations in, 363–364

www.it-ebooks.info

isalppha(char c) function, described, 182
isdigit(char c) function, described, 182
islower(char c) function, described, 182
isprint(char c) function, described, 182
isspace(char c) function, described, 182
istream& put() method, using with fstream classes, 374
istrstream versus istringstream classes, 382
isupper(char c) function, described, 182

• K •

keyboard input, performing, 364
keystrokes, saving, 270
keywords, using with variables, 48

• L •

languages. *See* computer languages
LateBinding program, 349–351
.layout extension, use in Code::Blocks, 27
left flag, explained, 378
left-shift (<<) operator, role in stream I/O, 363
less than (<) operator, meaning of, 70
less than or equal to (<=) operator, meaning of, 70
levels of abstraction, explained, 119
logical expressions
  comparing values in, 70–72
  defined, 14
  values of, 75
logical operators, explained, 79
long double descriptor, using with variables, 158
long double variable
  explained, 159
  size and range of, 160
long int descriptor, using with variables, 158
long int variable
  explained, 159
  size and range of, 160
long long int descriptor, using with variables, 158
long long int variable
  explained, 159
  size and range of, 160
"loop and test" statement, including in TCL, 14–16
loops. *See also* for loop; while loop
  body, 109
  increment, 109
  nesting, 105–108
  setup, 109
  test expression, 109
lowercase versus uppercase, 37, 84

• M •

machine language, representing, 17
Macintosh, system requirements for, 411
main()
  as function, 125–126
  requirement of, 43
main() arguments
  arrays of arguments, 216–221
  arrays of pointers, 215–216
  in program template, 214–215
main() program, using for loop with, 122–123
main.cpp file
  creating for FactorialModule, 136
  displaying, 35
  editing contents of, 36–37
  opening, 36
main.cpp.backup file, creating, 378
manipulators, member functions for, 381
mathematical operators. *See also* arithmetic, calculation speeds; operators, overloading
  combining parentheses, 54
  order of precedence, 52–53
member functions. *See also* formatting member functions, span of; functions; overriding member functions; protected members, initializing objects with
  accessing members from, 264–266
  adding keyword protected:, 282–285
  binding considerations, 351
  calling, 263–264, 268
  defining, 261–262

www.it-ebooks.info

defining outside classes, 266–267
invoking for pointers to objects, 271
for manipulators, 381
overloading, 267–268, 343–344
`protected` keyword, 281
protecting, 282
virtual, 350
memory
address in, 188
arrangement for arrays of objects, 227
handling, 191
returning to heap, 198–199
memory address, including in pointer variable, 188
memory heap. *See* heap
memory layout, example of, 192
memory model, applying for pointers, 204
mixed mode expression, defined, 154. *See also* expressions
mixed mode overloading, 162–164
modules
breaking programs into, 129–130
linking, 129
naming, 132
modulo (%) operator, order of precedence, 52–53
MS-DOS window, opening in Windows, 219
multiplication (*) operator, order of precedence, 52
multiplication table, creating, 105–108
`MyData.txt` file
opening and reading, 366
opening and writing to, 367
using with ReadIntegers program, 370–372

• N •

NamedStudent program, 304–307
namespaces, using, 407–408
negative (—) operator
order of precedence, 52
using, 54–55
nested `if` statements, using, 75–78
nested loops
explained, 16
using, 105–108
`new` keyword
using to allocate objects off heap, 277
using with memory heap, 197
newline
adding to buffer, 380
special symbol for, 66
NOT (!) operator
explained, 79
using with streams, 377
`null` character
adding after final `while` loop, 248–249
using in string concatenation, 178
using with arrays, 174
NULL character, special symbol for, 66

• O •

object code, explained, 18
object pointers, calling functions with, 272–274
object-oriented programming, model for, 257–258. *See also* inheritance
objects. *See also* input objects, creating; output objects, creating; pointers to objects; source object, performing deep copy of
accessing members of, 225–226
activating, 259–261
allocating off heap, 278
arrays of, 226–227
avoiding creating copies of, 274
versus classes, 225
copying, 323–327
initializing, 289
passing addresses of, 274–276, 330–331
passing by reference, 277
passing by values, 271–272, 274–276
"real types" of, 347
`oct`
flag, 378
manipulator, 381
octal, identifying, 65
`ofstream` class
constructor for, 366–367
definition of, 365
open modes, using with files, 367–368

www.it-ebooks.info

`operator>>()`
as inserter, 364
use as extractor, 364
operators, overloading, 355–356, 361–362. *See also* mathematical operators
OR (|) operator, using, 368
OR (||) operator, explained, 79
`ostream` object, use of, 364
`ostream& ()` methods, using with `fstream` classes, 374
`ostrstream` versus `ostringstream` classes, 382, 395
output and input. *See* I/O (input/output), errors related to
output files, using `put()` method with, 375. *See also* files
output objects, creating, 366–367. *See also* objects
output statements, using to find errors, 95
output streams, terminating, 380
overflow, defined, 160
OverloadedStudent program, 308–310
overriding member functions. *See also* member functions
AmbiguousBinding program, 346–347
EarlyBinding program, 344–345
late binding, 348–351

• P •

parallel arrays, storing data in, 223–224. *See also* arrays
parentheses (())
combining for operators, 54
using with pointers and arrays, 206
using with pointers to objects, 270
PassByReference program, 191, 193–197
PassObjects program, 274–276
PC, system requirements for, 411
PL/1 language, 40
pointer increment operator, using, 208–210
pointer types, operations on, 212
pointer variables, using index operator on, 207–208
pointers. *See also* arrays
addition versus indexing, 205–208
applying memory model for, 204
versus arrays, 214
arrays of, 215–216
declaring, 188–190
operations on, 203–205
overview of, 187–188
using constants with, 213
zeroing out, 403
pointers to objects. *See also* objects
arrow syntax, 270
example of, 269–270
invoking member functions, 271
using parentheses (()) with, 270
pound sign (#)
use with binary operators, 56–57
using with include files, 133
precedence, order of, 52–53
PrintArgs program, 216–217
`printErr()` function, using, 144–146
Product program, `break` and `continue` in, 102–103
program examples
AmbiguousBinding, 346–347
ArrayDemo, 168–170
BranchDemo, 70–72
CalculatorError1, 139–141
CalculatorError2, 144–145
CommaOperator, 113
CompoundStudent, 318–320
ConcatenateError2, 245
ConcatenateHeap, 199–201
ConcatenateNString, 181–182
ConcatenateString, 178–179
Conversion, 44, 92
ConversionError2, 95
CopyFiles, 375–376
CustomExceptionClass, 392–395
debugging ConcatenateHeap, 236–238
DeepStudent, 328–330
DisplayASCIIZ, 175–176
EarlyBinding, 344–345
Factorial, 100, 109–112
`factorial.cpp`, 131–133
FactorialException, 388–389
FactorialFunction, 124–128
FileCopy, 375–376
FunctionDemo, 121–123
GSInherit, 338–340
InputPerson, 227–230
LateBinding, 349–350
NamedStudent, 304–307
NestedLoops, 105–106

www.it-ebooks.info

OverloadedStudent, 308–310
PassByReference, 191, 193–197
PassObjects, 274–276
pointer version of concatenateString(), 208–211
PrintArgs, 216–217
Product, 102–103
prototype declaration, 128
ReadIntegers, 370
ShallowStudent, 325–327
SimpleStudent, 283–284
StringStream, 382–385
StudentAssignment, 358–360
StudentConstructor, 292–293
StudentDestructor, 298–299
StudentID class, 314–315
SwitchCalculator, 85–86
toupper() function, 183–184
TutorPairConstructor, 295–296, 300–301
programs. *See also* C++ programs, programs required for
breaking into modules, 129–130
building, 21–22
development process, 22
dividing into functions, 139–143
error checking, 95–96
linking step, 22
project file, finding path to, 217–218
projects
activating, 35
adding source files to, 131
creating, 33–34
organizing, 35
protected: keyword adding, 282–285
protected keyword versus public keyword, 281
protected members, initializing objects with, 289–291. *See also* member functions
prototype declaration, explained, 127–128
psz prefix, using with pointers, 208
public keyword
versus protected keyword, 281
using in inheritance, 337–338
using with classes, 224–225
pure virtual member function, explained, 408
put() method, using with output files, 375

• R •

read() member function, using, 375–376
ReadIntegers program, 370
real numbers, defined, 153
right flag, explained, 378
right-shift (>>) operator, role in stream I/O, 363
run-time errors. *See also* errors
adding output statements, 94
versus compile-time errors, 40
executing test cases, 94
finding, 93–96
formulating test data, 93–94
run-time type, defined, 347

• S •

safeFn() function, using in ReadIntegers program, 371–372
saving files, 37
Savings account class
calling member function in, 263–264
current object in, 265–266
deposits in, 260
functions in, 261–262
scientific flag, explained, 378
scores, averaging, 168–171
semicolon (;), missing, 40–41
setf() member function, format flags for, 378–379
setfill(c) manipulator, described, 381
setprecision(n) manipulator, described, 381
setter function, defined, 285. *See also* functions
setw(n) manipulator, described, 381
shallow copy, defined, 356
ShallowStudent program, 325–327
short int descriptor, using with variables, 158
short int variable
explained, 159
size and range of, 160
showbase flag, explained, 378
showpoint flag, explained, 378
SimpleStudent program, 283–284

www.it-ebooks.info

single quotes (')
  interpreting, 66
  special symbol for, 66
  using with characters, 65
skipws flag, explained, 378
SomeClass class example, 321
sorting algorithm. *See* Bubble Sort
source code, explained, 18
source file windows, closing, 35
source files, adding to projects, 131
source object, performing deep copy of, 328. *See also* objects
Sources folder, opening, 35
special characters, examples of, 66
stack unwinding, role in exceptions, 390
statements
  adding to find errors, 95
  avoiding cramming of, 212
  defined, 14
  executing once, 93
static data members, overview of, 302
static keyword, using with variables, 138
strcmp() function, described, 182
stream I/O, overview of, 363–365
streams, reading and writing directly, 375–378
streamsize precision() methods, using with fstream classes, 374
string class
  behavior of, 382
  omission of, 408
  resizing, 373
strings
  ASCIIZ manipulation functions, 182–183
  concatenating, 177–180
  displaying, 175–176
strlen() function, calling, 198
strlen(char s[]) function, described, 182
strncat() function, described, 183
strncpy() function, described, 182
strstream classes, deprecation of, 382
strstream versus stringstream classes, 382–386
struct keyword, using with classes, 225
Student class
  constructor with arguments, 303–304
  default copy constructor, 324
  default default constructor, 312
  inheritance in, 337
  initializing data member with constructor, 317–318
  from OverloadedStudent program, 308–310
  public member functions, 283
  using constructor with, 290–291
student object, performing deep copy of, 328–330
StudentAssignment program, 358–360
StudentConstructor program, 292–293
StudentDestructor program, 298–299
StudentID class example
  with default constructor, 314–315
  with different constructor, 315–318
subtraction (—) operator, order of precedence, 52
sumSequence() function, using, 121–123
switch statement
  calculator example, 84
  versus if statements, 84
  using to control flow, 81–84
SwitchCalculator program, 85–86, 139–141. *See also* CalculatorError program
syntax requirements, 37
system requirements, 411–412

• T •

tab, special symbol for, 66
TCL (Tire Changing Language)
  IF statement, 13–14
  logical expression, 14
  "loop and test" statement, 14–16
  syntax, 13
  verbs and nouns, 13
Technical Support, contacting, 414
templates
  comments, 43
  example, 42–43
  features of, 409–410
  include files, 43
  main(), 43
  system("PAUSE"), 44
ternary operator (?), using, 406
test cases. *See also* unit testing
  executing, 94
  generating for calculator() function, 141–143

www.it-ebooks.info

test data
  for Conversion program, 94
  deciding on, 93
text data, writing, 368
text editors, requirement of, 21
text versus binary mode, 368
this object, overview of, 279
throw, encountering, 390
throw; keyword, executing, 391
throw keyword, using in exception mechanism, 387–389
throwing exceptions, 391–392
tilde (~), using with destructors, 297
Tire Changing Language (TCL). *See* TCL (Tire Changing Language)
tire-changing module, components of, 118
tolower(char c) function, described, 183
toupper() function
  described, 183
  program example, 183–184
truncation, defined, 152
truncation problem, fixing via double variable type, 153–154
try block, finding end of, 390
try keyword, using in exception mechanism, 387–389
TutorPairConstructor program, 295–296
types
  dynamic versus static, 347
  run-time versus declared, 347
typing, reducing via inheritance, 336

• U •

unary operators
  order of precedence, 52
  using, 54–56
underscore (_), using with variable names, 48
Unicode, variants of, 67
unit level testing, performing, 141–143
unit testing. *See also* test cases
  defined, 89
  performing, 146–148
unitbuf flag, explained, 378
unsafeFn() function, using in ReadIntegers program, 370
unsetf() member function, format flags for, 378–379
unsigned descriptor, using with variables, 158
unsigned int variable, explained, 158
unwinding the stack, role in exceptions, 390
uppercase flag, explained, 378
uppercase versus lowercase, 37, 84
UTF variants, explained, 67

• V •

variable types, size and range of, 159–160
variables. *See also* arrays; character variables
  assigning values to, 49
  const descriptor, 158
  declaring, 47–48
  declaring constant, 160
  establishing naming conventions for, 91
  versus expressions, 47
  forgetting to initialize, 50
  initializing at declaration, 49–50
  legal combinations of, 158–159
  local versus global scope, 137–138
  long double descriptor, 158
  long int descriptor, 158
  long long int descriptor, 158
  memory consumption, 156, 160
  naming conventions, 48, 155
  passing to functions, 161–164
  promoting versus demoting, 155
  short int descriptor, 158
  unsigned descriptor, 158
  using : (colon) syntax with, 320
  using equals sign (=) with, 49–50
  using static keyword with, 138
virtual declarations, using with destructors, 352–354, 403–404
virtual keyword, using in late binding, 349–350
virtual member function, defined, 350
void keyword, using with functions, 121
void methods, using with fstream classes, 373–374

• W •

warnings, enabling to avoid bugs, 399–400
wchar_t variable, explained, 159
wchar_t "wide character," 67
while loop. *See also* loops
  adding terminating null after, 248
  creating, 99–102

www.it-ebooks.info

while loop *(continued)*
  versus for loop, 111, 115
  structure in debugger, 244
"wide character" of type wchar_t, 67
width(int) member function, using, 379–380
width(n) member function, using, 380–381
Wiley Product Technical Support,
    contacting, 414
Windows
  opening MS-DOS window in, 219
  passing arguments from, 220–221
  unhiding filename extensions in, 36
Windows/DOS, indicating backslash in, 365
withdraw() function, 266
withdraw() member function, defining,
    261–262
workspace, defined, 35
write() member function, using, 375–376

• X •

X: :X(const X&) format, explained, 323

• Z •

zero (0)
  importance in debugging, 247
  using with decimal point, 154

www.it-ebooks.info