Stephen Prata

Sixth Edition

C++ Primer Plus

Developer's Library

Paul DuBois
ISBN-13: 978-0-672-32938-8

Linux Kernel Development
Robert Love
ISBN-13: 978-0-672-32946-3

Developer's Library books are available at
as by subscription from Safari Books Onli

Developer's Library

informit.com/devlibrary

Stephe

Addiso

Upper Saddle River, NJ • Bosto
New York • Toronto • Montreal •
Cape Town • Sydney • Toky

duction, storage in a retrieval system, or transmission in any fo
tronic, mechanical, photocopying, recording, or likewise. To obt
al from this work, please submit a written request to Pearson
Department, One Lake Street, Upper Saddle River, New Jersey
request to (201) 236-3290.

ISBN-13: 978-0-321-77640-2
ISBN-10: 0-321-77640-2

Text printed in the United States on recycled paper at R.R. Don
Indiana.

Second printing: January 2012

Appendixes

Acknowledgments for the Third E

I'd like to thank the editors from Macmillan
played in putting this book together: Tracy D
Rosenberg. Thanks, too, to Russ Jacobs for h
Metrowerks, I'd like to thank Dave Mark, Al
their help and cooperation.

Acknowledgments for the Secon

I'd like to thank Mitchell Waite and Scott C
Joel Fugazzotto and Joanne Miller for guidin
Michael Marcotty of Metrowerks for dealing
CodeWarrior compiler. I'd also like to thank
time to give us feedback on the first edition
Holland, Andy Yao, Larry Sanders, Shahin M
thank Heidi Brumbaugh for her helpful con

Acknowledgments for the First E

Many people have contributed to this book.
for his work in developing, shaping, and resh
uscript. I appreciate Harry Henderson's worl

As the reader of this book, *you* are our most i
your opinion and want to know what we're d
areas you'd like to see us publish in, and any d
pass our way.

You can email or write directly to let us kno
book—as well as what we can do to make ou

Please note that we cannot help you with technica
that due to the high volume of mail we receive, we

When you write, please be sure to include th
name, email address, and phone number.

Email: feedback@developers-library.in
Mail: Reader Feedback
Addison-Wesley Developer's Li
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at ww
access to any updates, downloads, or errata th

C++ Primer Plus discusses the basic C lan
this book self-contained. It presents C++ fu
to-the-point programs that are easy to copy
input/output (I/O), how to make programs
the many ways to handle data, and how to u
features C++ has added to C, including the

- Classes and objects
- Inheritance
- Polymorphism, virtual functions, and r
- Function overloading
- Reference variables
- Generic, or type-independent, program
  Standard Template Library (STL)
- The exception mechanism for handlin
- Namespaces for managing names of fu

- It provides a variety of sidebars, includi

The author and editors of this book do ou simple, and fun. Our goal is that by the end o effective programs and enjoy yourself doing s

Sample Code Used in Thi

This book provides an abundance of sample c programs. Like the previous editions, this boo tied to any particular kind of computer, opera were tested on a Windows 7 system, a Macint Those programs using C++11 features requir the remaining programs should work with an

The sample code for the complete progra book's website. See the registration link given

How This Book Is Organiz

This book is divided into 18 chapters and 10

- **Chapter 1: Getting Started with C++** created the C++ programming languag

values of a single type; structures, whic
pointers, which identify locations in m
store text strings and to handle text I/
C++ `string` class. Finally, you'll learn
allocation, including using the `new` and
explicitly.

- **Chapter 5: Loops and Relational Ex**
repetitive actions, and C++ provides th
`for` loop, the `while` loop, and the `do`
they should terminate, and the C++ re
to guide such loops. In Chapter 5 you
process input character-by-character. F
sional arrays and how to use nested lo

- **Chapter 6: Branching Statements an**
intelligently if they can tailor their beh
learn how to control program flow by
ments and the conditional operator. Yo
help express decision-making tests. Als
for evaluating character relations, such
nonprinting character. Finally, you'll ge

at different methods of managing memo
which determine what parts of a progra

- **Chapter 10: Objects and Classes**—A
(such as a variable) is an instance of a cl
oriented programming and to class desi
mation stored in a class object and also
class objects. Some parts of an object ar
portion), and some are hidden (the priv
structors and destructors) come into pla
You will learn about all this and other c
how classes can be used to implement A
- **Chapter 11: Working with Classes**—
standing of classes. First, you'll learn abo
define how operators such as + will wor
friend functions, which can access class
You'll see how certain constructors and
be used to manage conversion to and fr
- **Chapter 12: Classes and Dynamic M**
have a class member point to dynamical
class constructor to allocate dynamic m
viding an appropriate destructor, of defi

automobile has a motor. You also can u
model such relationships. This chapter
ences among the different approaches.
let you define a class in terms of some
template to create specific classes in te
template enables you to create a stack
learn about multiple public inheritanc
one class.

- **Chapter 15: Friends, Exceptions, and**
of friends to include friend classes and
several new developments in C++, beg
mechanism for dealing with unusual p
function argument values and running
RTTI, a mechanism for identifying ob
alternatives to unrestricted typecasting.

- **Chapter 16: The `string` Class and th**
discusses some useful class libraries rec
is a convenient and powerful alternativ
class helps manage dynamically allocat
containers, including template represen
It also provides an efficient library of g

- **Appendix C: The ASCII Character S** set, along with decimal, octal, hexadecin
- **Appendix D: Operator Precedence**— order of decreasing precedence.
- **Appendix E: Other Operators**—Appe such as the bitwise operators, not cover
- **Appendix F: The `string` Template C** `class methods and functions.`
- **Appendix G: The Standard Template** Appendix G summarizes the STL conta algorithm functions.
- **Appendix H: Selected Readings and** some books that can further your unde
- **Appendix I: Converting to ISO Stan** for moving from C and older C++ im
- **Appendix J: Answers to Chapter Rev** the review questions posed at the end c

quer strategy.

- Most chapters are short enough to cov
- The book discusses *when* to use certain example, it links public inheritance to vate inheritance to *has-a* relationships, and when not to.

Conventions Used in This

This book uses several typographic conventio

- Code lines, commands, statements, vari in a `computer typeface`:

```
#include <iostream>
int main()
{
    using namespace std;
    cout << "What's up, Doc!\n";
    return 0;
}
```

categories.

Systems Used to Develop Programming Examples

For the record, the C++11 examples in this b
C++ 2010 and Cygwin with Gnu g++ 4.5.0
The remaining examples were tested with the
4.2.1 under OS X 10.6.8 and on an Ubuntu
pre-C++11 examples were originally develop
Metrowerks CodeWarrior Development Stud
sional and checked using the Borland C++ 5
3.3.3 on the same system, using Comeau 4.3.
Linux, and using Metrowerks Development S

C++ offers a lot to the programmer; learn

- The mechanics of creating a program

Welcome to C++! This exciting language
for object-oriented programming and for ge
important programming languages of the 19
ancestry brings to C++ the tradition of an e
Its object-oriented heritage brings C++ a fr
cope with the escalating complexity of mod
bring yet another new programming metho
itage is both a blessing and a bane. It makes t
there's a lot to learn.

This chapter explores C++'s background
ground rules for creating C++ programs. Th
C++ language, going from the modest basic
ented programming (OOP) and its supportin
sulation, data hiding, polymorphism, and inh
generic programming. (Of course, as you lea
from buzzwords to the necessary vocabulary

good review. Also it points out concepts that v
where C++ differs from C. After you have a g
learn about the C++ superstructure. At that p
and how C++ implements them. And you wi

This book is not intended to be a complet
nook and cranny of the language. But you wi
language, including templates, exceptions, and

Now let's take a brief look at some of C+

The Origins of C++: A Litt

Computer technology has evolved at an amaz
notebook computer can compute faster and s
computers of the 1960s. (Quite a few progran
of punched cards to be submitted to a mighty
majestic 100KB of memory—far less memory
puter languages have evolved, too. The change
important. Bigger, more powerful computers
which, in turn, raise new problems in progran

In the 1970s, languages such as C and Pasc
gramming, a philosophy that brought some o
of these qualities. Besides providing the tools

particular computer processor. So if you wan
kind of computer, you may have to complete
assembly language. It was a bit as if each time
designers decided to change where the contr
relearn how to drive.

But Unix was intended to work on a vari
suggested using a high-level language. A *high-*
solving instead of toward specific hardware. S
high-level language to the internal language
the same high-level language program on dif
for each platform. Ritchie wanted a language
hardware access with high-level generality an
languages, he created C.

C Programming Philosophy

Because C++ grafts a new programming phi
at the older philosophy that C follows. In ge
concepts—data and algorithms. The *data* con
processes. The *algorithms* are the methods the
mainstream languages when C was created, C
emphasizes the algorithm side of programmi

Figure 1.1 Data + a

Earlier procedural languages, such as FOR
problems as programs grew larger. For exampl
which route execution to one or another set
some sort of test. Many older programs had su
gramming") that it was virtually impossible to
modifying such a program was an invitation t
developed a more disciplined style of program
includes features to facilitate this approach. Fo
branching (choosing which instruction to do
structions. C incorporates these constructions
`while` loop, and the `if else` statement) into

Top-down design was another of the new p
program into smaller, more manageable tasks.
divide it into yet smaller tasks. You continue v
partmentalized into small, easily programmed
organize your desk, your table top, your filing
start with the desk and organize each drawer,
haps I can manage that task.) C's design facilit

height and width, the color and style of the b
to fill the rectangle. The operations part of th
moving the rectangle, resizing it, rotating it, c
rectangle to another location. If you then use
create an object according to the class specifi
values describing the rectangle, and you coul
tangle. If you drew two rectangles, the progra
rectangle.

The OOP approach to program design is
sent those things with which the program de
define classes to represent rectangles, lines, cir
definitions, recall, include a description of pe
moving a circle or rotating a line. Then you v
objects of those classes. The process of going
classes, to a higher level, such as program des

There's more to OOP than the binding o
example, OOP facilitates creating reusable co
work. Information hiding safeguards data fro
create multiple definitions for operators and
determining which definition is used. Inheri
ones. As you can see, OOP introduces many
to programming than does procedural progra

type. Generic programming involves extendin
function for a generic (that is, an unspecified)
types. C++ templates provide a mechanism fo

The Genesis of C++

Like C, C++ began its life at Bell Labs, where
in the early 1980s. In Stroustrup's own words,
friends and I would not have to program in as
languages. Its main purpose was to make writ
for the individual programmer" (Bjarne Strou
Edition. Reading, MA: Addison-Wesley, 1997)

Bjarne Stroustrup's Home Page

Bjarne Stroustrup designed and implemented
author of the definitive reference manuals *Th*
and Evolution of C++. His personal website at
bookmark, or favorite, you create:

www.research.att.com/~bs

This site includes an interesting historical pe
guage, Stroustrup's biographical material, and
quently asked question may be how to pronou
Stroustrup's website and download the `.WAV`

OOP onto C, you can ignore C++'s object–
that's all you do.

Only after C++ achieved some success di
programming. And only after the template fe
become apparent that templates were perhaps
more significant, some would argue. The fact
generic programming, as well as the more tra
that C++ emphasizes the utilitarian over the
reasons for the language's success.

Portability and Standards

Say you've written a handy C++ program fo
Windows 2000 at work, but management de
computer using a different operating system,
different processor design, such as a SPARC
new platform? Of course you'll have to reco
designed for the new platform. But will you
wrote? If you can recompile the program wit
hitch, we say the program is *portable*.

Figure 1.2

There are a couple obstacles to portability, that is hardware specific is not likely to be por IBM PC video board, for example, speaks gibl can minimize portability problems by localizi modules; then you just have to rewrite those s programming in this book.

The second obstacle to portability is langu lem with spoken languages. A Yorkshireman's portable to Brooklyn, even though English is languages, too, can develop dialects. Although their versions of C++ compatible with others standard describing exactly how the language Standards Institute (ANSI) created a committe dard for C++. (ANSI had already developed a zation for Standardization (ISO) soon joined t (ISO-WG-21), creating a joint ANSI/ISO eff

and the corresponding rules for C++, but th
some features first introduced in C++, such a
qualifier.

Prior to the emergence of ANSI C, the C
based on the book *The C Programming Langua*
Publishing Company, Reading, MA, 1978). Th
the emergence of ANSI C, the simpler K&R

The ANSI C Standard not only defines th
library that ANSI C implementations must s
refers to it as the *standard C library* or the *stan*
dard provides a standard library of C++ class

The C Standard was last revised as C99, w
ANSI in 2000. This standard adds some featu
some C++ compilers support.

Language Growth

Originally, the de facto standard for C++ wa
328-page *The C++ Programming Language*, by

The next major published de facto standa
by Ellis and Stroustrup (Addison-Wesley, 199
stantial commentary in addition to reference

to creating programs.

The Mechanics of Creatin

Suppose you've written a C++ program. How
depend on your computer environment and t
they should resemble the following steps (see

1. Use a text editor of some sort to write
   constitutes the *source code* for your progr
2. Compile the source code. This means ru
   code to the internal language, called *ma*
   The file containing the translated progr
3. Link the object code with additional co
   use *libraries*. A C++ library contains obj
   tines, called *functions,* to perform tasks su
   calculating the square root of a number.
   object code for the functions you use an
   duce a runtime version of your program
   called the *executable code*.

You will encounter the term *source code* thr
in your personal random-access memory.

Most of the programs in this book are generic and s
ports C++98. However, some, particularly those in Cha
support. At the time of this writing, some compilers req
their partial C++11 support. For instance, g++, beginn
the `–std=c++0x` flag when compiling a source code file

```
g++ -std=c++0x use_auto.cpp
```

The steps for putting together a program may vary. Le

Creating the Source Code File

The rest of the book deals with what goes into a source
mechanics of creating one. Some C++ implementation
Embarcadero C++ Builder, Apple Xcode, Open Watco
Freescale CodeWarrior, provide *integrated development en*
age all steps of program development, including editing
implementations, such as GNU C++ on Unix and Lin
the free versions of the Borland 5.5 (distributed by Em
pilers, just handle the compilation and linking stages an
on the system command line. In such cases, you can use
and modify source code. On a Unix system, for exampl
`emacs`. On a Windows system running in the Comman

The extension you use depends on the C+
common choices. For example, `spiffy.C` is a
Note that Unix is case sensitive, meaning you
ally, a lowercase `c` extension also works, but st
confusion on Unix systems, you should use `c`
grams. If you don't mind typing an extra char
extensions with some Unix systems. DOS, bei
doesn't distinguish between uppercase and lov
tional letters, as shown in Table 1.1, to distingu

Table 1.1 **Source Code Extensions**

C++ Implementation	Source Code
Unix	`C`, `cc`, `cxx`, `c`
GNU C++	`C`, `cc`, `cxx`, `cp`
Digital Mars	`cpp`, `cxx`
Borland C++	`cpp`
Watcom	`cpp`
Microsoft Visual C++	`cpp`, `cxx`, `cc`
Freestyle CodeWarrior	`cpp`, `cp`, `cc`, `c`

compiler being invoked differing from systen
CC is available, but realize that you might hav
the following discussion.

You use the CC command to compile you
distinguish it from the standard Unix C com
compiler, meaning you type compilation con

For example, to compile the C++ source
command at the Unix prompt:

```
CC spiffy.C
```

If, through skill, dedication, or luck, your
an object code file with an o extension. In th
spiffy.o.

Next, the compiler automatically passes th
gram that combines your code with library c
default, the executable file is called a.out. If
deletes the spiffy.o file because it's no long
the name of the executable file:

```
a.out
```

Note that if you compile a new program,
previous a.out. (That's because executable fi
cutable files helps reduce storage demands.) E

be installed. The g++ compiler works much lik
the following produces an executable file call

```
g++ spiffy.cxx
```

Some versions might require that you link

```
g++ spiffy.cxx -lg++
```

To compile multiple source files, you just li

```
g++ my.cxx precious.cxx
```

This produces an executable file called a.o
precious.o. If you subsequently modify just
can recompile by using my.cxx and the preci

```
g++ my.cxx precious.o
```

The GNU compiler is available for many p
mode for Windows-based PCs as well as for U

Command-Line Compilers for Windows C

An inexpensive route for compiling C++ pro
free command-line compiler that runs in Win
opens an MS-DOS-like window. Free Windo
compiler are Cygwin and MinGW; they use g

cation. Some of these may be available in bo

Because the programs in this book are ger platform-specific code, such as Windows appl acter-based mode. The choice depends on th if there is an option labeled Console, characte instance, in Microsoft Visual C++ 2010, selec click Application Settings, and select the Emp Console Application under C++Builder Proj

After you have the project set up, you hav typically gives you several choices, such as Co Run, and Debug (but not necessarily all thes

- *Compile* typically means compile the co
- *Build* or *Make* typically means compile project. This is often an incremental pr you change just one, and then just that
- *Build All* typically means compile all th
- As described earlier, *Link* means combi sary library code.
- *Run* or *Execut*e means run the program earlier steps, Run does them before try
- *Debug* means run the program with the

Occasionally, compilers get confused after inc
giving meaningless error messages that cann
up by selecting Build All to restart the proces
distinguish this situation from the more comm
seem to be meaningless.

Usually, the IDE lets you run the program
the window as soon as the program finishes e
compiler closes the window, you'll have a hard
quick eyes and a photographic memory. To se
code at the end of the program:

```
    cin.get();  // add this statement
    cin.get();  // and maybe this, too
    return 0;
}
```

The `cin.get()` statement reads the next k
gram to wait until you press the Enter key. (N
press Enter, so there's no point in pressing and
the program otherwise leaves an unprocessed
ple, if you enter a number, you type the numb
the number but leaves the Enter keystroke un
`cin.get()`.

programming and generic programming. Thi
tates the creation of reusable code, which sav

The popularity of C++ has resulted in a l
computing platforms; the C++ ISO standard
for keeping these many implementations mu
the features the language should have, the be
standard library of functions, classes, and tem
portable language across different computing
the language.

To create a C++ program, you create one
as expressed in the C++ language. These are
to produce the machine-language files that c
are often accomplished in an IDE that provid
compiler and a linker for producing executab
management and debugging capabilities. But
command-line environment by invoking the

- Placing comments in a C++ program
- How and when to use `endl`
- Declaring and using variables
- Using the `cin` object for input
- Defining and using simple functions

When you construct a simple home, you b
work. If you don't have a solid structure from
ing in the details, such as windows, door fram
ballrooms. Similarly, when you learn a compu
the basic structure for a program. Only then
and objects. This chapter gives you an overvi
gram and previews some topics—notably fun
detail in later chapters. (The idea is to introd
ually en route to the great awakenings that c

C++ Initiation

Let's begin with a simple C++ program that
`cout` (pronounced "see-out") facility to prod
includes several comments to the reader; thes
ignores them. C++ is *case sensitive*; that is, it d

Program Adjustments

You might find that you must alter the exampl
most common reason is a matter of the progr
ronments run the program in a separate wind
when the program finishes. As discussed in C
until you strike a key by adding the following l

```
cin.get();
```

For some programs you must add two of thes
press a key. You'll learn more about `cin.get`

If you have a very old system, it may not supp

Some programs require a compiler with some
will be clearly identified and, if possible, alter

After you use your editor of choice to cop
files available online from this book's web pag
cover for more information), you can use you
code, as Chapter 1 outlines. Here is the outpu
Listing 2.1:

```
Come up and C++ me some time.
You won't regret it!
```

place to start because some of the features tha
directive, are simpler to understand after you

Features of the `main()` Function

Stripped of the trimmings, the sample progra
fundamental structure:

```
int main()
{
    statements
    return 0;
}
```

These lines state that there is a function ca
function behaves. Together they constitute a *f*
parts: the first line, `int main()`, which is call
enclosed in braces ({ and }), which is the *fun*
braces also go by other names, including "cur
ets," and "chicken lips." However, the ISO St
shows the `main()` function. The function hea
interface with the rest of the program, and th
the computer about what the function shoul
called a *statement*. You must terminate each st
semicolons when you type the examples.

ment separator. FORTRAN, for example, uses
from the next. Pascal uses a semicolon to se
you can omit the semicolon in certain cases,
when you aren't actually separating two state
agree about whether *can* implies *should.*) But
rather than as a separator. The difference is t
of the statement rather than a marker *betwee*
C++ you should never omit the semicolon.

The Function Header as an Interface

Right now the main point to remember is tha
definition of the `main()` function with this he
function header syntax in more detail later, in
can't put their curiosity on hold, here's a previ

In general, a C++ function is activated, or
header describes the interface between a func
preceding the function name is called the *func*
from a function back to the function that calls
ing the function name is called the *argument li*
flow from the calling function to the called fu
fusing when you apply it to `main()` because y
parts of your program. Typically, however, `mai`
piler adds to your program to mediate betwee

int. However, C++ has phased out that usag
You can also use this variant:

```
int main(void)     // very explicit style
```

Using the keyword void in the parenthese
tion takes no arguments. Under C++ (but no
same as using void in the parentheses. (In C,
remaining silent about whether there are argu
Some programmers use this header and or

```
void main()
```

This is logically consistent because a void
return a value. However, although this varian
C++ Standard. Thus, on other systems it fails
C++ Standard form; it doesn't require that m
Finally, the ISO C++ Standard makes a co
tiresome necessity of having to place a return
piler reaches the end of main() without enco
be the same as if you ended main() with this

```
return 0;
```

This implicit return is provided only for m

grammer to the reader that usually identifies a
aspect of the code. The compiler ignores com
well as you do, and, in any case, it's incapable c
compiler is concerned, Listing 2.1 looks as if i

```
#include <iostream>
int main()
{
    using namespace std;
    cout << "Come up and C++ me some time.
    cout << endl;
    cout << "You won't regret it!" << endl
    return 0;
}
```

C++ comments run from the // to the en
line, or it can be on the same line as code. Inc

```
// myfirst.cpp -- displays a message
```

In this book all programs begin with a con
code and a brief program summary. As mentio
for source code depends on your C++ system
myfirst.cxx for names.

There are some alternatives to using the s
now. (If your compiler doesn't like these line
many other problems with the examples in t
make your programs work, but now let's take

C++, like C, uses a *preprocessor*. This is a p
main compilation takes place. (Some C++ i
Chapter 1, use a translator program to conve
tor is also a form of preprocessor, we're not d
cussing the one that handles directives whose
anything special to invoke this preprocessor.
the program.

Listing 2.1 uses the `#include` directive:

```
#include <iostream>    // a PREPROCESSOR
```

This directive causes the preprocessor to a
program. This is a typical preprocessor action
before it's compiled.

This raises the question of why you shoul
the program. The answer concerns communi
world. The `io` in `iostream` refers to *input*, w
gram, and to *output*, which is information se
scheme involves several definitions found in

the C++ version of math.h is the cmath head
of C header files are identical, whereas in othe
changes. For purely C++ header files such as
cosmetic change, for the h-free header files als
this chapter. Table 2.1 summarizes the naming

Table 2.1 **Header File Naming Conventions**

Kind of Header	Convention	Example
C++ old style	Ends in `.h`	`iostream`
C old style	Ends in `.h`	`math.h`
C++ new style	No extension	`iostream`
Converted C	`c` prefix, no extension	`cmath`

In view of the C tradition of using differen
file types, it appears reasonable to have some s
indicate C++ header files. The ANSI/ISO co
ing on which extension to use, so eventually t

```
fishing::wanda("a fish named Desire"); //
```

In this spirit, the classes, functions, and var
compilers are now placed in a namespace cal
files. This means, for example, that the `cout` v
`iostream` is really called `std::cout` and that
the `using` directive and, instead, code in the f

```
std::cout << "Come up and C++ me some tim
std::cout << std::endl;
```

However, many users don't feel like conve
`iostream.h` and `cout`, to namespace code, w
they can do so without a lot of hassle. This is
lowing line means you can use names defined
`std::` prefix:

```
using namespace std;
```

This `using` directive makes all the names
tice regards this as a bit lazy and potentially a
approaches are to use the `std::` qualifier or t
to make just particular names available:

```
using std::cout;   // make cout available
using std::endl;   // make endl available
using std::cin;    // make cin available
```

as you might remember from Chapter 1, is a p
defines how data is stored and used.)

Well, using objects so soon is a bit awkwar
several more chapters. Actually, this reveals on
to know the innards of an object in order to u
that is, how to use it. The `cout` object has a si
you can do the following to display it:

```
cout << string;
```

This is all you must know to display a strin
conceptual view represents the process. In thi
of characters flowing from the program. The c
the `iostream` file, represents that stream. The
tion operator (`<<`) that inserts the informatio
following statement (note the terminating se

```
cout << "Come up and C++ me some time.";
```

It inserts the string "Come up and C++ m
rather than say that your program displays a m
into the output stream. Somehow, that sound

Figure 2.2 Using c

A First Look at Operator Overloading

If you're coming to C++ from C, you probably
just like the bitwise left-shift operator (<<). T
which the same operator symbol can have di
text to figure out which meaning is intended.
example, the & symbol represents both the a
The * symbol represents both multiplication
point here is not the exact function of these
more than one meaning, with the compiler d
(You do much the same when you determine
"sound financial basis.") C++ extends the op
fine operator meanings for the user-defined t

The Manipulator `endl`

Now let's examine an odd-looking notation
in Listing 2.1:

```
cout << endl;
```

`endl` is a special C++ notation that repres
new line. Inserting `endl` into the output strea
beginning of the next line. Special notations

The Newline Character

C++ has another, more ancient, way to indica

```
cout << "What’s next?\n";    // \n means s
```

The `\n` combination is considered to be a sing

If you are displaying a string, you need less string than to tag an `endl` onto the end:

```
cout << "Pluto is a dwarf planet.\n";
cout << "Pluto is a dwarf planet." << endl
```

On the other hand, if you want to generate the same amount of typing, but most people f comfortable:

```
cout << "\n";   // start a new line
cout << endl;   // start a new line
```

Typically, this book uses an embedded new strings and the `endl` manipulator otherwise. O output will be *flushed* (in, this case, immediatel moves on. You don’t get that guarantee with "

```
"Come up and C++ me some time."
;    cout <<
endl; cout <<
"You won't regret it!" <<
endl;return 0; }
```

This is visually ugly but valid code. You do
C and C++ you can't put a space, tab, or carr
as a name, nor can you place a carriage return
of what you can't do:

```
int ma  in()     // INVALID -- space in n
re
turn 0; // INVALID -- carriage return in
cout << "Behold the Beans
 of Beauty!"; // INVALID -- carriage retu
```

(However, the *raw* string, added by C++1
allow including a carriage return in a string.)

Tokens and White Space in Source Code

The indivisible elements in a line of code are
must separate one token from the next with a
tively are termed *white space*. Some single cha

```
return0;            // INVALID, must be ret
return(0);          // VALID, white space c
return (0);         // VALID, white space u
intmain();          // INVALID, white space
int main()          // VALID, white space c
int main ( )        // ALSO VALID, white sp
```

C++ Source Code Style

Although C++ gives you much formatting fr
if you follow a sensible style. Having valid but
Most programmers use styles similar to that o

- One statement per line
- An opening brace and a closing brace fo
- Statements in a function indented from
- No whitespace around the parentheses a

The first three rules have the simple intent
fourth helps to differentiate functions from so
that also use parentheses. This book alerts you

```
    cout << carrots;        // display th
    cout << " carrots.";
    cout << endl;
    carrots = carrots - 1;  // modify the
    cout << "Crunch, crunch. Now I have "
    return 0;
}
```

A blank line separates the declaration fron
usual C convention, but it's somewhat less co
for Listing 2.2:

```
I have 25 carrots.
Crunch, crunch. Now I have 24 carrots.
```

The next few pages examine this program

Declaration Statements and Varia

Computers are precise, orderly machines. To s
you must identify both the storage location a
information requires. One relatively painless
statement to indicate the type of storage and t
ple, the program in Listing 2.2 has this declar

```
int carrots;
```

short term. The problem is that if you misspe
create a new variable without realizing it. Tha
following:

```
CastleDark = 34
...
CastleDank = CastleDark + MoreGhosts
...
PRINT CastleDark
```

Because `CastleDank` is misspelled (the *r* wa
leave `CastleDark` unchanged. This kind of e
rules in BASIC. However, in C++, `CastleDar`
`CastleDank` would not be declared. Therefor
about the need to declare a variable for you t
stomps the potential bug.

In general, then, a declaration indicates the
program will use for the data that's stored the
a variable called `carrots` in which it can stor

The declaration statement in the program
definition, for short. This means that its presenc
space for the variable. In more complex situat
These tell the computer to use a variable that
eral, a declaration need not be a definition, bu

Tip

The C++ style for declaring variables is to de
possible.

Assignment Statements

An assignment statement assigns a value to a
statement assigns the integer 25 to the locatio

```
carrots = 25;
```

The = symbol is called the *assignment opera*
that you can use the assignment operator seri

```
int steinway;
int baldwin;
int yamaha;
yamaha = baldwin = steinway = 88;
```

The assignment works from right to left. F
value of steinway, which is now 88, is assign
assigned to yamaha. (C++ follows C's pencha

The second assignment statement in Listin
value of a variable:

```
carrots = carrots - 1;  // modify the var
```

Bases," discusses this representation.) The mai
number in integer form into character form b
smart enough to recognize that `carrots` is an

Perhaps the contrast with old C will indica
`"25"` and the integer `25` in C, you could use C

```
printf("Printing a string: %s\n", "25");
printf("Printing an integer: %d\n", 25);
```

Without going into the intricacies of `prin`
(`%s` and `%d`) to indicate whether you are going
tell `printf()` to print a string but give it an i
ticated to notice your mistake. It just goes ahe

The intelligent way in which `cout` behave
In essence, the C++ insertion operator (`<<`) a
follows it. This is an example of operator over
function overloading and operator overloading
designs yourself.

`cout` and `printf()`

If you are used to C and `printf()`, you migh
to cling to your hard-won mastery of `printf(`
ance than `printf()`, with all its conversion s
nificant advantages. Its capability to recogniz

```
    cout << "How many carrots do you have
    cin >> carrots;              // C++
    cout << "Here are two more. ";
    carrots = carrots + 2;
// the next line concatenates output
    cout << "Now you have " << carrots <<
    return 0;
}
```

Program Adjustments

If you found that you had to add a `cin.get(` to add two `cin.get()` statements to this lis onscreen. The first one will read the newline key after typing a number, and the second wi Return or Enter again.

Here is an example of output from the pr

```
How many carrots do you have?
12
Here are two more. Now you have 14 carrot
```

The program has two new features: using four output statements into one. Let's take a l

The second new feature of `getinfo.cpp` is co
The `iostream` file defines the `<<` operator so
output as follows:

```
cout << "Now you have " << carrots << " ca
```

This allows you to combine string output
resulting output is the same as what the follow

```
cout << "Now you have ";
cout << carrots;
cout << " carrots";
cout << endl;
```

While you're still in the mood for `cout` ad
version this way, spreading the single statemen

```
cout << "Now you have "
     << carrots
     << " carrots."
     << endl;
```

That's because C++'s free format rules trea
changeably. This last technique is convenient
Another point to note is that

```
Now you have 14 carrots.
```

you've been exposed to different OOP termi
class corresponds to what some languages ter
sponds to an object instance or instance varia

Now let's get a little more specific. Recall

```
int carrots;
```

This creates a particular variable (carrots
is, carrots can store an integer and can be u
traction, for example. Now consider cout. It
the ostream class. The ostream class definitio
describes the sort of data an ostream object
form with and to it, such as inserting a numb
cin is an object created with the properties o
iostream.

> **Note**
>
> The class describes all the properties of a da
> formed with it, and an object is an entity cre

You have learned that classes are user-defi
design the ostream and istream classes. Just
classes can come in class libraries. That's the c
Technically, they are not built in to the C++

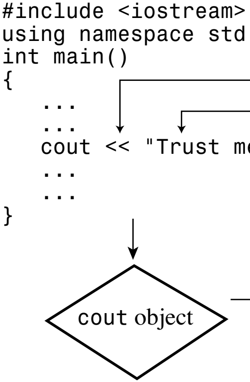


Figure 2.5 Sending a

Functions

Because functions are the modules from whic
are essential to C++ OOP definitions, you sh
them. Some aspects of functions are advanced
comes later, in Chapter 7, "Functions: C++'s

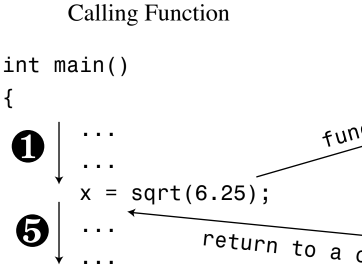


Figure 2.6 C

The value in the parentheses (6.25, in this
function; it is said to be *passed* to the functio
called an *argument* or *parameter* (see Figure 2.7
to be 2.5 and sends that value back to the cal
the *return value* of the function. Think of the
function call in the statement after the functi
the return value to the variable x. In short, ar
tion, and the return value is a value sent back

is to use a function prototype statement.

> **Note**
>
> A C++ program should provide a prototype fo

A function prototype does for functions w
It tells what types are involved. For example, t
to take a number with (potentially) a fraction
return a number of the same type. Some langu
but the name C++ uses for this type is `doubl`
The function prototype for `sqrt()` looks like

```
double sqrt(double);   // function prototy
```

The initial `double` means `sqrt()` returns a
parentheses means `sqrt()` requires a `double` a
`sqrt()` exactly as used in the following code:

```
double x;       // declare x as a type do
x = sqrt(6.25);
```

The terminating semicolon in the prototyp
it a prototype instead of a function header. If y
prets the line as a function header and expects
defines the function.

usual practice is to place prototypes just befo
Listing 2.4 demonstrates the use of the librar
including the `cmath` file.

Listing 2.4 **sqrt.cpp**

```
// sqrt.cpp -- using the sqrt() function

#include <iostream>
#include <cmath>    // or math.h

int main()
{
    using namespace std;

    double area;
    cout << "Enter the floor area, in squ
    cin >> area;
    double side;
    side = sqrt(area);
    cout << "That's the equivalent of a s
         << " feet to the side." << endl;
    cout << "How fascinating!" << endl;
    return 0;
}
```

```
That's the equivalent of a square 39.1918
How fascinating!
```

Because `sqrt()` works with type `double` v
type. Note that you declare a type `double` var
when you declare a type `int` variable:

```
type-name variable-name;
```

Type `double` allows the variables `area` and
such as `1536.0` and `39.1918`. An apparent inte
with a decimal fraction part of `.0` when store
Chapter 3, type `double` encompasses a much

C++ allows you to declare new variables a
declare `side` until just before using it. C++ al
when you create it, so you could also have do

```
double side = sqrt(area);
```

You'll learn more about this process, called

Note that `cin` knows how to convert infor
`double`, and `cout` knows how to insert type d
lier, these objects are smart.

the function call even if there are no argume

There also are functions that have no retu
function that displayed a number in dollars-a
argument of, say, 23.5, and it would display $
value to the screen instead of to the calling p
indicate this in the prototype by using the ke

```
void bucks(double);  // prototype for fun
```

Because `bucks()` doesn't return a value, y
assignment statement or of some other expre
statement:

```
bucks(1234.56);      // function call, no
```

Some languages reserve the term *function* f
terms *procedure* or *subroutine* for those withou
term *function* for both variations.

User-Defined Functions

The standard C library provides more than 1
needs, by all means use it. But often you have
design classes. Anyway, it's fun to design your
process. You've already used several user-defin
`main()`. Every C++ program must have a `ma`

```
}

void simon(int n)   // define the simon()
{
    using namespace std;
    cout << "Simon says touch your toes "
}                   // void functions don'
```

The main() function calls the simon() fur and once with a variable argument count. In used to set the value of count. The example d prompting message. This results in the user inp prompt. Here is a sample run of the program

```
Simon says touch your toes 3 times.
Pick an integer: 512
Simon says touch your toes 512 times.
Done!
```

Function Form

The definition for the simon() function in Li the definition for main(). First, there is a func the function body. You can generalize the forr function #3

```
do
{

}
```

Figure 2.8 Funct
sequentia

Function Headers

The `simon()` function in Listing 2.5 has this

```
void simon(int n)
```

The initial `void` means that `simon()` has r
produce a number that you can assign to a va
looks like this:

```
simon(3);            // ok for void funct
```

Because poor `simon()` lacks a return valu

```
simple = simon(3);   // not allowed for v
```

The `int n` within the parentheses means
single argument of type `int`. The `n` is a new v

all, nowhere in any of your programs have yo

```
squeeze = main();   // absent from our pro
```

The answer is that you can think of your c Windows) as calling your program. So main() part of the program but to the operating syste gram's return value. For example, Unix shell s batch files can be designed to run programs ar *values*. The normal convention is that an exit v fully, whereas a nonzero value means there wa program to return a nonzero value if, say, it fa script or batch file to run that program and to signals failure.

Keywords

Keywords are the vocabulary of a computer la words: `int`, `void`, `return`, and `double`. Bec can't use them for other purposes. That is, yo or `double` as the name of a function. But you `painter` (with its hidden `int`) or `return_ac` vides a complete list of C++ keywords. Incide part of the language. Instead, it is the name o variable name. (That can cause a problem in and because it is confusing in any case, you'd

```
    using namespace std;
    int stone;
    cout << "Enter the weight in stone: "
    cin >> stone;
    int pounds = stonetolb(stone);
    cout << stone << " stone = ";
    cout << pounds << " pounds." << endl;
    return 0;
}

int stonetolb(int sts)
{
     return 14 * sts;
}
```

Here's a sample run of the program in List

```
Enter the weight in stone: 15
15 stone = 210 pounds.
```

In `main()`, the program uses `cin` to provid
value is passed to the `stonetolb()` function
`sts` in that function. `stonetolb()` then uses
`14 * sts` to `main()`. This illustrates that you
simple number. Here, by using a more compl

sort of information goes into the function, an
returned. Programmers sometimes describe fu
ics) specified by the flow of information into
perfectly portrays that point of view (see Figu

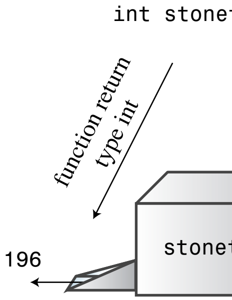


Figure 2.9 The func
function as a

The `stonetolb()` function is short and sin
tional features:

- It has a header and a body.
- It accepts an argument.

```
int main()
{
    simon(3);
    cout << "Pick an integer: ";
    int count;
    cin >> count;
    simon(count);
    cout << "Done!" << endl;
    return 0;
}

void simon(int n)
{
    cout << "Simon says touch your toes "
}
```

The current prevalent philosophy is that i
limit access to the `std` namespace to only tho
in Listing 2.6, only `main()` uses `cout`, so ther
available to the `stonetolb()` function. Thus,
`main()` function only, limiting `std` namespac

std::cout << "I'm using cout and endl

Naming Conventions

C++ programmers are blessed (or cursed) wi
classes, and variables. Programmers have st
these often surface as holy wars in public fo
function name, a programmer might select a

```
MyFunction( )
myfunction( )
myFunction( )
my_function( )
my_funct( )
```

The choice will depend on the development t
libraries used, and the tastes and preference
that any style consistent with the C++ rules p
C++ language is concerned, and it can be us

Language allowances aside, it is worth notin
you through consistency and precision—is w
sonal naming convention is a hallmark of goo
throughout your programming career.

along with the number and type of arg

- **Return statement**—A return statemen to the calling function.

A class is a user-defined specification for a information is to be represented and also the data. An object is an entity created according able is an entity created according to a data t

C++ provides two predefined objects (`ci` They are examples of the `istream` and `ostre` `iostream` file. These classes view input and o operator (`<<`), which is defined for the `ostre` stream, and the extraction operator (`>>`), whi extract information from the input stream. B of automatically converting information from gram context.

C++ can use the extensive set of C librar should include the header file that provides t

Now that you have an overall view of sim next chapters to fill in details and expand ho

9. What do the following function prototy

```
int froop(double t);
void rattle(int n);
int prune(void);
```

10. When do you not have to use the keyw

11. Suppose your `main()` function has the

```
cout << "Please enter your PIN: ";
```

    And suppose the compiler complains th
    the likely cause of this complaint, and w

Programming Exercises

1. Write a C++ program that displays you
   privacy, a fictitious name and address).

2. Write a C++ program that asks for a di
   (One furlong is 220 yards.)

For reference, here is the formula for m

Fahrenheit = 1.8 degrees Celsius + 3

6. Write a program that has `main()` call a in light years as an argument and then The program should request the light y the result, as shown in the following co

```
Enter the number of light years: 4.
4.2 light years = 265608 astronomic
```

An astronomical unit is the average dis 150,000,000 km or 93,000,000 miles), a year (about 10 trillion kilometers or sun is about 4.2 light years away.) Use version factor:

1 light year = 63,240 astronomical uni

7. Write a program that asks the user to e `main()` function should then pass these plays the two values in the format show

```
Enter the number of hours: 9
Enter the number of minutes: 28
Time: 9:28
```

- Numeric literals (constants) of various
- Using the `const` qualifier to create sym
- C++'s built-in floating-point types: `fl`
- The `cfloat` file, which represents syste
- Numeric literals of various floating-po
- C++'s arithmetic operators
- Automatic type conversions
- Forced type conversions (type casts)

The essence of object-oriented programm own data types. Designing your own data typ the data. If you do this properly, you'll find it But before you can create your own types, yo are built in to C++ because those types will

The built-in C++ types come in two gro In this chapter you'll meet the fundamental t point numbers. That might sound like just tw one integer type and no one floating-point t it offers several variants on these two data the up by covering several types that are built on types include arrays, strings, pointers, and stru

```
int braincount;
braincount = 5;
```

These statements tell the program that it is
braincount represents the integer's value, 5 i
chunk of memory large enough to hold an in
value 5 into the location. You then can use br
that memory location. These statements don't
value is stored, but the program does keep tra
use the & operator to retrieve braincount's a
operator in the next chapter, when you invest
data—using pointers.

Names for Variables

C++ encourages you to use meaningful nam
cost of a trip, you should call it cost_of_tri
have to follow a few simple C++ naming rul

- The only characters you can use in nam
  and the underscore (_) character.
- The first character in a name cannot be
- Uppercase characters are considered dis

```
int FOOBLE;    // valid and even more dis
Int terrier;   // invalid -- has to be in
int my_stars3  // valid
int _Mystars3; // valid but reserved -- s
int 4ever;     // invalid because starts
int double;    // invalid -- double is a
int begin;     // valid -- begin is a Pas
int __fools;   // valid but reserved -- s
int the_very_best_variable_i_can_be_versi
int honky-tonk;        // invalid -- no h
```

If you want to form a name from two or
the words with an underscore character, as in
acter of each word after the first, as in myEye
score method in the C tradition, whereas tho
capitalization approach.) Either form makes i
distinguish between, say, carDrip and cardR

Naming Schemes

Schemes for naming variables, like schemes
fervid discussion. Indeed, this topic produce
programming. Again, as with function names
able names as long as they satisfy the rules
vention will serve you well.

computer memory can represent all possible
a subset of all integers. Some languages offer j
C++ provides several choices. This gives you
best meets a program's particular requirement
presages the designed data types of OOP.

The various C++ integer types differ in th
integer. A larger block of memory can represe
some types (signed types) can represent both
(unsigned types) can't represent negative value
of memory used for an integer is *width*. The m
C++'s basic integer types, in order of increasi
with C++11, `long long`. Each comes in both
you a choice of ten different integer types! Le
Because the `char` type has some special prope
acters instead of numbers), this chapter covers

The `short`, `int`, `long`, and `long`

Computer memory consists of units called *bit*
this chapter.) By using different numbers of b
`int`, `long`, and `long long` can represent up to
convenient if each type were always some par
`short` were always 16 bits, `int` were always 3

A *byte* usually means an 8-bit unit of memory
that describes the amount of memory in a co
and a megabyte equal to 1,024 kilobytes. Ho
byte consists of at least enough adjacent bit
the implementation. That is, the number of p
ber of distinct characters. In the United State
ASCII and EBCDIC sets, each of which can be
typically 8 bits on systems using those chara
can require much larger character sets, such
use a 16-bit byte or even a 32-bit byte. Some

Many systems currently use the minimum
32 bits. This still leaves several choices open f
and meet the standard. It could even be 64 bi
least that wide. Typically, `int` is 16 bits (the sa
tations and 32 bits (the same as `long`) for Win
tosh OS X, VAX, and many other minicompu
give you a choice of how to handle `int`. (Wh
example shows you how to determine the lin
open a manual.) The differences between imp
problems when you move a C++ program fr
using a different compiler on the same system
chapter, can minimize those problems.

variable.

Listing 3.1 **limits.cpp**

```
// limits.cpp -- some integer limits
#include <iostream>
#include <climits>              // use lim
int main()
{
    using namespace std;
    int n_int = INT_MAX;        // initial
    short n_short = SHRT_MAX;   // symbols
    long n_long = LONG_MAX;
    long long n_llong = LLONG_MAX;

    // sizeof operator yields size of type
    cout << "int is " << sizeof (int) << "
    cout << "short is " << sizeof n_short
    cout << "long is " << sizeof n_long <<
    cout << "long long is " << sizeof n_ll
    cout << endl;

    cout << "Maximum values:" << endl;
    cout << "int: " << n_int << endl;
    cout << "short: " << n_short << endl;
```

```
long: 2147483647
long long: 9223372036854775807

Minimum int value = -2147483648
Bits per byte = 8
```

These particular values came from a system running
The following sections look at the chief programmi

The `sizeof` Operator and the `climits` Header

The `sizeof` operator reports that `int` is 4 bytes on the byte. You can apply the `sizeof` operator to a type name use the `sizeof` operator with a type name, such as `int` theses. But when you use the operator with the name c parentheses are optional:

```
cout << "int is " << sizeof (int) << " bytes.\n";
cout << "short is " << sizeof n_short << " bytes.\
```

The `climits` header file defines symbolic constants stants the Preprocessor Way," later in this chapter) to rep previously, `INT_MAX` represents the largest value type `in` 2,147,483,647 for our Windows 7 system. The compile file that reflects the values appropriate to that compiler. some older systems that used a 16-bit `int`, defines `INT_`

UINT_MAX	Maximum unsign
LONG_MAX	Maximum long v
LONG_MIN	Minimum long va
ULONG_MAX	Maximum unsign
LLONG_MAX	Maximum long l
LLONG_MIN	Minimum long l
ULLONG_MAX	Maximum unsign

Symbolic Constants the Preprocessor W

The climits file contains lines similar to the

```
#define INT_MAX 32767
```

Recall that the C++ compilation process first
sor. Here #define, like #include, is a prepr
tells the preprocessor is this: Look through th
replace each occurrence with 32767. So the
and-replace command in a text editor or word
after these replacements occur. The preproce
words) and skips embedded words. That is, th

time the program tries to initialize the other

The initialization syntax shown previously

syntax that is not shared with C:

```
int owls = 101;   // traditional C initia
int wrens(432);   // alternative C++ synt
```

Caution

If you don't initialize a variable that is define
indeterminate. That means the value is what
location prior to the creation of the variable.

If you know what the initial value of a var
the declaring of a variable from assigning it a

```
short year;       // what could it be?
year = 1492;      // oh
```

But initializing the variable when you dec
the value later.

C++ added the parentheses form of initializat
more like initializing class variables. C++11 m
(with or without the =) with all types—a univ
may introduce you to initialization using the b
historical oddities retained for backward comp

Unsigned Types

Each of the four integer types you just learne
can't hold negative values. This has the advant
able can hold. For example, if short represent
unsigned version can represent the range 0 to
types only for quantities that are never negativ
happy face manifestations. To create unsigned
use the keyword unsigned to modify the decl

```
unsigned short change;          // unsigne
unsigned int rovert;            // unsigne
unsigned quarterback;           // also un
unsigned long gone;             // unsigne
unsigned long long lang_lang;   // unsigne
```

Note that unsigned by itself is short for un

```
    sam = ZERO;
    sue = ZERO;
    cout << "Sam has " << sam << " dollar
    cout << " dollars deposited." << endl
    cout << "Take $1 from each account."
    sam = sam - 1;
    sue = sue - 1;
    cout << "Sam has " << sam << " dollar
    cout << " dollars deposited." << endl
    return 0;
}
```

Here's the output from the program in Lis

```
Sam has 32767 dollars and Sue has 32767 d
Add $1 to each account.
Now Sam has -32768 dollars and Sue has 32
Poor Sam!
Sam has 0 dollars and Sue has 0 dollars d
Take $1 from each account.
Now Sam has -1 dollars and Sue has 65535
Lucky Sue!
```

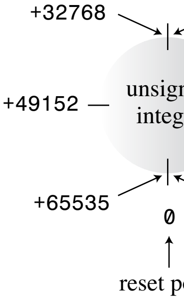
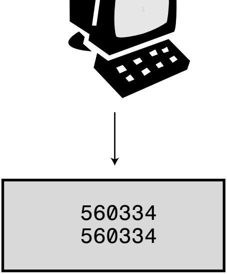


Figure 3.1 Typical overf

Choosing an Integer Type

With the richness of C++ integer types, whic
the most "natural" integer size for the target c
form that the computer handles most efficien
choose another type, you should use int.

Now look at reasons why you might use a
thing that is never negative, such as the numbe
unsigned type; that way the variable can repre



Type int worked on this computer.

Figure 3.2 For portabilit

Using short can conserve memory if sho
important only if you have a large array of in
several values of the same type sequentially in
space, you should use short instead of int, e
example, that you move your program from a
That doubles the amount of memory needed
requirements for a short array. Remember, a

If you need only a single byte, you can use

```
    int waist = 0x42;   // hexadecimal int
    int inseam = 042;   // octal integer l

    cout << "Monsieur cuts a striking figu
    cout << "chest = " << chest << " (42 i
    cout << "waist = " << waist << " (0x42
    cout << "inseam = " << inseam << " (04
    return 0;
}
```

By default, `cout` displays integers in decim
in a program, as the following output shows:

```
Monsieur cuts a striking figure!
chest = 42 (42 in decimal)
waist = 66 (0x42 in hex)
inseam = 34 (042 in octal)
```

Keep in mind that these notations are mer
you belong to a vintage PC club and read tha
in hexadecimal, you don't have to convert the
your program. Instead, you can simply use 0x
as 10, 012, or 0xA, it's stored the same way in

```
    cout << "waist = " << waist << " (hex
    cout << oct;      // manipulator for
    cout << "inseam = " << inseam << " (o
    return 0;
}
```

Here's the program output for Listing 3.4:

```
Monsieur cuts a striking figure!
chest = 42 (decimal for 42)
waist = 2a (hexadecimal for 42)
inseam = 52 (octal for 42)
```

Note that code like the following doesn't

```
cout << hex;
```

Instead, it changes the way `cout` displays i message to `cout` that tells it how to behave. A part of the `std` namespace and the program u `hex` as the name of a variable. However, if you used `std::cout`, `std::endl`, `std::hex`, and s name for a variable.

base 16; the term *decimal* does not necessarily
without a suffix is represented by the smallest
`long`, or `long long`. On a computer system u
represented as type `int`, `40000` is represented
`long long`. A hexadecimal or octal integer wi
of the following types that can hold it: `int`, `un`
`long`, or `unsigned long long`. The same co
represents the hexadecimal equivalent 0x9C4
decimal is frequently used to express memory
So `unsigned int` is more appropriate than `lo`

The `char` Type: Characters and S

It's time to turn to the final integer type: `char`
`char` type is designed to store characters, such
storing numbers is no big deal for computers,
ming languages take the easy way out by using
type is another integer type. It's guaranteed to
range of basic symbols—all the letters, digits, p
computer system. In practice, many systems su
a single byte can represent the whole range. T
handle characters, you can also use it as an int

```
    cout << "Hola! ";
    cout << "Thank you for the " << ch <<
    return 0;
}
```

Here's the output from the program in Lis

```
Enter a character:
M
Hola! Thank you for the M character.
```

The interesting thing is that you type an M
Also the program prints an M, not 77. Yet if yo
value stored in the ch variable. The magic, su
and cout. These worthy facilities make conve
verts the keystroke input M to the value 77. C
displayed character M; cin and cout are guide
same value 77 into an int variable, cout disp
characters.) Listing 3.6 illustrates this point. I
in C++: Enclose the character within two sir
the example doesn't use double quotation ma
character and double quotation marks for a s
as Chapter 4 discusses, the two are quite diffe
introduces a cout feature, the cout.put() fu

```
    cout << endl << "Done" << endl;
    return 0;
}
```

Here is the output from the program in Li

```
The ASCII code for M is 77
Add one to the character code:
The ASCII code for N is 78
Displaying char ch using cout.put(ch): N!
Done
```

Program Notes

In the program in Listing 3.6, the notation 'M
character, so initializing the char variable ch t
then assigns the identical value to the int vari
Next, cout displays ch as M and i as 77. As pre
it chooses how to display that value—just ano

Because ch is really an integer, you can app
This changes the value of ch to 78. The progr
lently, you can simply add 1 to i.) Again, cout
character and the int version as a number.

Remember that a class defines how to repres
function belongs to a class and describes a m
`ostream` class, for example, has a `put()` mem
acters. You can use a member function only v
the `cout` object, in this case. To use a class me
you use a period to combine the object nam
The period is called the *membership operator*. T
class member function `put()` with the class o
detail when you reach classes in Chapter 10,
you have are the `istream` and `ostream` classe
ber functions to get more comfortable with t

The `cout.put()` member function provi
display a character. At this point you might w
`cout.put()`. Much of the answer is historica
display character *variables* as characters but di
as numbers. The problem was that earlier ver
stants as type `int`. That is, the code `77` for `'M`
Meanwhile, `char` variables typically occupie
8 bits (the important 8 bits) from the consta

```
char ch = 'M';
```

- `'a'` is `97`, the ASCII code for `a`.
- `'5'` is `53`, the ASCII code for the digit 5
- `' '` is `32`, the ASCII code for the space
- `'!'` is `33`, the ASCII code for the exclar

Using this notation is better than using the doesn't assume a particular code. If a system u but `'A'` still represents the character.

There are some characters that you can't er board. For example, you can't make the newli Enter key; instead, the program editor interpre new line in your source code file. Other chara guage imbues them with special significance. character delimits string literals, so you can't ju C++ has special notations, called *escape sequen* in Table 3.2. For example, `\a` represents the al speaker or rings its bell. The escape sequence the double quotation mark as an ordinary cha use these notations in strings or in character c

The last line produces the following outp

```
Ben "Buggsie" Hacker
was here!
```

Note that you treat an escape sequence, su
That is, you enclose it in single quotes to cre
quotes when including it as part of a string.

The escape sequence concept dates back t
ers using the teletype, an electromechanical t
always honor the complete set of escape sequ
silent for the alarm character.

The newline character provides an alterna
put. You can use the newline character in cha
ter in a string (`"\n"`). All three of the followi
of the next line:

```
cout << endl;    // using the endl manipu
cout << '\n';    // using a character con
cout << "\n";    // using a string
```

You can embed the newline character in a
than using `endl`. For example, the following t

```
cout << endl << endl << "What next?" << e
cout << "\n\nWhat next?\nEnter a number:\
```

attention, the newline character to advance th
giant step for cursorkind), and the backspace c
left. (Houdini once painted a picture of the H
he was, of course, a great escape artist.)

Listing 3.7 **bondini.cpp**

```
// bondini.cpp -- using escape sequences
#include <iostream>
int main()
{
    using namespace std;
    cout << "\aOperation \"HyperHype\" is
    cout << "Enter your agent code:_______
    long code;
    cin >> code;
    cout << "\aYou entered " << code << ".
    cout << "\aCode verified! Proceed with
    return 0;
}
```

Note

Some systems might behave differently, displ
backspacing, for example, or perhaps erasing

allows an implementation to offer extended s
character sets. Furthermore, those additional
part of the name of an identifier. Thus, a Ger
umlauted vowels, and a French implementati
mechanism for representing such internation
ticular keyboard: the use of *universal character*

Using universal character names is similar
acter name begins either with `\u` or `\U`. The
and the `\U` form by 16 hexadecimal digits. Th
point for the character. (ISO 10646 is an inte
provides numeric codes for a wide range of c
later in this chapter.)

If your implementation supports extended
names in identifiers, as character constants, an
lowing code:

```
int k\u00F6rper;
cout << "Let them eat g\u00E2teau.\n";
```

The ISO 10646 code point for ö is 00F6,
C++ code would set the variable name to kö

```
Let them eat gâteau.
```

also incorporates other Latin characters, such
acters from other alphabets, including Greek,
Bengali; and ideographs, such as those used
resents more than 109,000 symbols and mo
ment. If you want to know more, you can chec
unicode.org.

Unicode assigns a number, called a `code po`
notation for Unicode code points looks like th
character, and the 222B is the hexadecimal r
this case.

The International Organization for Standardiza
develop ISO 10646, also a standard for codin
the Unicode group have worked together sinc
with one another.

`signed char` and `unsigned char`

Unlike `int`, `char` is not signed by default. No
to the C++ implementation in order to allow
to the hardware properties. If it is vital to you
use `signed char` or `unsigned char` explicitl

```
char fodo;              // may be signed,
unsigned char bar;      // definitely unsi
signed char snark;      // definitely sign
```

so they are not suitable for handling the `wcha`
parallel facilities in the form of `wcin` and `wco`
can indicate a wide-character constant or stri
code stores a `wchar_t` version of the letter `P`
version of the word `tall`:

```
wchar_t bob = L'P';         // a wide-cha
wcout << L"tall" << endl;   // outputting
```

On a system with a 2-byte `wchar_t`, this c
memory. This book doesn't use the wide-cha
particularly if you become involved in intern
ISO 10646.

New C++11 Types: `char16_t` and `cha`

As the programming community gained mor
that the `wchar_t` type wasn't enough. It turn
characters on a computer system is more con
values (called code points). In particular, it's u
have a type of definite size and signedness. Bu
one implementation to another. So C++11 i
unsigned and 16 bits, and `char32_t`, which is
prefix for char16_t character and string const

and false, and the predefined literals `true` and
can make statements like the following:

```
bool is_ready = true;
```

The literals `true` and `false` can be conver
converting to 1 and `false` to 0:

```
int ans = true;           // ans assigned
int promise = false;      // promise assig
```

Also any numeric or pointer value can be
explicit type cast) to a `bool` value. Any nonzer
value converts to `false`:

```
bool start = -100;        // start assigned
bool stop = 0;            // stop assigned
```

After the book introduces `if` statements (i
Logical Operators"), the `bool` type will becom

The `const` Qualifier

Now let's return to the topic of symbolic nam
gest what the constant represents. Also if the p
and you need to change the value, you can ju

The general form for creating a constant i

```
const type name = value;
```

Note that you initialize a `const` in the de

```
const int toes;    // value of toes undef
toes = 10;         // too late!
```

If you don't provide a value when you de
fied value that you cannot modify.

If your background is in C, you might fee
cussed earlier, already does the job adequately
you specify the type explicitly. Second, you c
nition to particular functions or files. (Scopin
is to different modules; you'll learn about thi
Models and Namespaces.") Third, you can us
arrays and structures, as discussed in Chapter

Tip

If you are coming to C++ from C and you are
constant, use `const` instead.

ANSI C also uses the `const` qualifier, whi
with the ANSI C version, you should be awa

except it's based on binary numbers, so the sc
of 10. Fortunately, you don't have to know m
main points are that floating-point numbers l
very small values, and they have internal repre
integers.

Writing Floating-Point Numbers

C++ has two ways of writing floating-point
mal-point notation you've been using much

```
12.34              // floating-point
939001.32          // floating-point
0.00023            // floating-point
8.0                // still floating-point
```

Even if the fractional part is 0, as in 8.0, th
represented in floating-point format and not
for implementations to represent different lo
for using the European method of using a co
point. However, these choices govern how th
not in code.)

The second method for representing floati
looks like this: `3.45E6`. This means that the va
means 10 to the 6th power, which is 1 follow

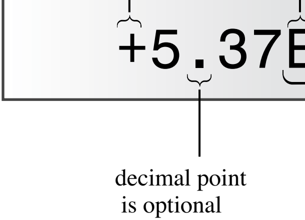


Figure 3.3

To use a negative exponent means to divi
a power of 10. So 8.33E-4 means 8.33 / 10^4,
9.11e-31 kg means 0.00000000000000000
(Incidentally, note that 911 is the usual emerg
and that telephone messages are carried by el
You be the judge.) Note that –8.33E4 means
ber value, and a sign in the exponent applies

Note

The form `d.dddE+n` means move the decima
`d.dddE-n` means move the decimal point `n`
is the origin of the term "floating-point."

```
#define DBL_DIG 15          // double
#define FLT_DIG 6           // float
#define LDBL_DIG 18         // long double

// the following are the number of bits us
#define DBL_MANT_DIG     53
#define FLT_MANT_DIG     24
#define LDBL_MANT_DIG    64

// the following are the maximum and minim
#define DBL_MAX_10_EXP   +308
#define FLT_MAX_10_EXP   +38
#define LDBL_MAX_10_EXP  +4932

#define DBL_MIN_10_EXP   -307
#define FLT_MIN_10_EXP   -37
#define LDBL_MIN_10_EXP  -4931
```

Listing 3.8 examines types `float` and `doub` to which they represent numbers (that's the si views an `ostream` method called `setf()` from This particular call forces output to stay in fix the precision. It prevents the program from sw causes the program to display six digits to the

```
}
```

Here is the output from the program in L

```
tub = 3.333333, a million tubs = 3333333.
and ten million tubs = 33333332.000000
mint = 3.333333 and a million mints = 333
```

Program Notes

Normally `cout` drops trailing zeros. For exam 3333333.25. The call to `cout.setf()` overrid tions. The main thing to note in Listing 3.8 is Both `tub` and `mint` are initialized to 10.0 / 3 3.33333333333333333...(etc.). Because `cout` you can see that both `tub` and `mint` are accur each number by a million, you see that `tub` d enth three. `tub` is good to seven significant fig figures for `float`, but that's the worst-case sce shows 13 threes, so it's good to at least 13 sig tees 15, this shouldn't surprise you. Also note doesn't quite result in the correct answer; this precision.

When you write a floating-point constant in a
the program store it? By default, floating-poin
`double`. If you want a constant to be type `flo`
`double`, you use an `l` or `L` suffix. (Because the
uppercase `L` is a better choice.) Here are some

```
1.234f          // a float constant
2.45E20F        // a float constant
2.345324E28     // a double constant
2.2L            // a long double constant
```

Advantages and Disadvantages of

Floating-point numbers have two advantages
between integers. Second, because of the scali
range of values. On the other hand, floating p
than integer operations, and you can lose prec

Listing 3.9 **fltadd.cpp**

```
// fltadd.cpp -- precision problems with f
#include <iostream>
int main()
{
```

C++ brings some order to its basic types by
`char`, `short`, `int`, and `long` are termed *sig*
that list. The unsigned versions are termed
`wchar_t`, signed integer, and unsigned integ
integer types. C++11 adds `char16_t` and `c`
`long double` types are termed *floating-poin*
lectively termed *arithmetic* types.

C++ Arithmetic Operator

Perhaps you have warm memories of doing a
that same pleasure to your computer. C++ u
operators for five basic arithmetic calculation
sion, and taking the modulus. Each of these o
calculate a final answer. Together, the operato
example, consider the following statement:

```
int wheels = 4 + 2;
```

The values `4` and `2` are operands, the `+` sy
expression whose value is 6.

```
// arith.cpp -- some C++ arithmetic
#include <iostream>
int main()
{
    using namespace std;
    float hats, heads;

    cout.setf(ios_base::fixed, ios_base::f
    cout << "Enter a number: ";
    cin >> hats;
    cout << "Enter another number: ";
    cin >> heads;

    cout << "hats = " << hats << "; heads
    cout << "hats + heads = " << hats + he
    cout << "hats - heads = " << hats - he
    cout << "hats * heads = " << hats * he
    cout << "hats / heads = " << hats / he
    return 0;
}
```

As you can see in the following sample out
can trust C++ to do simple arithmetic:

operator can be applied to the same operand,
operator is used first. The arithmetic operator
multiplication, division, and the taking of the
tion. Thus `3 + 4 * 5` means `3 + (4 * 5)`, n
Of course, you can use parentheses to enforce
Precedence," shows precedence for all the C+
the same row in Appendix D. That means the
and subtraction share a lower precedence.

Sometimes the precedence list is not enou

```
float logs = 120 / 4 * 5;    // 150 or 6?
```

Once again, 4 is an operand for two opera
precedence, so precedence alone doesn't tell t
or multiply 4 by 5. Because the first choice le
result of 6, the choice is an important one. W
dence, C++ looks at whether the operators h
left associativity. Left-to-right associativity me
operand have the same precedence, you apply
associativity, you apply the right-hand operato
in Appendix D. Appendix D shows that multi
That means you use 4 with the leftmost oper
as a result, and then multiply the result by 5 t

Listing 3.11 **divide.cpp**

```
// divide.cpp -- integer and floating-poin
#include <iostream>
int main()
{
    using namespace std;
    cout.setf(ios_base::fixed, ios_base::f
    cout << "Integer division: 9/5 = " <<
    cout << "Floating-point division: 9.0/
    cout << 9.0 / 5.0 << endl;
    cout << "Mixed division: 9.0/5 = " <<
    cout << "double constants: 1e7/9.0 = "
    cout << 1.e7 / 9.0 <<  endl;
    cout << "float constants: 1e7f/9.0f =
    cout << 1.e7f / 9.0f <<  endl;
    return 0;
}
```

Here is the output from the program in Li

```
Integer division: 9/5 = 1
Floating-point division: 9.0/5.0 = 1.80000
Mixed division: 9.0/5 = 1.800000
```

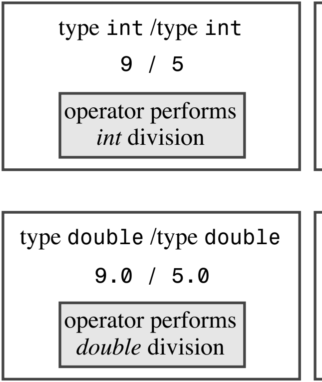


Figure 3.4 Di

The Modulus Operator

Most people are more familiar with addition
than with the modulus operation, so let's tak
in action. The modulus operator returns the
nation with integer division, the modulus op
require dividing a quantity into different inte
and inches or converting dollars to quarters,

```
}
```

Here is a sample run of the program in Lis

```
Enter your weight in pounds: 181
181 pounds are 12 stone, 13 pound(s).
```

In the expression `lbs / Lbs_per_stn`, bo
performs integer division. With a `lbs` value o
product of 12 and 14 is 168, so the remainder
value of `lbs % Lbs_per_stn`. Now you are p
respond to questions about your weight when

Type Conversions

C++'s profusion of types lets you match the t
the computer. For example, adding two `short`
instructions than adding two `long` values. Wit
types, the computer can have a lot of differen
ing types. To help deal with this potential mis
automatically:

- C++ converts values when you assign a
  of another arithmetic type.

float can have just six significant figures, th
while some conversions are safe, some may p
possible conversion problems.

Table 3.3 **Potential Numeric Conversion Pr**

Conversion Type	**P**
Bigger floating-point type to smaller floating-point type, such as `double` to `float`	L r c
Floating-point type to integer type	L c i
Bigger integer type to smaller integer type, such as `long` to `short`	C t

A zero value assigned to a `bool` variable is
converted to `true`.

Assigning floating-point values to integer
ing floating-point to integer results in trunca
part). Second, a `float` value might be too big

Here is the output from the program in L

```
tree = 3.000000
guess = 3
debt = 1634811904
```

In this case, `tree` is assigned the floating-p variable `guess` causes the value to be truncate fractional part) and not rounding (finding the floating-point types to integer types. Finally, n hold the value 7.2E12. This creates a situation On this system, `debt` ends up with the value novel way to reduce massive indebtedness!

Some compilers issue warnings of possible integer variables to floating-point values. Also compiler to compiler. For example, running t second system produced a value of 21474836

Initialization Conversions When { } Are U

C++11 calls an initialization that uses braces can be used more generally to provide lists of more restrictive in type conversions than the initialization doesn't permit *narrowing*, which able to represent the assigned value. For exam

sion. C++ makes two kinds of automatic co
automatically converted whenever they occu
they are combined with other types in an ex

First, let's examine the automatic conversi
converts `bool`, `char`, `unsigned char`, `signed`
lar, `true` is promoted to 1 and `false` to 0. Th
promotions. For example, consider the followi

```
short chickens = 20;           // line 1
short ducks = 35;              // line 2
short fowl = chickens + ducks; // line 3
```

To execute the statement on line 3, a C++
`ducks` and converts both to `int`. Then the pr
because the answer is assigned to a type `shor`
about, but it does make sense. The `int` type i
most natural type, which means the compute
that type.

There are more integral promotions: The
`short` is smaller than `int`. If the two types ar
verted to `unsigned int`. This rule ensures th
`unsigned short`. Similarly, `wchar_t` is prom
wide enough to accommodate its range: `int`

unsigned operand.

7. Otherwise, if the signed type can repres
unsigned operand is converted to the ty

8. Otherwise, both operands are convertec

ANSI C follows the same rules as ISO 200
the preceding rules, and classic K&R C has ye
C always promotes `float` to `double`, even if b

This list introduces the concept of ranking
expect, the basic ranking for signed integer ty
`int`, `short`, and `signed char`. Unsigned type
signed type. The three types `char`, `signed ch`
rank. The `bool` type has the lowest rank. The
same types as their underlying types.

Conversions in Passing Arguments

Normally, C++ function prototyping control
ments, as you'll learn in Chapter 7, "Function
it is possible, although usually unwise, to waiv
In that case, C++ applies the integral promot

The first form is straight C. The second fo
form is to make a type cast look like a functi
types look like the type conversions you can

C++ also introduces four type cast operat
be used. Chapter 15, "Friends, Exceptions, an
`static_cast<>` operator, can be used for con
another. For example, using it to convert `tho`

```
static_cast<long> (thorn)     // returns
```

More generally, you can do the following:

```
static_cast<typeName> (value)   // conver
```

As Chapter 15 discusses further, Stroustru
dangerously unlimited in its possibilities. The
than the traditional type cast.

Listing 3.14 briefly illustrates both the bas
Imagine that the first section of this listing is
gram that does floating-point calculations tha
and animals. The results you get depend on w
first adds the floating-point values and then c
But the calculations for `bats` and `coots` first
values to `int` and then sum the values. The fi
use a type cast to display the ASCII code for

```
    cout << static_cast<int>(ch) << endl;
    return 0;
}
```

Here is the result of the program in Listing

```
auks = 31, bats = 30, coots = 30
The code for Z is 90
Yes, the code is 90
```

First, adding 19.99 to 11.99 yields 31.98. W
able `auks`, it's truncated to 31. But using type
and 11 before addition, making 30 the result f
statements use type casts to convert a type `cha`
These conversions cause `cout` to print the val

This program illustrates two reasons to use
that are stored as type `double` but are used to
might be fitting a position to a grid or model
floating-point numbers. You might want the c
casting enables you to do so directly. Notice t
these values, when you convert to `int` and ad
convert to `int`.

```
auto z = 0;     // oops, z is int because
```

Using `0` instead of `0.0` doesn't cause prob
automatic type conversion.

Automatic type deduction becomes much
cated types, such as those in the STL (Standa
code might have this:

```
std::vector<double> scores;
std::vector<double>::iterator pv = scores
```

C++11 allows you to write this instead:

```
std::vector<double> scores;
auto pv = scores.begin();
```

We'll mention this new meaning of `auto`
to the topics at hand.

Summary

C++'s basic types fall into two groups. One g
integers. The second group consists of values
integer types differ from each other in the an
whether they are signed or unsigned. From s

which operation takes place first.

C++ converts values from one type to ano mix types in arithmetic, and use type casts to sions are "safe," meaning you can make them ple, you can convert an `int` value to a `long` va conversions of floating-point types to integer

At first, you might find the large number o larly when you take into account the various eventually find occasions when one of the typ you'll thank C++ for having it.

Chapter Review

1. Why does C++ have more than one in

2. Declare variables matching the followin

   a. A `short` integer with the value 8

   b. An `unsigned int` integer with th

   c. An integer with the value 3,000,0

e. 15 % 4

9. Suppose `x1` and `x2` are two type `doubl`
and assign to an integer variable. Const
you want to add them as type `double`

10. What is the variable type for each of th

    a. `auto cars = 15;`

    b. `auto iou = 150.37f;`

    c. `auto level = 'B';`

    d. `auto crat = U'/U00002155';`

    e. `auto fract = 8.25f/2.5;`

Programming Exercises

1. Write a short program that asks for yo
your height to feet and inches. Have th
indicate where to type the response. Al
sent the conversion factor.

lent time in days, hours, minutes, and se
the number of hours in the day, the nu
of seconds in a minute. The output shou

```
Enter the number of seconds: 3160000
31600000 seconds = 365 days, 17 hour
```

5. Write a program that requests the user
the current population of the U.S. (or o
the information in variables of type lo
cent that the U.S. (or other nation's) po
output should look something like this:

```
Enter the world's population: 689875
Enter the population of the US: 3107
The population of the US is 4.50492%
```

You can use the Internet to get more r

6. Write a program that asks how many m
of gasoline you have used and then rep
Or, if you prefer, the program can requ
and then report the result European sty

- Creating and using structures
- Creating and using unions
- Creating and using enumerations
- Creating and using pointers
- Managing dynamic memory with `new`
- Creating dynamic arrays
- Creating dynamic structures
- Automatic, static, and dynamic storage
- The `vector` and `array` classes (an intro

Say you've developed a computer game ca
wits with a cryptic and abusive computer int
keeps track of your monthly game sales for a
your accumulation of hacker-hero trading ca
thing more than C++'s simple basic types to
offers something more—compound types. Th
and floating-point types. The most far-reachi
of OOP toward which we are progressing. B
compound types taken from C. The array, for
type. A particular kind of array can hold a str
can hold several values of differing types. The
tell a computer where data is placed. You'll e

short value:

```
short months[12];     // creates array of
```

Each element, in essence, is a variable that

This is the general form for declaring an a

```
typeName arrayName[arraySize];
```

The expression `arraySize`, which is the n stant, such as 10 or a `const` value, or a constar for which all values are known at the time co `arraySize` cannot be a variable whose value ever, later in this chapter you'll learn how to u restriction.

The Array as Compound Type

An array is called a *compound type* because i term *derived type,* but because C++ uses the come up with a new term.) You can't simply d to be an array of some particular type. There many specific array types, such as array of ch declaration:

```
float loans[20];
```

The type for `loans` is not "array"; rather, it is `loans` array is built from the `float` type.

The Importance of Valid Subscript Valu

The compiler does not check to see if you us
won't complain if you assign a value to the n
assignment could cause problems when the
possibly causing the program to abort. So it
gram uses only valid subscript values.

The yam analysis program in Listing 4.1 d
including declaring an array, assigning values

Listing 4.1 **arrayone.cpp**

```
// arrayone.cpp -- small arrays of intege
#include <iostream>
int main()
{
    using namespace std;
    int yams[3];    // creates array with
    yams[0] = 7;    // assign value to fi
```

Here is the output from the program in Li

```
Total yams = 21
The package with 8 yams costs 30 cents per
The total yam expense is 410 cents.

Size of yams array = 12 bytes.
Size of one element = 4 bytes.
```

Program Notes

First, the program in Listing 4.1 creates a thre
has three elements, the elements are numbere
index values of `0` through `2` to assign values to
vidual `yam` element is an `int` with all the righ
`arrayone.cpp` can, and does, assign values to
and display elements.

The program uses the long way to assign v
initialize array elements within the declaration
assign values to the `yamcosts` array:

```
int yamcosts[3] = {20, 30, 5};
```

```
int cards[4] = {3, 6, 8, 10};       // ok
int hand[4];                        // ok
hand[4] = {5, 6, 7, 9};             // no
hand = cards;                       // no
```

However, you can use subscripts and assign

When initializing an array, you can provid
example, the following statement initializes o

```
float hotelTips[5] = {5.0, 2.5};
```

If you partially initialize an array, the com
Thus, it's easy to initialize all the elements of
first element to zero and then let the compile

```
long totals[500] = {0};
```

Note that if you initialize to `{1}` instead o
rest still get set to `0`.

If you leave the square brackets (`[]`) empt
piler counts the elements for you. Suppose, fo

```
short things[] = {1, 5, 3, 8};
```

The compiler makes `things` an array of fo

Third, as discussed in Chapter 3, list-initiali

```
long plifs[] = {25, 92, 3.0};              /
char slifs[4] {'h', 'i', 1122011, '\0'}; /
char tlifs[4] {'h', 'i', 112, '\0'};     /
```

The first initialization fails because convert
type is narrowing, even if the floating-point va
The second initialization fails because 112201
we have an 8-bit char. The third succeeds bec
is in the range of a char.

The C++ Standard Template Library (STL
vector template class, and C++11 adds an ar
more sophisticated and flexible than the built-
discuss them briefly later, and Chapter 16, "Th
Library," discusses them more fully.

Strings

A *string* is a series of characters stored in conse
of dealing with strings. The first, taken from C
one this chapter examines. Later, this chapter
string class library.

The cat array example makes initializing
single quotes and then having to remember t
ter way to initialize a character array to a stri
stant or *string literal*, as in the following:

```
char bird[11] = "Mr. Cheeps";     // the
char fish[] = "Bubbles";          // let
```

Quoted strings always include the termin
have to spell it out (see Figure 4.2). Also the
string from keyboard input into a `char` array
character for you. (If, when you run the prog
have to use the keyword `static` to initialize
arrays, too.)

Of course, you should make sure the array
the string, including the null character. Initial
is one case where it may be safer to let the co
you. There is no harm, other than wasted spa
That's because functions that work with strin
character, not by the size of the array. C++ in

> **Caution**
>
> When determining the minimum array size ne
> the terminating null character in your count.

address to shirt_size:

```
char shirt_size = "S";         // illegal
```

Because an address is a separate type in C+
nonsense. (We'll return to this point later in th

Concatenating String Literals

Sometimes a string may be too long to conve
you to concatenate string literals—that is, to c
Indeed, any two string constants separated onl
are automatically joined into one. Thus, all the
to each other:

```
cout << "I'd give my right arm to be" " a
cout << "I'd give my right arm to be a gre
cout << "I'd give my right ar"
"m to be a great violinist.\n";
```

Note that the join doesn't add any spaces t
the second string immediately follows the last
string. The `\0` character from the first string is
second string.

```
    cout << "Howdy! I'm " << name2;
    cout << "! What's your name?\n";
    cin >> name1;
    cout << "Well, " << name1 << ", your
    cout << strlen(name1) << " letters an
    cout << "in an array of " << sizeof(n
    cout << "Your initial is " << name1[0
    name2[3] = '\0';                  // se
    cout << "Here are the first 3 charact
    cout << name2 << endl;
    return 0;
}
```

Here is a sample run of the program in Li

```
Howdy! I'm C++owboy! What's your name?
Basicman
Well, Basicman, your name has 8 letters a
in an array of 15 bytes.
Your initial is B.
Here are the first 3 characters of my nam
```

```
name2[3] = '\0';
```

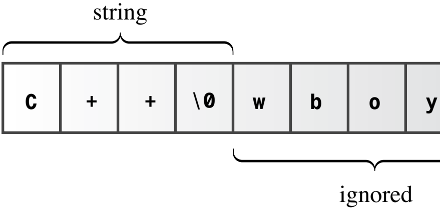


Figure 4.3 Shorte

Note that the program in Listing 4.2 uses
the size of an array appears in several stateme
to represent the size of an array simplifies rev
size; you just have to change the value once,

Adventures in String Input

The `strings.cpp` program has a blemish tha
nique of carefully selected sample input. Listi
string input can be tricky.

```
Enter your name:
Alistair Dreeb
Enter your favorite dessert:
I have some delicious Dreeb for you, Alis
```

We didn't even get a chance to respond to
and then immediately moved on to display th

The problem lies with how `cin` determin
You can't enter the null character from the k
locating the end of a string. The `cin` techniq
newlines—to delineate a string. This means c
for a character array. After it reads this word,
character when it places the string into the a

The practical result in this example is that
and puts it into the `name` array. This leaves po
When `cin` searches the input queue for the r
finds `Dreeb` still there. Then `cin` gobbles up D
Figure 4.4).

Another problem, which didn't surface in
turn out to be longer than the destination ar
protection against placing a 30-character strin

Sao Paulo. You would want the program to r
and Sao. To be able to enter whole phrases ins
different approach to string input. Specifically
a word-oriented method. You are in luck, for
ple, has some line-oriented class member func
entire input line—that is, up until a newline c
the newline character, whereas get() leaves it
beginning with getline().

Line-Oriented Input with getline()

The getline() function reads a whole line, u
the Enter key to mark the end of input. You i
as a function call. The function takes two argu
the target (that is, the array destined to hold th
is a limit on the number of characters to be re
no more than 19 characters, leaving room to a
end. The getline() member function stops r
limit or when it reads a newline character, wh

For example, suppose you want to use get
name array. You would use this call:

```
cin.getline(name,20);
```

```
    return 0;
}
```

Here is some sample output for Listing 4.

```
Enter your name:
Dirk Hammernose
Enter your favorite dessert:
Radish Torte
I have some delicious Radish Torte for yo
```

The program now reads complete names getline() function conveniently gets a line character marking the end of the line, but it replaces it with a null character when storing

Line-Oriented Input with `get()`

Let's try another approach. The `istream` class which comes in several variations. One varia same arguments, interprets them the same wa than read and discard the newline character, queue. Suppose you use two calls to `get()` in

```
cin.get(name, ArSize);
cin.get(dessert, Arsize);   // a problem
```

Because the first call leaves the newline ch
character is the first character the second call
the end of line without having found anythin
past that newline character.

Fortunately, there is help in the form of a v
no arguments) reads the single next character,
dispose of the newline character and prepare
sequence works:

```
cin.get(name, ArSize);      // read first
cin.get();                  // read newlin
cin.get(dessert, Arsize);   // read secon
```

Another way to use `get()` is to *concatenate*,
follows:

```
cin.get(name, ArSize).get(); // concatenat
```

What makes this possible is that `cin.get(`
which is then used as the object that invokes

```
    cout << " for you, " << name << ".\n"
    return 0;
}
```

Here is a sample run of the program in Li

```
Enter your name:
Mai Parfait
Enter your favorite dessert:
Chocolate Mousse
I have some delicious Chocolate Mousse fo
```

One thing to note is how C++ allows mu they have different argument lists. If you use, notices you're using the form that puts a strin member function. If, instead, you use `cin.ge` that reads one character. Chapter 8, "Adventu which is called *function overloading*.

Why use `get()` instead of `getline()` at a `getline()`. Second, `get()` lets you be a bit m `get()` to read a line into an array. How can y stopped because the array was filled? Look at character, then the whole line was read. If it i

Mixing numeric input with line-oriented stri
simple program in Listing 4.6.

Listing 4.6 **numstr.cpp**

```
// numstr.cpp -- following number input wi
#include <iostream>
int main()
{
    using namespace std;
    cout << "What year was your house buil
    int year;
    cin >> year;
    cout << "What is its street address?\n
    char address[80];
    cin.getline(address, 80);
    cout << "Year built: " << year << endl
    cout << "Address: " << address << endl
    cout << "Done!\n";
    return 0;
}
```

```
What year was your house built?
1966
What is its street address?
43821 Unsigned Short Street
Year built: 1966
Address: 43821 Unsigned Short Street
Done!
```

C++ programs frequently use pointers ins
that aspect of strings after talking a bit about
more recent way to handle strings: the C++

Introducing the `string`

The ISO/ANSI C++98 Standard expanded
now, instead of using a character array to hol
(or object, to use C++ terminology). As you
the array and also provides a truer representa

To use the `string` class, a program has to
class is part of the `std` namespace, so you hav
or else refer to the class as `std::string`. The
string and lets you treat a string much like ar
some of the similarities and differences betw

```
    cout << "The third letter in " << str2
         << str2[2] << endl;    // use arr

    return 0;
}
```

Here is a sample run of the program in Lis

```
Enter a kind of feline: ocelot
Enter another kind of feline: tiger
Here are some felines:
ocelot jaguar tiger panther
The third letter in jaguar is g
The third letter in panther is n
```

You should learn from this example that, i
the same manner as a character array:

- You can initialize a `string` object to a
- You can use `cin` to store keyboard inpu
- You can use `cout` to display a `string` o
- You can use array notation to access in

```
string fourth_date {"Hank's Fine Eats"};
```

Assignment, Concatenation, and

The `string` class makes some operations sim
you can't simply assign one array to another.
another:

```
char charr1[20];            // create an
char charr2[20] = "jaguar"; // create an
string str1;                // create an
string str2 = "panther";    // create an
charr1 = charr2;            // INVALID, n
str1 = str2;                // VALID, obj
```

The `string` class simplifies combining str
`string` objects together and the `+=` operator
`string` object. Continuing with the precedin

```
string str3;
str3 = str1 + str2;         // assign str
str1 += str2;               // add str2 t
```

Listing 4.8 illustrates these usages. Note th
well as `string` objects to a `string` object.

```
    cout <<"s1 += s2 yields s1 = " << s1 <
    s2 += " for a day";
    cout <<"s2 += \" for a day\" yields s2

    return 0;
}
```

Recall that the escape sequence \" represe
literal character rather than as marking the lin
program in Listing 4.8:

```
You can assign one string object to anothe
s1: penguin, s2: penguin
You can assign a C-style string to a strin
s2 = "buzzard"
s2: buzzard
You can concatenate strings: s3 = s1 + s2
s3: penguinbuzzard
You can append strings.
s1 += s2 yields s1 = penguinbuzzard
s2 += " for a day" yields s2 = buzzard for
```

```
    char charr2[20] = "jaguar";
    string str1;
    string str2 = "panther";

    // assignment for string objects and
    str1 = str2;                // copy s
    strcpy(charr1, charr2);     // copy c

    // appending for string objects and c
    str1 += " paste";           // add pa
    strcat(charr1, " juice");   // add ju

    // finding the length of a string obj
    int len1 = str1.size();     // obtain
    int len2 = strlen(charr1);  // obtain

    cout << "The string " << str1 << " co
         << len1 << " characters.\n";
    cout << "The string " << charr1 << "
         << len2 << " characters.\n";

    return 0;
}
```

to strcat() and strcpy(), called strncat()
taking a third argument to indicate the maxin
using them adds another layer of complexity

Notice the different syntax used to obtain

```
int len1 = str1.size();     // obtain leng
int len2 = strlen(charr1);  // obtain leng
```

The strlen() function is a regular functio
and that returns the number of characters in t
does the same thing, but the syntax for it is di
argument, str1 precedes the function name a
with the put() method in Chapter 3, this syn
size() is a class method. A method is a funct
belonging to the same class as the method. In
and size() is a string method. In short, the
identify which string to use, and the C++ st
the dot operator to indicate which string to u

More on string Class I/O

As you've seen, you can use cin with the >> o
with the << operator to display a string obje

```
    getline(cin, str);          // cin no
    cout << "You entered: " << str << end
    cout << "Length of string in charr af
         << strlen(charr) << endl;
    cout << "Length of string in str afte
         << str.size() << endl;

    return 0;
}
```

Here's a sample run of the program in Lis

```
Length of string in charr before input: 2
Length of string in str before input: 0
Enter a line of text:
peanut butter
You entered: peanut butter
Enter another line of text:
blueberry jam
You entered: blueberry jam
Length of string in charr after input: 13
Length of string in str after input: 13
```

So it takes cin as an argument that tells it wh
argument for the size of the string because th
the string.

So why is one getline() an istream clas
istream class was part of C++ long before th
design recognizes basic C++ types such as do
string type. Therefore, there are istream cla
the other basic types, but there are no istrea
objects.

Because there are no istream class metho
wonder why code like this works:

```
cin >> str;  // read a word into the str s
```

It turns out that code like the following do
function of the istream class:

```
cin >> x;  // read a value into a basic C-
```

But the string class equivalent uses a frie
the string class. You'll have to wait until Cha
how this technique works. In the meantime, y
objects and not worry about the inner workin

strings:

```
cout << R"(Jim "King" Tutt uses "\n" instead of en
```

This would display the following:

```
Jim "King" Tutt uses \n instead of endl.
```

The standard string literal equivalent would be this:

```
cout << "Jim \"King\" Tutt uses \" \\n\" instead o
```

Here we had to use `\\` to display `\` because a single
ter of an escape sequence.

If you press the Enter or Return key while typing a
cursor to the next line onscreen, it also places a carriag

What if you want to display the combination `)"` in
Won't the compiler interpret the first occurrence of `)"`
will. But the raw string syntax allows you to place addi
opening `"` and `(`. This implies that the same additional
the final `)` and `"`. So a raw string beginning with `R"+*`
the statement

```
cout << R"+*("(Who wouldn't?)", she whispered.)+*"
```

would display the following:

```
"(Who wouldn't?)", she whispered.
```

representation by storing all the related basket
able. If you want to keep track of a whole tean
structure type is also a stepping stone to that b
little about structures now takes you that muc

A structure is a user-definable type, with a
type's data properties. After you define the typ
Thus, creating a structure is a two-part proces
that describes and labels the different types of
you can create structure variables, or, more ge
description's plan.

For example, suppose that Bloataire, Inc., w
of its product line of designer inflatables. In pa
the item, its volume in cubic feet, and its sellin
meets those needs:

```
struct inflatable   // structure declarati
{
    char name[20];
    float volume;
    double price;
};
```

The keyword `struct` indicates that the co
identifier `inflatable` is the name, or *tag*, for

After you have defined the structure, you

```
inflatable hat;             // hat is a
inflatable woopie_cushion;  // type inf
inflatable mainframe;       // type inf
```

If you're familiar with C structures, you'll
allows you to drop the keyword struct whe

```
struct inflatable goose;    // keyword
inflatable vincent;         // keyword
```

In C++, the structure tag is used just like
emphasizes that a structure declaration define
struct from the list of curse-inducing errors

Given that hat is type inflatable, you u
individual members. For example, hat.volum
ture, and hat.price refers to the price mem
member of the vincent variable. In short, th
bers of a structure much as indices enable yo
price member is declared as type double, ha
equivalent to type double variables and can b
double variable can be used. In short, hat is

```
    using namespace std;
    inflatable guest =
    {
        "Glorious Gloria",  // name value
        1.88,               // volume valu
        29.99               // price value
    };  // guest is a structure variable o
// It's initialized to the indicated value
    inflatable pal =
    {
        "Audacious Arthur",
        3.12,
        32.99
    };  // pal is a second variable of typ
// NOTE: some implementations require usin
// static inflatable guest =

    cout << "Expand your guest list with "
    cout << " and " << pal.name << "!\n";
// pal.name is the name member of the pal
    cout << "You can have both for $";
    cout << guest.price + pal.price << "!\
    return 0;
}
```

local declaration—can be used only in this function

type `parts` variable
type `perks` variable

type `parts` variable

can't declare a type `perks` variable here

Figure 4.7 Local and ext

Variables, too, can be defined internally or
among functions. (Chapter 9, "Memory Mod
topic.) C++ practices discourage the use of e

structure, not an array.

C++11 Structure Initialization

As with arrays, C++11 extends the features o

```
inflatable duck {"Daphne", 0.12, 9.98};  /
```

Next, empty braces result in the individual following declaration results in `mayor.volume` bytes in `mayor.name` being set to 0:

```
inflatable mayor {};
```

Finally, narrowing is not allowed.

Can a Structure Use a `string` Cl

Can you use a `string` class object instead of a is, can you declare a structure like this:

```
#include <string>
struct inflatable   // structure definitio
{
    std::string name;
    float volume;
    double price;
};
```

```
struct inflatable
{
    char name[20];
    float volume;
    double price;
};
int main()
{
    using namespace std;
    inflatable bouquet =
    {
        "sunflowers",
        0.20,
        12.49
    };
    inflatable choice;
    cout << "bouquet: " << bouquet.name <
    cout << bouquet.price << endl;

    choice = bouquet;  // assign one stru
    cout << "choice: " << choice.name <<
    cout << choice.price << endl;
    return 0;
}
```

```
    "Packard"     // value for mr_glit
};
```

However, keeping the structure definition ally makes a program easier to read and follow

Another thing you can do with structures do this by omitting a tag name while simultan variable:

```
struct       // no tag
{
    int x;   // 2 members
    int y;
} position;  // a structure variable
```

This creates one structure variable called p the membership operator, as in `position.x`, b You can't subsequently create other variables limited form of structure.

Aside from the fact that a C++ program c structures have all the features discussed so far changes. But C++ structures go further. Unli can have member functions in addition to me features most typically are used with classes ra when we cover classes, beginning with Chapt

```
inflatable guests[2] =          // in
    {
        {"Bambi", 0.5, 21.99},      // fi
        {"Godzilla", 2000, 565.99}  // ne
    };
```

As usual, you can format this the way you
on the same line, or each separate structure n

Listing 4.13 shows a short example that u
`guests` is an array of `inflatable`, `guest[0]`
the dot operator to access a member of the i

Listing 4.13 **arrstruc.cpp**

```
// arrstruc.cpp -- an array of structures
#include <iostream>
struct inflatable
{
    char name[20];
    float volume;
    double price;
};
int main()
{
```

merations are discussed later in this chapter), a
the actual number of bits to be used. You can
member is termed a *bit field*. Here's an exampl

```
struct torgle_register
{
    unsigned int SN : 4;   // 4 bits for S
    unsigned int : 4;      // 4 bits unuse
    bool goodIn : 1;       // valid input
    bool goodTorgle : 1;   // successful t
};
```

You can initialize the fields in the usual ma
tion to access bit fields:

```
torgle_register tr = { 14, true, false };
...
if (tr.goodIn)   // if statement covered i
...
```

Bit fields are typically used in low-level pro
and the bitwise operators listed in Appendix E
approach.

another time. The member name identifies t
Because a union holds only one value at a ti
largest member. Hence, the size of the union

One use for a union is to save space when
but never simultaneously. For example, suppo
ets, some of which have an integer ID, and so
you could use the following:

```
struct widget
{
char brand[20];
int type;
union id               // format depends o
{
    long id_num;       // type 1 widgets
    char id_char[20];  // other widgets
} id_val;
};
...
widget prize;
...
if (prize.type == 1)               // if-
    cin >> prize.id_val.id_num;    // use
else
    cin >> prize.id_val.id_char;
```

inated. It is up to the programmer to keep tra

Unions often (but not exclusively) are used that necessary in these days of gigabytes of R C++ programs are written for such systems. as the processors used to control a toaster ove applications space may be at a premium. Also operating systems or hardware data structures

Enumerations

The C++ `enum` facility provides an alternative also lets you define new types but in a fairly r resembles structure syntax. For example, consi

```
enum spectrum {red, orange, yellow, green,
```

This statement does two things:

- It makes `spectrum` the name of a new much as a `struct` variable is called a st
- It establishes `red`, `orange`, `yellow`, and values 0–7. These constants are called *er*

```
...
```

However, some implementations do not h
ble to violate the type limits. For example, if
++band, if valid, increments band to 8, which
Again, for maximum portability, you should a

Enumerators are of integer type and can b
not converted automatically to the enumerat

```
int color = blue;        // valid, spectr
band = 3;                // invalid, int
color = 3 + red;         // valid, red co
...
```

Note that in this example, even though 3
assigning 3 to band is a type error. But assign
both type spectrum. Again, some implement
expression 3 + red, addition isn't defined fo
type int, and the result is type int. Because
in this situation, you can use enumerations in
with ordinary integers, even though arithmet

The earlier example

```
band = orange + red;     // not valid, bu
```

variables of the enumeration type, you can on
example:

```
enum {red, orange, yellow, green, blue, vi
```

Setting Enumerator Values

You can set enumerator values explicitly by u

```
enum bits{one = 1, two = 2, four = 4, eigh
```

The assigned values must be integers. You a
explicitly:

```
enum bigstep{first, second = 100, third};
```

In this case, `first` is `0` by default. Subsequ
one than their predecessors. So, `third` would

Finally, you can create more than one enum

```
enum {zero, null = 0, one, numero_uno = 1}
```

Here, both `zero` and `null` are `0`, and both
sions of C++, you could assign only `int` valu
merators, but that restriction has been remove
`long long` values.

in a minus sign. (For example, if the smallest
[times a minus sign] is -8, and the lower limi

The idea is that the compiler can choose
tion. It might use 1 byte or less for an enume
enumeration with type `long` values.

C++11 extends enumerations with a form
discusses this form briefly in the section "Cla

Pointers and the Free St

The beginning of Chapter 3 mentions three
program must keep track when it stores data.
thumbing back to that chapter, here are those

- Where the information is stored
- What value is kept there
- What kind of information is stored

You've used one strategy for accomplishin
declaration statement provides the type and a
the program to allocate memory for the valu

```
    return 0;
}
```

Here is the output from the program in Li

```
donuts value = 6 and donuts address = 0x00
cups value = 4.5 and cups address = 0x0065
```

The particular implementation of `cout` sho
displaying address values because that is the us
address. (Some implementations use base 10 n
`donuts` at a lower memory location than `cup`
is 0x0065fd44 - 0x0065fd40, or 4. This makes
uses 4 bytes. Different systems, of course, will
some may store `cups` first, then `donuts`, givin
`double`. And some may not even use adjacent

Using ordinary variables, then, treats the va
a derived quantity. Now let's look at the poin
programming philosophy of memory manage
and the C++ Philosophy.")

Making runtime decisions is not unique to O
straightforward than does C.

The new strategy for handling stored data
tion as the named quantity and the value as a
the *pointer*—holds the address of a value. Thu
location. Applying the * operator, called the *i*
yields the value at the location. (Yes, this is th
C++ uses the context to determine whether
Suppose, for example, that `manly` is a pointer.
`*manly` represents the value at that address. T
to an ordinary type `int` variable. Listing 4.15
to declare a pointer.

Listing 4.15 **pointer.cpp**

```
// pointer.cpp -- our first pointer varia
#include <iostream>
int main()
{
    using namespace std;
    int updates = 6;        // declare a
    int * p_updates;        // declare po
```

two sides of the same coin. The updates varia
the & operator to get the address, whereas the
primary and uses the * operator to get the val
points to updates, *p_updates and updates
*p_updates exactly as you would use a type i
shows, you can even assign values to *p_updat
pointed-to value, updates.

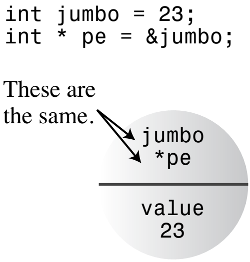
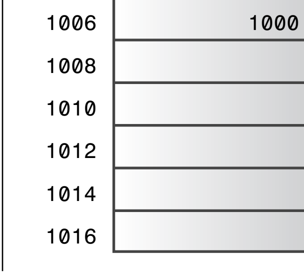


Figure 4.8 Two



`int ducks = 12;`

creates ducks variable, stores
the value 12 in the variable

Figure 4.9 Pointe

Incidentally, the use of spaces around the
grammers have used this form:

```
int *ptr;
```

This accentuates the idea that the combin
programmers, on the other hand, use this for

```
int* ptr;
```

both are pointers, they are pointers of two dif
on other types.

Note that whereas tax_ptr and str point
variables tax_ptr and str themselves are typi
char is the same size as the address of a doubl
for a department store, whereas 1024 could be
size or value of an address doesn't really tell yo
able or building you find at that address. Usua
on the computer system. (Some systems migh
different address sizes for different types.)

You can use a declaration statement to init
the pointed-to value, is initialized. That is, the
the value &higgens:

```
int higgens = 5;
int * pt = &higgens;
```

Listing 4.16 demonstrates how to initialize

Listing 4.16 **init_ptr.cpp**

```
// init_ptr.cpp -- initialize a pointer
#include <iostream>
int main()
{
```

when you create a pointer in C++, the comp
but it does not allocate memory to hold the
space for the data involves a separate step. On
invitation to disaster:

```
long * fellow;          // create a point
*fellow = 223323;       // place a value
```

Sure, `fellow` is a pointer. But where does
to `fellow`. So where is the value `223323` plac
tialized, it could have any value. Whatever tha
address at which to store `223323`. If `fellow` h
computer attempts to place the data at addres
in the middle of your program code. Chance
where you want to put the number `223323`.
most insidious and hard-to-trace bugs.

Caution

Pointer Golden Rule: *Always* initialize a point
you apply the dereferencing operator (`*`) to it

Now both sides of the assignment statemen
assignment is valid. Note that just because it is
mean that pt itself is type int. For example, o
int is a 2-byte value and the addresses are 4-b

Pointers have some other interesting prope
vant. Meanwhile, let's look at how pointers ca
memory space.

Allocating Memory with `new`

Now that you have a feel for how pointers wo
important technique of allocating memory as
pointers to the addresses of variables; the varia
compile time, and each pointer merely provid
directly by name anyway. The true worth of p
unnamed memory during runtime to hold valu
access to that memory. In C, you can allocate
malloc(). You can still do so in C++, but C+

Let's try out this new technique by creatin
value and accessing the value with a pointer. T
new for what data type you want memory; ne

The general form for obtaining and assign
can be a structure as well as a fundamental ty

typeName * *pointer_name* = new *typeName*;

You use the data type twice: once to speci
to declare a suitable pointer. Of course, if you
type, you can use it rather than declare a new
two different types.

Listing 4.17 **use_new.cpp**

```
// use_new.cpp -- using the new operator
#include <iostream>
int main()
{
    using namespace std;
    int nights = 1001;
    int * pt = new int;          // alloca
    *pt = 1001;                  // store

    cout << "nights value = ";
    cout << nights << ": location " << &n
    cout << "int ";
    cout << "value = " << *pt << ": locat
```

Program Notes

The program in Listing 4.17 uses `new` to alloc `double` data objects. This occurs while the pro point to these two data objects. Without them With them, you can use `*pt` and `*pd` just as y `*pt` and `*pd` to assign values to the new data c display those values.

The program in Listing 4.17 also demonstr the type a pointer points to. An address in itse object stored, not its type or the number of by values. They are just numbers with no type or pointer-to-`int` is the same as the size of a poi because `use_new.cpp` declares the pointer typ value of 8 bytes, whereas `*pt` is an `int` value c value of `*pd`, `cout` can tell how many bytes to

Another point to note is that typically `new` the ordinary variable definitions that we have pd have their values stored in a memory regio cated by `new` is in a region called the *heap* or *f*

This removes the memory to which ps po
You can reuse ps, for example, to point to an
ance a use of new with a use of delete; other
that is, memory that has been allocated but ca
grows too large, it can bring a program seeki

You should not attempt to free a block of
C++ Standard says the result of such an atten
quences could be anything. Also you cannot
declaring ordinary variables:

```
int * ps = new int;   // ok
delete ps;            // ok
delete ps;            // not ok now
int jugs = 5;         // ok
int * pi = &jugs;     // ok
delete pi;            // not allowed, mem
```

Caution

You should use delete only to free memory
delete to a null pointer.

that the array is built in to the program at con
array during runtime if you need it and skip c
can select an array size after the program is ru
that the array is created while the program is r
array. With static binding, you must specify the
With dynamic binding, the program can decid

For now, we'll look at two basic matters co
`new` operator to create an array and how to us

Creating a Dynamic Array with `new`

It's easy to create a dynamic array in C++; yo
number of elements you want. The syntax req
the number of elements, in brackets. For exam
use this:

```
int * psome = new int [10]; // get a block
```

The `new` operator returns the address of the
that value is assigned to the pointer `psome`.

As always, you should balance the call to `ne`
finishes using that block of memory. However
requires using an alternative form of `delete` v

```
delete [] psome;                  // free
```

Now let's return to the dynamic array. No
first element of the block. It's your responsibi
in the block. That is, because the compiler do
to the first of 10 integers, you have to write y
number of elements.

Actually, the program does keep track of t
can be correctly freed at a later time when yo
information isn't publicly available; you can't
the number of bytes in a dynamically allocate

The general form for allocating and assign

```
type_name * pointer_name = new type_name
```

Invoking the `new` operator secures a block
`num_elements` elements of type *`type_name`*,
ment. As you're about to see, you can use *`poi`*
can use an array name.

Using a Dynamic Array

After you create a dynamic array, how do you
ceptually. The following statement creates a p
of a block of 10 `int` values:

```
int * psome = new int [10]; // get a bloc
```

```
#include <iostream>
int main()
{
    using namespace std;
    double * p3 = new double [3]; // space
    p3[0] = 0.2;                  // treat
    p3[1] = 0.5;
    p3[2] = 0.8;
    cout << "p3[1] is " << p3[1] << ".\n";
    p3 = p3 + 1;                  // incre
    cout << "Now p3[0] is " << p3[0] << " 
    cout << "p3[1] is " << p3[1] << ".\n";
    p3 = p3 - 1;                  // point
    delete [] p3;                 // free 
    return 0;
}
```

Here is the output from the program in Li

```
p3[1] is 0.5.
Now p3[0] is 0.5 and p3[1] is 0.8.
```

double adds 8 to the numeric value on syste
to a pointer-to-short adds two to the pointe
demonstrates this amazing point. It also show
the array name as an address.

Listing 4.19 **addpntrs.cpp**

```
// addpntrs.cpp -- pointer addition
#include <iostream>
int main()
{
    using namespace std;
    double wages[3] = {10000.0, 20000.0,
    short stacks[3] = {3, 2, 1};

// Here are two ways to get the address o
    double * pw = wages;     // name of a
    short * ps = &stacks[0]; // or use ad
// with array element
    cout << "pw = " << pw << ", *pw = " <
    pw = pw + 1;
    cout << "add 1 to the pw pointer:\n";
    cout << "pw = " << pw << ", *pw = " <
```

```
ps = 0x28ccea, *ps = 3
add 1 to the ps pointer:
ps = 0x28ccec, *ps = 2

access two elements with array notation
stacks[0] = 3, stacks[1] = 2
access two elements with pointer notation
*stacks = 3, *(stacks + 1) =  2
24 = size of wages array
4 = size of pw pointer
```

Program Notes

In most contexts, C++ interprets the name of
Thus, the following statement makes `pw` a poi
to `wages`, which is the address of the first elem

```
double * pw = wages;
```

For `wages`, as with any array, we have the fo

```
wages = &wages[0] = address of first eleme
```

Just to show that this is no jive, the program
expression `&stacks[0]` to initialize the `ps` po

adding 1 to *pw* changes its
value by 8 bytes.

Figure 4.10

After this, the program goes through simil
to type short and because short is 2 bytes, a
2 (0x28ccea + 2 = 0x28ccec in hexadecimal)
point to the next element of the array.

Note

Adding one to a pointer variable increases its
which it points.

Now consider the array expression stacks
sion exactly as if you wrote it as *(stacks +
the address of the second element of the arra
end result is precisely what stacks[1] means
the parentheses. Without them, 1 would be ad

The program output demonstrates that *(
Similarly, *(stacks + 2) is the same as stac
notation, C++ makes the following conversio

```
arrayname[i] becomes *(arrayname + i)
```

array? Not quite—the name of the array is int
an array, whereas applying the address opera

```
short tell[10];        // tell an array
cout << tell << endl;  // displays &tel
cout << &tell << endl; // displays addr
```

Numerically, these two addresses are the sar
`tell`, is the address of a 2-byte block of mem
byte block of memory. So the expression `tel`
`&tell + 1` adds `20` to the address value. Ar
`tell` is type pointer-to-`short`, or `short *`, ar
or `short (*)[20]`.

Now you might be wondering about the genes
how you could declare and initialize a pointer

```
short (*pas)[20] = &tell;  // pas point
```

If you omit the parentheses, precedence rules
`pas` an array of 20 pointers-to-`short`, so the
describe the type of a variable, you can use th
remove the variable name. Thus, the type of p
because `pas` is set to `&tell`, `*pas` is equiva
element of the `tell` array.

You should assign a memory address to a poi
able name to get an address of named memor
unnamed memory.

Here are some examples:

```
double * pn;          // pn can point to
double * pa;          // so can pa
char * pc;            // pc can point to
double bubble = 3.2;
pn = &bubble;         // assign address o
pc = new char;        // assign address o
pa = new double[30];  // assign address o
```

Dereferencing Pointers

Dereferencing a pointer means referring to t
encing, or indirect value, operator (*) to a po
pointer to `bubble`, as in the preceding examp
in this case.

Here are some examples:

```
cout << *pn; // print the value of bubble
*pc = 'S';   // place 'S' into the memory
```

Pointer Arithmetic

C++ allows you to add an integer to a pointe
nal address value plus a value equal to the nun
can also subtract an integer from a pointer to
The last operation, which yields an integer, is
into the same array (pointing to one position
the separation between the two elements.

Here are some examples:

```
int tacos[10] = {5,2,8,4,1,2,2,4,6,8};
int * pt = tacos;       // suppose pf and
pt = pt + 1;            // now pt is 3004
int *pe = &tacos[9];    // pe is 3036 if a
pe = pe - 1;            // now pe is 3032,
int diff = pe - pt;     // diff is 7, the
                        // tacos[8] and ta
```

Dynamic Binding and Static Binding for A

You can use an array declaration to create an a
whose size is set during the compilation proce

```
int tacos[10]; // static binding, size fix
```

```
int coats[10];
*(coats + 4) = 12;        // set coats[4
```

Pointers and Strings

The special relationship between arrays and p
the following code:

```
char flower[10] = "rose";
cout << flower << "s are red\n";
```

The name of an array is the address of its f
ment is the address of the `char` element cont
assumes that the address of a `char` is the addr
that address and then continues printing char
(`\0`). In short, if you give `cout` the address of
character to the first null character that follow

The crucial element here is not that `flowe`
the address of a `char`. This implies that you ca
ment to `cout` also because it, too, is the addre
point to the beginning of a string. We'll chec

But what about the final part of the prece
address of the first character of a string, what
consistent with `cout`'s handling of string outp

```
int main()
{
    using namespace std;
    char animal[20] = "bear";   // animal
    const char * bird = "wren"; // bird ho
    char * ps;                  // uniniti

    cout << animal << " and ";  // display
    cout << bird << "\n";       // display
    // cout << ps << "\n";      //may disp

    cout << "Enter a kind of animal: ";
    cin >> animal;              // ok if i
    // cin >> ps; Too horrible a blunder t
    //            point to allocated space

    ps = animal;                // set ps
    cout << ps << "!\n";       // ok, same
    cout << "Before using strcpy():\n";
    cout << animal << " at " << (int *) an
    cout << ps << " at " << (int *) ps <<

    ps = new char[strlen(animal) + 1];  //
    strcpy(ps, animal);         // copy st
```

"bear" string, just as you've initialized arrays
new. It initializes a pointer-to-char to a strin

```
const char * bird = "wren"; // bird holds
```

Remember, "wren" actually represents the
assigns the address of "wren" to the bird poi
in memory to hold all the quoted strings use
each stored string with its address.) This mea
would use the string "wren", as in this examp

```
cout << "A concerned " << bird << " speak
```

String literals are constants, which is why
laration. Using const in this fashion means y
to change it. Chapter 7 takes up the topic of
pointer ps remains uninitialized, so it doesn't
usually a bad idea, and this example is no exc

Next, the program illustrates that you can
bird equivalently with cout. Both, after all, a
the two strings ("bear" and "wren") stored a
makes the error of attempting to display ps, y
garbage displayed, and you might get a progr
a bit like distributing a blank signed check: Yo

lems:You just use a sufficiently large char arra
stants to receive input or uninitialized pointer
these issues and use std::string objects inste

Caution

When you read a string into a program-style s
previously allocated memory. This address ca
pointer that has been initialized using new.

Next, notice what the following code acco

```
ps = animal;                // set ps to p
...
cout << animal << " at " << (int *) animal
cout << ps << " at " << (int *) ps << endl
```

It produces the following output:

```
fox at 0x0065fd30
fox at 0x0065fd30
```

Normally, if you give cout a pointer, it pri
char *, cout displays the pointed-to string. If
have to type cast the pointer to another point

here by using strlen() to find the correct si

Note that by using strcpy() and new, you

```
fox at 0x0065fd30
fox at 0x004301c8
```

Also note that new located the new storag that of the array animal.

Often you encounter the need to place a s when you initialize an array; otherwise, you u strcpy() function; it works like this:

```
char food[20] = "carrots"; // initializat
strcpy(food, "flan");      // otherwise
```

Note that something like the following ca smaller than the string:

```
strcpy(food, "a picnic basket filled with
```

In this case, the function copies the rest of ately following the array, which can overwrit avoid that problem, you should use strncpy(imum number of characters to be copied. Be

many structures as a program needs during a p
tool to use. With it, you can create dynamic st
is allocated during runtime, not at compile tin
structures, you are able to use the techniques y
classes, too.

Using `new` with structures has two parts: cr
bers. To create a structure, you use the structur
unnamed structure of the `inflatable` type an
can use the following:

```
inflatable * ps = new inflatable;
```

This assigns to `ps` the address of a chunk o
ture of the `inflatable` type. Note that the sy
built-in types.

The tricky part is accessing members. Whe
use the dot membership operator with the str
name. All you have is its address. C++ provide
arrow membership operator (`->`). This operato
greater-than symbol, does for pointers to struc
ture names. For example, if `ps` points to a type
the `price` member of the pointed-to structure

Sometimes new C++ users become confuse
to use the arrow operator to specify a structu
identifier is the name of a structure, use the
structure, use the arrow operator.

A second, uglier approach to accessing stru
pointer to a structure, then `*ps` represents the
because `*ps` is a structure, `(*ps).price` is the
operator precedence rules require that you us

Listing 4.21 uses `new` to create an unname
notations for accessing structure members.

Listing 4.21 **newstrct.cpp**

```
// newstrct.cpp -- using new with a struc
#include <iostream>
struct inflatable   // structure definiti
{
    char name[20];
    float volume;
    double price;
};
int main()
{
    using namespace std;
```

An Example of Using `new` and `delete`

Let's look at an example that uses `new` and `del`
keyboard. Listing 4.22 defines a function `getr`
string. This function reads the input into a larg
with an appropriate size to create a chunk of
the function returns the pointer to the block.
ory for programs that read in a large number
it would be easier to use the `string` class, whi
its design.)

Suppose your program has to read 1,000 st
characters long, but most of the strings are mu
to hold the strings, you'd need 1,000 arrays of
much of that block of memory would wind u
ate an array of 1,000 pointers to `char` and the
memory needed for each string. That could sa
ing to use a large array for every string, you fi
could also use `new` to find space to store only
tle too ambitious for right now. Even using an
tious for right now, but Listing 4.22 illustrates
how `delete` works, the program uses it to free

```
    char temp[80];      // temporary stor
    cout << "Enter last name: ";
    cin >> temp;
    char * pn = new char[strlen(temp) + 1
    strcpy(pn, temp);   // copy string in

    return pn;          // temp lost when
}
```

Here is a sample run of the program in Li

```
Enter last name: Fredeldumpkin
Fredeldumpkin at 0x004326b8
Enter last name: Pook
Pook at 0x004301c8
```

Program Notes

Consider the function `getname()` in the prog
input word into the `temp` array. Next, it uses
word. Including the null character, the progra
store the string, so that's the value given to ne
`getname()` uses the standard library function
the new block. The function doesn't check to

the *free store* or *heap*. Data objects allocated in
how long they remain in existence. We'll take
fourth form called *thread storage* that we'll disc

Automatic Storage

Ordinary variables defined inside a function u
variables. These terms mean that the variables c
function containing them is invoked, and they
example, the `temp` array in Listing 4.22 exists
When program control returns to `main()`, the
cally. If `getname()` returned the address of `te`
pointing to a memory location that would soo
use `new` in `getname()`. Actually, automatic valu
A *block* is a section of code enclosed between
functions. But as you'll see in the next chapter
you define a variable inside one of those block
cuting statements inside the block.

Automatic variables typically are stored on
cution enters a block of code, its variables are
and then are freed in reverse order when exec
in, first-out, or *LIFO*, process.) So the stack gro

the life of a function. Using `new` and `delete`
how a program uses memory than does usin
agement becomes more complex. In a stack,
nism results in the part of memory in use alw
between `new` and `delete` can leave holes in t
to allocate new memory requests more diffic

Stacks, Heaps, and Memory Leaks

What happens if you *don't* call `delete` after
with the `new` operator? The variable or const
continues to persist if `delete` is not called,
pointer has been freed due to rules of scope
way to access the construct on the free stor
tains it is gone. You have now created a *mem*
unusable through the life of the program; it's
In extreme (though not uncommon) cases, m
all the memory available to the application, c
addition, these leaks may negatively affect s
ning in the same memory space, causing the

Even the best programmers and software co
it's best to get into the habit of joining your
ning for and entering the deletion of your co
on the free store. C++'s smart pointers (Cha

We can then access members using the me

```
s01.year = 1998;
```

We can create a pointer to such a structure

```
antarctica_years_end * pa = &s02;
```

Provided the pointer has been set to a vali
membership operator to access members:

```
pa->year = 1999;
```

We can create arrays of structures:

```
antarctica_years_end trio[3]; // array of
```

We then can use the membership operator

```
trio[0].year = 2003;  // trio[0] is a stru
```

Here, `trio` is an array, but `trio[0]` is a stru
that structure. Because an array name is a poi
ship operator:

```
(trio+1)->year = 2004; // same as trio[1]
```

We can create an array of pointers:

```
const antarctica_years_end * arp[3] = {&s0
```

operator:

```
std::cout << (*ppa)->year << std::endl;
std::cout << (*(ppb+1))->year << std::end
```

Because `ppa` points to the first member of
`&s01`. So `(*ppa)->year` is the `year` member
points to the next element, `arp[1]`, which is
correct associations. For example, `*ppa->yea`
`ppa->year`, which fails because the `year` me

Is all this really true? Listing 4.23 incorpo
program.

Listing 4.23 **mixtypes.cpp**

```
// mixtypes.cpp -- some type combinations
#include <iostream>

struct antarctica_years_end
{
    int year;
 /* some really interesting data, etc. */
};

int main()
```

```
1999
```

The program compiles and works as prom

Array Alternatives

Earlier this chapter mentioned the `vector` an
the built-in array. Let's take a brief look now a
benefits of using them.

The `vector` Template Class

The `vector` template class is similar to the `st`
can set the size of a `vector` object during run
end or insert new data in the middle. Basically
dynamic array. Actually, the `vector` class does
but it does so automatically.

At this time we won't venture very deeply
Instead, we'll look at a few basic practical mat
to include the `vector` header file. Second, the
space, so you can use a `using` directive, a `usin`

The vector class has more capabilities than t
cost of slightly less efficiency. If all you need
to use the built-in type. However, that has its
safety. C++11 responded to this situation by
part of the std namespace. Like the built-in
the stack (or else static memory allocation) i
ciency of built-in arrays. To this it adds conve
array object, you need to include the array
that for a vector:

```
#include <array>
...
using namespace std;
array<int, 5> ai;   // create array objec
array<double, 4> ad = {1.2, 2.1, 3.43. 4.
```

More general, the following declaration c
ments of *typeName*:

```
array<typeName, n_elem> arr;
```

Unlike the case for vector, *n_elem* can't b

With C++11, you can use list-initializatio
that was not an option with C++98 vector

```
    a2[3] = 1.0/9.0;
// C++11 -- create and initialize array ob
    array<double, 4> a3 = {3.14, 2.72, 1.6
    array<double, 4> a4;
    a4 = a3;     // valid for array object
// use array notation
    cout << "a1[2]: " << a1[2] << " at " <
    cout << "a2[2]: " << a2[2] << " at " <
    cout << "a3[2]: " << a3[2] << " at " <
    cout << "a4[2]: " << a4[2] << " at " <
// misdeed
    a1[-2] = 20.2;
    cout << "a1[-2]: " << a1[-2] <<" at "
    cout << "a3[2]: " << a3[2] << " at " <
    cout << "a4[2]: " << a4[2] << " at " <
    return 0;
}
```

Here's some sample output:

```
a1[2]: 3.6 at 0x28cce8
a2[2]: 0.142857 at 0xca0328
a3[2]: 1.62 at 0x28ccc8
a4[2]: 1.62 at 0x28cca8
```

a4, and other compilers might make yet othe
unsafe behavior of built-in arrays.

Do the vector and array objects protect
them. That is, you still can write unsafe code,

```
a2[-2] = .5;    // still allowed
a3[200] = 1.4;
```

However, you have alternatives. One is us
can use the getline() member function wit
member function with objects of the vector

```
a2.at(1) = 2.3;  // assign 2.3 to a2[1]
```

The difference between using bracket not
if you use at(), an invalid index is caught du
aborts. This added checking does come at the
C++ gives you the option of using either no
ways of using objects that reduce the chances
classes have begin() and end() member fun
accidentally exceeding the bounds. But we'll

string from one location to another. When us
cstring or the string.h header file.

The C++ string class, supported by the s
more user-friendly means to deal with strings
cally resized to accommodate stored strings, a
copy a string.

The new operator lets you request memory
ning. The operator returns the address of the
address to a pointer. The only means to access
data object is a simple variable, you can use th
value. If the data object is an array, you can us
access the elements. If the data object is a stru
operator (->) to access structure members.

Pointers and arrays are closely connected. I
ar[i] is interpreted as *(ar + i), with the ar
first element of the array. Thus, the array name
can use a pointer name with array notation to

The new and delete operators let you exp
cated and when they are returned to the men
those declared within a function, and static va
or with the keyword static, are less flexible.

Question 3 to the variable `even`.

5. Write a statement that displays the valu
`ideas`.

6. Declare an array of `char` and initialize

7. Declare a `string` object and initialize

8. Devise a structure declaration that desc
kind, the weight in whole ounces, and

9. Declare a variable of the type defined i

10. Use `enum` to define a type called `Respo`
`Maybe`. `Yes` should be `1`, `No` should be

11. Suppose `ted` is a `double` variable. Decl
pointer to display `ted`'s value.

12. Suppose `treacle` is an array of 10 `floa`
ment of `treacle` and use the pointer to

13. Write a code fragment that asks the u
ates a dynamic array of that many `in`
`vector` object.

1. Write a C++ program that requests an
lowing example of output:

```
What is your first name? Betty Sue
What is your last name? Yewe
What letter grade do you deserve? B
What is your age? 22
Name: Yewe, Betty Sue
Grade: C
Age: 22
```

Note that the program should be able t
than one word. Also note that the progr
one letter. Assume that the user requests
worry about the gap between a D and a

2. Rewrite Listing 4.4, using the C++ `st`

3. Write a program that asks the user to en
name, and that then constructs, stores, a
user's last name followed by a comma, a
functions from the `cstring` header file.

```
Enter your first name: Flip
Enter your last name: Fleming
Here's the information in a single s
```

7. William Wingate runs a pizza-analysis s
the following information:

   - The name of the pizza company,
   - The diameter of the pizza
   - The weight of the pizza

   Devise a structure that can hold this in
structure variable of that type. The prog
preceding items of information, and th
tion. Use `cin` (or its methods) and `cou`

8. Do Programming Exercise 7 but use `n`
a structure variable. Also have the prog
requests the pizza company name.

9. Do Programming Exercise 6, but inste
structures, use `new` to allocate the array

10. Write a program that requests the user
40-meter, if you prefer) and then displa
object to hold the data. (Use a built-in

- Compound statements (blocks)
- The comma operator
- Relational operators: `>`, `>=`, `==`, `<=`, `<`, ar
- The `while` loop
- The `typedef` facility
- The `do while` loop
- The `get()` character input method
- The end-of-file condition
- Nested loops and two-dimensional arr

Computers do more than store data. They
ify, extrapolate, synthesize, and otherwise ma
and trash data, but we'll try to steer clear of t
manipulative miracles, programs need tools f
ing decisions. Of course, C++ provides such
`while` loops, `do while` loops, `if` statements,
employs, so if you know C, you can zip throu
Statements and Logical Operators." (But don
`cin` handles character input!) These various p
tional expressions and logical expressions to

```
    return 0;
}
```

Here is the output from the program in Li

```
C++ knows loops.
C++ knows loops.
C++ knows loops.
C++ knows loops.
C++ knows loops.
C++ knows when to stop.
```

This loop begins by setting the integer `i` t

```
i = 0
```

This is the *loop initialization* part of the loo
whether `i` is less than 5:

```
i < 5
```

If it is, the program executes the following

```
cout << "C++ knows loops.\n";
```

Then the program uses the *loop update* part

```
i++
```

The initialization, test, and update actions co
in parentheses. Each part is an expression, and
each other. The statement following the cont
it is executed as long as the test expression re

```
for (initialization; test-expression; upd
    body
```

C++ syntax counts a complete `for` statem
incorporate one or more statements in the b
ment requires using a compound statement,

The loop performs initialization just once
a variable to a starting value and then use the

test-expression determines whether th
expression is a relational expression—that is,
compares the value of `i` to 5, checking wheth
the program executes the loop body. Actually
true/false comparisons. You can use any expr
`bool`. Thus, an expression with a value of 0 is
loop terminates. If the expression evaluates to
`true`, and the loop continues. Listing 5.2 dem
the test condition. (In the update section, `i--`
value of `i` by 1 each time it's used.)

```
Done now that i = 0
```

Note that the loop terminates when i reac

How do relational expressions, such as i <

loop with a 0 value? Before the bool type wa
to 1 if true and 0 if false. Thus, the value of th
5 < 5 was 0. Now that C++ has added the bo
evaluate to the bool literals true and false i
to incompatibilities, however, because a C++
0 where integer values are expected, and it co
where bool values are expected.

The for loop is an *entry-condition* loop. Thi
before each loop cycle. The loop never execute
false. For example, suppose you rerun the prog
value. Because the test condition fails the very
never gets executed:

```
Enter the starting countdown value: 0
Done now that i = 0
```

This look-before-you-loop attitude can he

update-expression is evaluated at the en
cuted. Typically, it's used to increase or decrea

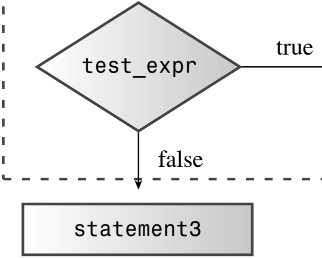


Figure 5.1 The c

A for statement looks something like a fu
by paired parentheses. However, for's status a
from thinking for is a function. It also preve

Tip

Common C++ style is to place a space betw
omit space between a function name and th

```
for (i = 6; i < 10; i++)
      smart_function(i);
```

The expression `cooks = 4` has the value 4. just because C++ permits this behavior doesn same rule that makes this peculiar statement p ment possible:

```
x = y = z = 0;
```

This is a fast way to set several variables to Appendix D, "Operator Precedence") reveals first `0` is assigned to `z`, and then `z = 0` is assign

Finally, as mentioned previously, relational `bool` values `true` or `false`. The short program expression values. The `<<` operator has higher expressions, so the code uses parentheses to en

Listing 5.3 **express.cpp**

```
// express.cpp -- values of expressions
#include <iostream>
int main()
{
    using namespace std;
    int x;

    cout << "The expression x = 100 has th
```

Normally, cout converts bool values to in
cout.setf(ios::boolalpha) function call s
words true and false instead of 1 and 0.

Note

A C++ expression is a value or a combinatio
expression has a value.

To evaluate the expression `x = 100`, C++
very act of evaluating an expression changes
uation has a *side effect*. Thus, evaluating an assi
changing the assignee's value. You might thin
from the standpoint of how C++ is construc
effect. Not all expressions have side effects. F
new value, but it doesn't change the value of
effect because it involves incrementing `x`.

From expression to statement is a short st
lowing is an expression:

```
age = 100
```

Whereas the following is a statement:

```
age = 100;
```

code such as the following invalid:

```
eggs = int toad * 1000;   // invalid, not
cin >> int toad;          // can't combine
```

Similarly, you can't assign a `for` loop to a v
loop is not an expression, so it has no value ar

```
int fx = for (i = 0; i< 4; i++)
     cout >> i;   // not possible
```

Bending the Rules

C++ adds a feature to C loops that requires so
tax. This was the original syntax:

```
for (expression; expression; expression)
     statement
```

In particular, the control section of a `for` st
defined earlier in this chapter, separated by sem
like the following, however:

```
for (int i = 0; i < 5; i++)
```

That is, you can declare a variable in the ini
convenient, but it doesn't fit the original synta
This once outlaw behavior was originally acco

earlier rule and treat the preceding loop as if
it available after the loop terminates.

Back to the `for` Loop

Let's be a bit more ambitious with loops. List
first 16 factorials. Factorials, which are handy
lowing way. Zero factorial, written as 0!, is de
2! is 2 * 1!, or 2. Then, 3! is 3 * 2!, or 6, and s
the product of that integer with the precedin
Victor Borge's best-known monologues featu
exclamation mark is pronounced something l
ever, in this case, "!" is pronounced "factorial.
the values of successive factorials, storing the
display the results. Also the program introduc

Listing 5.4 **formore.cpp**

```
// formore.cpp -- more looping with for
#include <iostream>
const int ArSize = 16;      // example of
int main()
{
    long long factorials[ArSize];
```

```
13! = 6227020800
14! = 87178291200
15! = 1307674368000
```

Factorials get big fast!

> **Note**
>
> This listing uses the `long long` type. If your
> can use `double`. However, the integer format
> numbers grow larger.

Program Notes

The program in Listing 5.4 creates an array to
element 1 is 1!, and so on. Because the first tw
first two elements of the `factorials` array to
array has an index value of 0.) After that, the p
the product of the index with the previous fac
the loop counter as a variable in the body of t

The program in Listing 5.4 demonstrates h
arrays by providing a convenient means to acc
`formore.cpp` uses `const` to create a symbolic
Then it uses `ArSize` wherever the array size c

so we ll practice using them next.

Also this example reminds us that we can
make selected standard names available.

Changing the Step Size

So far the loop examples in this chapter have
one in each cycle.You can change that by ch
Listing 5.5, for example, increases the loop c
than use `i++` as the update expression, it uses
user-selected step size.

Listing 5.5 **bigstep.cpp**

```
// bigstep.cpp -- count as directed
#include <iostream>
int main()
{

    using std::cout;   // a using declara
    using std::cin;
    using std::endl;
    cout << "Enter an integer: ";
    int by;
```

value 100.

Finally, this example illustrates the use of us

Inside Strings with the `for` Loop

The for loop provides a direct way to access e
ple, Listing 5.6 enables you to enter a string an
character, in reverse order. You could use eithe
in this example because both allow you to use
ters in a string; Listing 5.6 uses a string class
yields the number of characters in the string; t
expression to set i to the index of the last cha
character. To count backward, the program use
the array subscript by one in each loop. Also L
relational operator (>=) to test whether the loo
marize all the relational operators soon.

Listing 5.6 **forstr1.cpp**

```
// forstr1.cpp -- using for with a string
#include <iostream>
#include <string>
int main()
{
```

The Increment (++) and Decreme

C++ features several operators that are frequ
examine them now. You've already seen two:
the name C++, and the decrement operator
ingly common loop operations: increasing an
ever, there's more to their story than you've s
two varieties. The *prefix* version comes before
comes after the operand, as in `x++`. The two v
but they differ in terms of when they take pla
in advance or afterward; both methods have t
differ in when the money gets added. Listing
increment operator.

Listing 5.7 **plus_one.cpp**

```
// plus_one.cpp -- the increment operator
#include <iostream>
int main()
{

    using std::cout;
    int a = 20;
    int b = 20;
```

Using the increment and decrement opera
the common task of increasing or decreasing

The increment and decrement operators a
away and increment or decrement the same v
The problem is that the use-then-change and
ous. That is, a statement such as the following
ent systems:

```
x = 2 * x++ * (3 - ++x);    // don't do it
```

C++ does not define correct behavior for

Side Effects and Sequence Points

Let's take a closer look at what C++ does and
tors take effect. First, recall that a *side effect* is a
expression modifies something, such as a valu
point in program execution at which all side
going on to the next step. In C++ the semico
That means all changes made by assignment o
ment operators in a statement must take plac
statement. Some operators that we'll discuss i
the end of any full expression is a sequence p

expression is the entire assignment statement
point, so all that C++ guarantees is that x wi
the program moves to the following statemen
mented after each subexpression is evaluated
evaluated, which is why you should avoid sta

C++11 documentation has dropped the t
doesn't carry over well when discussing mult
are framed in terms of sequencing, with some
before other events. This descriptive approach
is to provide language that can more clearly h

Prefixing Versus Postfixing

Clearly, whether you use the prefix or postfix
for some purpose, such as a function argumen
value of an increment or decrement expressio

```
x++;
```

and

```
++x;
```

different from one another? Or are

```
for (n = lim; n > 0; --n)
```

The Increment/Decrement Operat

You can use increment operators with pointer
adding an increment operator to a pointer inc
the type it points to. The same rule holds for i

```
double arr[5] = {21.1, 32.8, 23.4, 45.2, 3
double *pt = arr;  // pt points to arr[0],
++pt;              // pt points to arr[1],
```

You can also use these operators to change
them in conjunction with the * operator. App
questions of what gets dereferenced and what
mined by the placement and precedence of th
decrement, and dereferencing operators all hav
right to left. The postfix increment and decrem
dence, which is higher than the prefix precede
to right.

The right-to-left association rule for prefix
apply ++ to pt (because the ++ is to the right
value of pt:

```
double x = *++pt;  // increment pointer, t
```

the first member of an array, `++pt` changes p

Combination Assignment Operato

Listing 5.5 uses the following expression to u

```
i = i + by
```

C++ has a combined addition and assignm result more concisely:

```
i += by
```

The `+=` operator adds the values of its two operand on the left. This implies that the left can assign a value, such as a variable, an array identify by dereferencing a pointer:

```
int k = 5;
k += 3;                  // ok, k set to
int *pa = new int[10];   // pa points to
pa[4] = 12;
pa[4] += 6;              // ok, pa[4] set
*(pa + 4) += 7;          // ok, pa[4] set
pa += 2;                 // ok, pa points
34 += 10;                // quite wrong
```

to use paired braces to construct a *compound st*
braces and the statements they enclose and, fo
statement. For example, the program in Listin
statements into a single block. This enables the
input, and do a calculation. The program calcu
enter, and this provides a natural occasion for

Listing 5.8 **block.cpp**

```
// block.cpp -- use a block statement
#include <iostream>
int main()
{
    using namespace std;
    cout << "The Amazing Accounto will sum
    cout << "five numbers for you.\n";
    cout << "Please enter five values:\n";
    double number;
    double sum = 0.0;
    for (int i = 1; i <= 5; i++)
    {                                         //
        cout << "Value " << i << ": ";
        cin >> number;
        sum += number;
    }                                         //
```

```
        cin >> number;
        sum += number;
cout << "Five exquisite choices indeed! "
```

The compiler ignores indentation, so only
Thus, the loop would print the five prompts
pletes, the program moves to the following li

Compound statements have another intere
inside a block, the variable persists only as lo
within the block. When execution leaves the
the variable is known only within the block:

```
#include  <iostream>
int main()
{
    using namespace std;
    int x = 20;
    {                         // block star
        int y = 100;
        cout << x << endl;  // ok
        cout << y << endl;  // ok
    }                         // block ends
    cout << x << endl;      // ok
    cout << y << endl;      // invalid, w
    return 0;
```

As you have seen, a block enables you to snea
C++ syntax allows just one statement. The co
sions, enabling you to sneak two expressions i
one expression. For example, suppose you hav
one each cycle and a second variable decrease
update part of a `for` loop control section wou
just one expression there. The solution is to us
expressions into one:

```
++j, --i   // two expressions count as one
```

The comma is not always a comma operat
tion serves to separate adjacent names in a list

```
int i, j;  // comma is a separator here, n
```

Listing 5.9 uses the comma operator twice
`string` class object. (You could also write the
length of the word would be limited by your
displays the contents of an array in reverse ord
around in the array. The program in Listing 5.
ments into one.

}

Here is a sample run of the program in Li

```
Enter a word: stressed
desserts
Done
```

By the way, the `string` class offers more c leave those for Chapter 16, "The `string` Cla

Program Notes

Look at the `for` control section of the progra operator to squeeze two initializations into o section. Then it uses the comma operator aga expression for the last part of the control sect

Next, look at the body. The program uses single unit. In the body, the program reverses the array with the last element. Then it incre refer to the next-to-the-first element and the the program swaps those elements. Note that when it reaches the center of the array. If it w begin swapping the switched elements back t

Another thing to note is the location for d
code declares `i` and `j` before the loop because
comma operator. That's because declarations a
pose—separating items in a list. You can use a
create and initialize two variables, but it's a bit

```
int j = 0, i = word.size() - 1;
```

In this case the comma is just a list separato
sion declares and initializes both `j` and `i`. How

Incidentally, you can declare `temp` inside th

```
int temp = word[i];
```

This may result in `temp` being allocated and
be a bit slower than declaring `temp` once befo
loop is finished, `temp` is discarded if it's declare

Comma Operator Tidbits

By far the most common use for the comma
into a single `for` loop expression. But C++ de
properties. First, it guarantees that the first exp

Computers are more than relentless number
pare values, and this capability is the foundati
relational operators embody this ability. C++
numbers. Because characters are represented
operators with characters, too. They don't wo
with `string` class objects. Each relational exp
comparison is true and to the `bool` value `fal`
tors are well suited for use in a loop test expr
relational expressions to `1` and false relational
operators.

Table 5.2 **Relational Operators**

Operator
`<<`
`<=`
`==`
`>`
`>=`
`!=`

Assignment, Comparison, and a M

Don't confuse testing the is-equal-to operator
expression asks the musical question "Is music

```
musicians == 4    // comparison
```

The expression has the value `true` or `fals`
`musicians`:

```
musicians = 4     // assignment
```

The whole expression, in this case, has the v
The flexible design of the `for` loop creates
accidentally drop an equals sign (`=`) from the `=`
instead of a relational expression for the test p
code. That's because you can use any valid C+
Remember that nonzero values test as `true`, a
assigns `4` to `musicians` has the value `4` and is t
guage, such as Pascal or BASIC, that uses `=` to
prone to this slip.

Listing 5.10 shows a situation in which you
attempts to examine an array of quiz scores an
not 20. It shows a loop that correctly uses con
assignment in the test condition. The program

```
    return 0;
}
```

Because the program in Listing 5.10 has a
about it to actually running it. Here is some s

```
Doing it right:
quiz 0 is a 20
quiz 1 is a 20
quiz 2 is a 20
quiz 3 is a 20
quiz 4 is a 20
Doing it dangerously wrong:
quiz 0 is a 20
quiz 1 is a 20
quiz 2 is a 20
quiz 3 is a 20
quiz 4 is a 20
quiz 5 is a 20
quiz 6 is a 20
quiz 7 is a 20
quiz 8 is a 20
quiz 9 is a 20
quiz 10 is a 20
```

Caution

Don't use `=` to compare for equality; use `==`.

Like C, C++ grants you more freedom tha
at the cost of requiring greater responsibility o
planning prevents a program from going beyo
However, with C++ classes, you can design a
of nonsense. Chapter 13, "Class Inheritance," p
build the protection into your programs wher
Listing 5.10 should include a test that keeps it
even for the "good" loop. If all the scores were
the array bounds. In short, the loop needs to t
index. Chapter 6 shows how to use logical op
gle condition.

Comparing C-Style Strings

Suppose you want to see if a string in a charac
array name, the following test might not do w

```
word == "mate"
```

Remember that the name of an array is a s
string constant is a synonym for its address. Th

```
char little[6] = "Daffy";       // 5 letters plus
```

By the way, although you can't use relational operat
them to compare characters because characters are actu
following is valid code, at least for the ASCII and Unic
the characters of the alphabet:

```
for (ch = 'a'; ch <= 'z'; ch++)
      cout << ch;
```

The program in Listing 5.11 uses `strcmp()` in the t
program displays a word, changes its first letter, displays
until `strcmp()` determines that `word` is the same as the
ing includes the `cstring` file because it provides a func

Listing 5.11 **compstr1.cpp**

```
// compstr1.cpp -- comparing strings using arrays
#include <iostream>
#include <cstring>     // prototype for strcmp()
int main()
{
    using namespace std;
    char word[5] = "?ate";
```

```
kate
late
After loop ends, word is mate
```

Program Notes

The program in Listing 5.11 has some interes
want the loop to continue as long as `word` is n
tinue as long as `strcmp()` says the two strings
that is this:

```
strcmp(word, "mate") != 0    // strings ar
```

This statement has the value `1` (`true`) if the
if they are equal. But what about `strcmp(wor`
(`true`) if the strings are unequal and the value
the function returns `true` if the strings are dif
can use just the function instead of the whole
same behavior and involves less typing. Also it
traditionally used `strcmp()`.

Testing for Equality or Order

You can use `strcmp()` to test C-style strings
is true if `str1` and `str2` are identical:

```
strcmp(str1,str2) == 0
```

type char really is an integer type, so the ope
stored in the variable. Also note that using an
vidual characters in a string:

```
word[0] = ch;
```

Comparing `string` Class Strings

Life is a bit simpler if you use `string` class st
class design allows you to use relational opera
ble because one can define class functions tha
12, "Classes and Dynamic Memory Allocatio
into class designs, but from a practical standpo
can use the relational operators with `string`
to use a `string` object instead of an array of

Listing 5.12 **compstr2.cpp**

```
// compstr2.cpp -- comparing strings usin
#include <iostream>
#include <string>     // string class
int main()
{
    using namespace std;
    string word = "?ate";
```

relational test expression, or as an aggregate o
to extract individual characters.

As you can see, you can achieve the same r
objects, but programming with `string` object

Finally, unlike most of the `for` loops you h
aren't counting loops. That is, they don't execu
of times. Instead, each of these loops watches
`"mate"`) to signal that it's time to stop. More t
this second kind of test, so let's examine that f

The `while` Loop

The `while` loop is a `for` loop stripped of the
test condition and a body:

```
while (test-condition)
        body
```

First, a program evaluates the parenthesized
sion evaluates to `true`, the program executes t
loop, the body consists of a single statement o
finishes with the body, the program returns to
condition is nonzero, the program executes th
cution continues until the test condition evalu

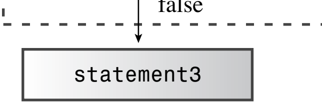


Figure 5.3 The stru

Listing 5.13 puts a `while` loop to work. Th
string and displays the character and its ASCI
null character. This technique of stepping thr
reaching the null character is a standard C++
Because a string contains its own termination
information about how long a string is.

Listing 5.13 **`while.cpp`**

```
// while.cpp -- introducing the while loo
#include <iostream>
const int ArSize = 20;
int main()
{
    using namespace std;
    char name[ArSize];
```

they do add an endearing technoid tone to th

Program Notes

The `while` condition in Listing 5.13 looks lik

```
while (name[i] != '\0')
```

It tests whether a particular character in th eventually succeed, the loop body needs to ch menting `i` at the end of the loop body. Omitt same array element, printing the character and gram. Getting such an infinite loop is one of t Often you can cause it when you forget to up

You can rewrite the `while` line this way:

```
while (name[i])
```

With this change, the program works just a is an ordinary character, its value is the charac when `name[i]` is the null character, its charact more concise (and more commonly used) but Dumb compilers might produce faster code fo produce the same code for both.

```
    update-expression;
}
```

Similarly, the `while` loop

```
while (test-expression)
    body
```

could be rewritten this way:

```
for ( ;test-expression;)
    body
```

This `for` loop requires three expressions (
by two expressions), but they can be empty e
semicolons are mandatory. Incidentally, a mis
as true, so this loop runs forever:

```
for ( ; ; )
    body
```

Because `for` loops and `while` loops are ne
matter of style. There are three differences. O
condition in a `for` loop is interpreted as `true`
ing statement in a `for` loop to declare a varia
with a `while` loop. Finally, there is a slight dif
statement, which is discussed in Chapter 6. Ty

```
i = 0;
while (name[i] != '\0')
      cout << name[i] << endl;
      i++;
cout << "Done\n";
```

The indentation tells you that the program a
of the loop body. The absence of braces, ho
sists solely of the first `cout` statement. Thu
of the array indefinitely. The program never r
side the loop.

The following example shows another potenti

```
i = 0;
while (name[i] != '\0');    // problem
{
      cout << name[i] << endl;
      i++;
}
cout << "Done\n";
```

This time the code gets the braces right, but
a semicolon terminates a statement, so this
words, the body of the loop is a *null statemen*
the material in braces now comes *after* the lo
cycles, doing nothing forever. Beware the stra

unsigned long on others, and perhaps some

But the ctime header file (time.h on less to these problems. First, it defines a symbolic number of system time units per second. So seconds. Or you can multiply seconds by CLO Second, ctime establishes clock_t as an alias "Type Aliases," later in this chapter.) This mea clock_t, and the compiler converts it to lon type for your system.

Listing 5.14 shows how to use clock() and

Listing 5.14 **waiting.cpp**

```
// waiting.cpp -- using clock() in a time
#include <iostream>
#include <ctime> // describes clock() fun
int main()
{
    using namespace std;
    cout << "Enter the delay time, in sec
    float secs;
    cin >> secs;
    clock_t delay = secs * CLOCKS_PER_SEC
```

```
typedef char byte;   // makes byte an a
```

Here's the general form:

```
typedef typeName aliasName;
```

In other words, if you want aliasName to be
aliasName as if it were a variable of that typ
typedef keyword. For example, to make byt
could declare byte_pointer as a pointer-to-

```
typedef char * byte_pointer; // pointer
```

You could try something similar with #define
of variables. For example, consider the follow

```
#define FLOAT_POINTER float *
FLOAT_POINTER pa, pb;
```

Preprocessor substitution converts the decla

```
float * pa, pb;  // pa a pointer to flo
```

The typedef approach doesn't have that pro
aliases makes using typedef a better choic
only choice.

Notice that typedef doesn't create a new ty
If you make word an alias for int, cout trea

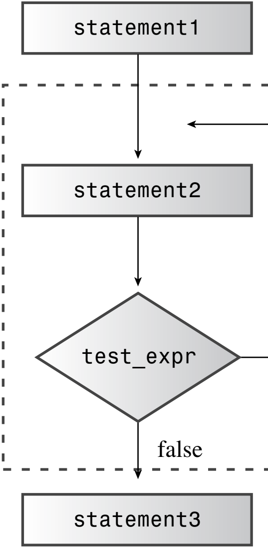


Figure 5.4 The struc

Usually, an entry-condition loop is a bett
the entry-condition loop checks before loop

```
}
```

Here's a sample run of the program in Listi

```
Enter numbers in the range 1-10 to find my
9
4
7
Yes, 7 is my favorite.
```

Strange `for` loops

It's not terribly common, but you may occasio

```
int I = 0;
for(;;)  // sometimes called a "forever
{
    I++;
    // do something ...
    if (30 >= I) break;    // if statem
}
```

Or here is another variation:

```
int I = 0;
for(;;I++)
{
```

The Range-Based `for` Loop (C

The C++11 adds a new form of loop called the *range-b*
common loop task—that of doing something with each
generally, of one of the container classes, such as `vecto`

```
double prices[5] = {4.99, 10.99, 6.87, 7.99, 8.49}
for (double x : prices)
    cout << x << std::endl;
```

Here `x` initially represents the first member of the `pr`
first element, the loop then cycles `x` to represent the re
turn, so this code would print all five members, one pe
every value included in the range of the array.

To modify array values, you need a different syntax f

```
for (double &x : prices)
    x = x * 0.80;        //20% off sale
```

The `&` symbol identifies `x` as a reference variable, a to
"Adventures in Functions." The significance here is tha
subsequent code to modify the array contents, whereas

The range-based `for` loop also can be used with ini

```
for (int x : {3, 5, 2, 8, 6})
    cout << x << " ";
cout << '\n';
```

example, Listing 5.16 stops reading input wh
program counts the number of characters it re
plays the characters that have been read. (Pres
place a character onscreen; programs have to
character. Typically, the operating system hand
system and the test program echo the input.)
total number of characters processed. Listing 5

Listing 5.16 **textin1.cpp**

```
// textin1.cpp -- reading chars with a whi
#include <iostream>
int main()
{
    using namespace std;
    char ch;
    int count = 0;      // use basic input
    cout << "Enter characters; enter # to
    cin >> ch;          // get a character
    while (ch != '#')   // test the charac
    {
        cout << ch;     // echo the charac
        ++count;        // count the chara
        cin >> ch;      // get the next ch
    }
```

plays the character, increments the count, and
vital. Without it, the loop repeatedly processe
last step, the program advances to the next ch

Note that the loop design follows the guid
terminates the loop is if the last character rea
a character before the loop starts. The conditi
the end of the loop.

This all sounds reasonable. So why does th
cin. When reading type char values, just as w
spaces and newline characters. The spaces in t
counted.

To further complicate things, the input to
you type don't get sent to the program until t
type characters after the # when running the
Enter, the whole sequence of characters is ser
cessing the input after it reaches the # charact

cin.get(char) to the Rescue

Usually, programs that read input character-b
including spaces, tabs, and newlines. The istr
cin belongs, includes member functions that
function cin.get(ch) reads the next charact

```
}
```

Here is a sample run of the program in Lis

```
Enter characters; enter # to quit:
Did you use a #2 pencil?
Did you use a
14 characters read
```

Now the program echoes and counts every
buffered, so it is still possible to type more inp

If you are familiar with C, this program ma
`cin.get(ch)` call places a value in the `ch` vari
variable. In C you must pass the address of a v
the value of that variable. But the call to `cin.`
C, code like this won't work. In C++ it can w
argument as a *reference*. The reference type is s
`iostream` header file declares the argument to
function can alter the value of its argument. Yo
while, the C mavens among you can relax; ord
as it does in C. For `cin.get(ch)`, however, it

the same function with a single argument of
because the language supports an OOP featu
loading allows you to create different functio
they have different argument lists. If, for exan
C++, the compiler finds the version of `cin.`
ments. But if you use `cin.get(ch)`, the comp
type `char` argument. And if the code provide
of `cin.get()` that takes no arguments.

Function overloading enables you to use t
form the same basic task in different ways or
awaiting you in Chapter 8. Meanwhile, you c
by using the `get()` examples that come with
the different function versions, we'll include
Thus, `cin.get()` means the version that take
the version that takes one argument.

The End-of-File Condition

As Listing 5.17 shows, using a symbol such as
satisfactory because such a symbol might be
other arbitrarily chosen symbols, such as `@` an
employ a much more powerful technique—

GNU C++ for the PC recognize Ctrl+Z wh
require a subsequent Enter. In short, many PC
Ctrl+Z as a simulated EOF, but the exact deta
on a line, Enter key required or not required)

If your programming environment can test
to Listing 5.17 with redirected files and you c
simulate the EOF. That sounds useful, so let's s

When `cin` detects the EOF, it sets two bits
member function named `eof()` to see whethe
returns the `bool` value `true` if the EOF has be
the `fail()` member function returns `true` if
`1` and `false` otherwise. Note that the `eof()` a
most recent attempt to read; that is, they repor
`cin.eof()` or `cin.fail()` test should always
Listing 5.18 reflects this fact. It uses `fail()` in
appears to work with a broader range of impl

Note

Some systems do not support simulated EOF
imperfectly. If you have been using `cin.get(`
that won't work here because detecting the E

Here is sample output from the program i

```
The green bird sings in the winter.<ENTER
The green bird sings in the winter.
Yes, but the crow flies in the dawn.<ENTE
Yes, but the crow flies in the dawn.
<CTRL>+<Z><ENTER>
73 characters read
```

Because I ran the program on a Windows
simulate the EOF condition. Unix and Linux
in Unix and Unix-like systems, including Lin
of the program; the `fg` command lets executi

By using redirection, you can use the prog
report how many characters it has. This time,
characters from a two-line file on a Unix syst

```
$ textin3 < stuff
I am a Unix file. I am proud
to be a Unix file.
48 characters read
$
```

operator, which toggles `true` to `false` and vi
test to look like this:

```
while (!cin.fail())    // while input has
```

The return value for the `cin.get(char)` m
`istream` class provides a function that can co
`bool` value; this conversion function is called
is expected, such as in the test condition of a
the conversion is `true` if the last attempted re
means you can rewrite the `while` test to look

```
while (cin)    // while input is successfu
```

This is a bit more general than using `!cin`
other possible causes of failure, such as disk fa

Finally, because the return value of `cin.ge`
to this format:

```
while (cin.get(ch))  // while input is suc
{
    ...              // do stuff
}
```

Here, `cin.get(char)` is called once in the
before the loop and once at the end of the lo

```
cout.put(ch);
```

It works much like C's `putchar()`, except
instead of type `int`.

> **Note**
>
> Originally, the `put()` member had the single
> `int` argument, which would then be type cas
> prototype. However, some C++ implementati
> `put(signed char)`, and `put(unsigned c`
> these implementations generates an error m
> choice for converting the `int`. An explicit typ
> for `int` types.

To use `cin.get()` successfully, you need t
When the function reaches the EOF, there ar
`cin.get()` returns a special value, represented
is defined in the `iostream` header file. The `E`
character value so that the program won't co
`EOF` is defined as the value `-1` because no cha
don't need to know the actual value. You can
heart of Listing 5.18 looks like this:

You should realize that EOF does not represen
that there are no more characters.

There's a subtle but important point about so far. Because EOF represents a value outside might not be compatible with the char type. unsigned, so a char variable could never have you use cin.get() (with no argument) and t to type int instead of to type char. Also if yo you might have to do a type cast to char whe

Listing 5.19 incorporates the cin.get() ap also condenses the code by combining charac

Listing 5.19 **textin4.cpp**

```
// textin4.cpp -- reading chars with cin.g
#include <iostream>
int main(void)
{
    using namespace std;
    int ch;                          // sho
    int count = 0;

    while ((ch = cin.get()) != EOF) // tes
```

```
104 characters read
```

Let's analyze the loop condition:

```
while ((ch = cin.get()) != EOF)
```

The parentheses that enclose the subexpre
evaluate that expression first. To do the evalu
`cin.get()` function. Next, it assigns the func
an assignment statement is the value of the le
to the value of `ch`. If this value is `EOF`, the loo
test condition needs all the parentheses. Supp

```
while (ch = cin.get() != EOF)
```

The `!=` operator has higher precedence th
`cin.get()`'s return value to `EOF`. A comparis
`bool` value is converted to `0` or `1`, and that's t

Using `cin.get(ch)` (with an argument) f
any type problems. Remember that the `cin.`
value to `ch` at the EOF. In fact, it doesn't assi
called on to hold a non-`char` value. Table 5.3
`cin.get(char)` and `cin.get()`.

`cout.put()` methods of `iostream`. You just re
globally replace `getchar()` and `putchar()` w
the old code uses a type `int` variable for inpu
your implementation has multiple prototypes

Nested Loops and Two-Di

Earlier in this chapter you saw that the `for` lo
Now let's go a step further and look at how a
serves to handle two-dimensional arrays.

First, let's examine what a two-dimensiona
ter are termed *one-dimensional arrays* because y
of data. You can visualize a two-dimensional a
rows and columns of data. You can use a two-
quarterly sales figures for six separate districts,
you can use a two-dimensional array to repres
puterized game board.

C++ doesn't provide a special two-dimens
for which each element is itself an array. For e
mum temperature data for five cities over a 4-
array as follows:

```
int maxtemps[4][5];
```

The expression `maxtemps[0]` is the first e
`maxtemps[0]` is itself an array of five `ints`. Th
`maxtemps[0][0]`, and this element is a single
access the `int` elements. You can think of the
the second subscript as representing the colu

```
int maxtemps[4][5];
```

The `maxtemps` array viewed as a table:

		0	1
`maxtemps[0]`	0	`maxtemps[0][0]`	`maxtemps[0][1]`
`maxtemps[1]`	1	`maxtemps[1][0]`	`maxtemps[1][1]`
`maxtemps[2]`	2	`maxtemps[2][0]`	`maxtemps[2][1]`
`maxtemps[3]`	3	`maxtemps[3][0]`	`maxtemps[3][1]`

Figure 5.6 Accessing arr

For a two-dimensional array, each element
element by using a form like that in the previ
consists of a comma-separated series of one-d
set of braces:

```
int maxtemps[4][5] =   // 2-D array
{
    {96, 100, 87, 101, 105},   // values f
    {96, 98, 91, 107, 104},    // values f
    {97, 101, 93, 108, 107},   // values f
    {98, 103, 95, 109, 108}    // values f
};
```

You can visualize maxtemps as four rows o
`87, 103, 101}` initializes the first row, repres
placing each row of data on its own line, if po

Using a Two-Dimensional Array

Listing 5.20 incorporates an initialized two-di
program. This time the program reverses the o
(city index) on the outside and the row loop

```
    int maxtemps[Years][Cities] =   // 2-
    {
        {96, 100, 87, 101, 105},   // val
        {96, 98, 91, 107, 104},    // val
        {97, 101, 93, 108, 107},   // val
        {98, 103, 95, 109, 108}    // val
    };

    cout << "Maximum temperatures for 200
    for (int city = 0; city < Cities; ++c
    {
        cout << cities[city] << ":\t";
        for (int year = 0; year < Years;
            cout << maxtemps[year][city]
        cout << endl;
    }
        // cin.get();
    return 0;
}
```

```
    "Gribble Vista"
};
```

This approach limits each of the five string
of pointers stores the addresses of the five stri
copies each of the five string literals to the co
array of pointers is much more economical in
modify any of the strings, the two-dimension
enough, both choices use the same initializati
play the strings.

Also you could use an array of `string` clas
the string data. The declaration would look lik

```
const string cities[Cities] =   // array o
{
    "Gribble City",
    "Gribbletown",
    "New Gribble",
    "San Gribble",
    "Gribble Vista"
};
```

provides several ways to do this. If `ch` is a typ
reads the next input character into `ch`:

```
cin >> ch;
```

However, it skips over spaces, newlines, an
reads the next input character, regardless of it

```
cin.get(ch);
```

The member function call `cin.get()` ret
spaces, newlines, and tabs, so it can be used as

```
ch = cin.get();
```

The `cin.get(char)` member function ca
by returning a value with the `bool` conversio
function call reports the EOF by returning th
`iostream` file.

A nested loop is a loop within a loop. Nes
two-dimensional arrays.

5. What would the following code fragm

```
int k = 8;
do
      cout <<" k = " << k << endl;
while (k++ < 5);
```

6. Write a `for` loop that prints the values
a counting variable by a factor of two i

7. How do you make a loop body include

8. Is the following statement valid? If not,

```
int x = (1,024);
```

What about the following?

```
int y;
y = 1,024;
```

9. How does `cin>>ch` differ from `cin.ge`
input?

Cleo earns 5% of $100 the first year, gi
$105, or $5.25, and so on. Write a prog
the value of Cleo's investment to excee
displays the value of both investments

5. You sell the book *C++ for Fools*. Write
of monthly sales (in terms of number o
use a loop to prompt you by month, us
`string` objects, if you prefer) initialized
data in an array of `int`. Then, the progr
tents and report the total sales for the y

6. Do Programming Exercise 5 but use a
years of monthly sales. Report the tota
combined years.

7. Design a structure called `car` that hold
mobile: its make, as a string in a charac
it was built, as an integer. Write a progr
log. The program should then use `new`
structures. Next, it should prompt the u
of more than one word) and year infor
requires some care because it alternates

You should include the cstring heade
make the comparison test.

9. Write a program that matches the descr
Exercise 8, but use a string class objec
header file and use a relational operator

10. Write a program using nested loops tha
number of rows to display. It should the
one asterisk in the first row, two in the s
asterisks are preceded by the number of
display a total number of characters equ
would look like this:

```
Enter number of rows: 5
....*
...**
..***
.****
*****
```

- The `cctype` library of character functi
- The conditional operator: `?:`
- The `switch` statement
- The `continue` and `break` statements
- Number-reading loops
- Basic file input/output

One of the keys to designing intelligent p
decisions. Chapter 5, "Loops and Relational E
making—looping—in which a program deci
ter investigates how C++ lets you use branch
actions. Which vampire-protection scheme (g
menu choice has the user selected? Did the u
`switch` statements to implement decisions, a
chapter also looks at the conditional operato
choice, and the logical operators, which let y
chapter takes a first look at file input/output

```
    statement2
statement3
```

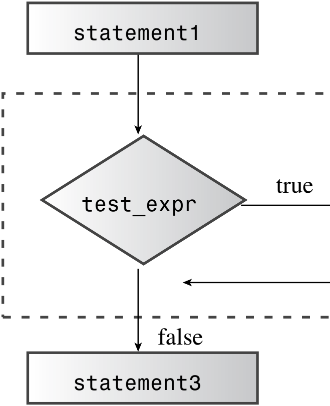


Figure 6.1 The struc

Most often, *test-condition* is a relationa
loops. Suppose, for example, that you want a p
as well as the total number of characters. You

```
    cout << " characters total in sentenc
    return 0;
}
```

Here's some sample output from the prog

```
The balloonist was an airhead
with lofty goals.
6 spaces, 46 characters total in sentence
```

As the comments in Listing 6.1 indicate, t
when `ch` is a space. Because it is outside the
executed in every loop cycle. Note that the t
that is generated by pressing Enter.

The `if else` Statement

Whereas an `if` statement lets a program deci
executed, an `if else` statement lets a progra
is executed. It's an invaluable statement for cr
C++ `if else` statement is modeled after sin
Cookie card, you get a Cookie Plus Plus, else
`else` statement has this general form:

```
statement3
    statement4
```

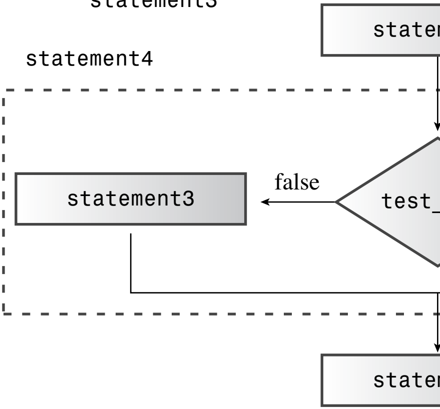


Figure 6.2 The structure

For example, suppose you want to alter inc
keeping the newline character intact. In that c
output line of equal length. This means you w
action for newline characters and a different c
Listing 6.2 shows, `if else` makes this task eas
fier, one of the alternatives to a `using` directiv

```
}
```

Here's some sample output from the prog

```
Type, and I shall repeat.
An ineffable joy suffused me as I beheld
Bo!jofggbcmf!kpz!tvggvtfe!nf!bt!J!cfifme
the wonders of modern computing.
uif!xpoefst!pg!npefso!dpnqvujoh
Please excuse the slight confusion.
```

Note that one of the comments in Listing
an interesting effect. Can you deduce what it
can explain what's happening. (Hint: Think a

Formatting `if else` Statement

Keep in mind that the two alternatives in an
ments. If you need more than one statement,
single block statement. Unlike some languag
does not automatically consider everything b
use braces to make the statements a block. Th
compiler error:

they enclose the statements. The preceding co
another:

```
if (ch == 'Z') {
    zorro++;
    cout << "Another Zorro candidate\n";
    }
else {
    dull++;
    cout << "Not a Zorro candidate\n";
    }
```

The first form emphasizes the block struct
form more closely ties the blocks to the keyw
consistent and should serve you well; however
employer with strong and specific views on th

The `if else if else` Construc

Computer programs, like life, might present y
tions. You can extend the C++ `if else` state
the `else` should be followed by a single staten
`else` statement itself is a single statement, it ca

Listing 6.3 uses this preferred formatting

Listing 6.3 **ifelseif.cpp**

```
// ifelseif.cpp -- using if else if else
#include <iostream>
const int Fave = 27;
int main()
{
    using namespace std;
    int n;

    cout << "Enter a number in the range
    cout << "my favorite number: ";
    do
    {
        cin >> n;
        if (n < Fave)
            cout << "Too low -- guess aga
        else if (n > Fave)
            cout << "Too high -- guess ag
        else
            cout << Fave << " is right!\n
    } while (n != Fave);
    return 0;
}
```

```
if (myNumber = 3)
```

The compiler would simply assign the value 3
would run—a very common error, and a difficu
issue a warning, which you would be wise to h
allows the compiler to find errors is much eas
faulty results.

Logical Expressions

Often you must test for more than one condit
lowercase letter, its value must be greater than
'z'. Or, if you ask a user to respond with a **y**
N) as well as lowercase. To meet this kind of ne
combine or modify existing expressions. The
AND, written &&; and logical NOT, written !

The Logical OR Operator: ||

In English, the word *or* can indicate when one
ment. For example, you can go to the MegaM
work for MegaMicro, Inc. The C++ equivaler
This operator combines two expressions into

ance of C++11, the subexpression to the left
subexpression to the right.) For example, con

```
i++ < 6 || i == j
```

Suppose `i` originally has the value `10`. By
has the value `11`. Also C++ won't bother eval
expression on the left is true, for it only takes
cal expression true. (The semicolon and the c
points.)

Listing 6.4 uses the `||` operator in an `if` s
lowercase versions of a character. Also it uses
Chapter 4, "Compound Types") to spread a s

Listing 6.4 **or.cpp**

```
// or.cpp -- using the logical OR operato
#include <iostream>
int main()
{
    using namespace std;
    cout << "This program may reformat yo
            "and destroy all your data.\n
            "Do you wish to continue? <y/
    char ch;
```

The Logical AND Operator: `&&`

The logical AND operator, written `&&`, also co
resulting expression has the value `true` only if
Here are some examples:

```
5 == 5 && 4 == 4   // true because both ex
5 == 3 && 4 == 4   // false because first
5 > 3 && 5 > 10    // false because second
5 > 8 && 5 < 10    // false because first
5 < 8 && 5 > 2     // true because both ex
5 > 8 && 5 < 2     // false because both e
```

Because the `&&` has a lower precedence tha
use parentheses in these expressions. Like the
sequence point, so the left side is evaluated, an
right side is evaluated. If the left side is false, th
C++ doesn't bother evaluating the right side
`&&` operator works.

```
{
    using namespace std;
    float naaq[ArSize];
    cout << "Enter the NAAQs (New Age Awa
         << "of\nyour neighbors. Program
         << "when you make\n" << ArSize <
         << "or enter a negative value.\n

    int i = 0;
    float temp;
    cout << "First value: ";
    cin >> temp;
    while (i < ArSize && temp >= 0) // 2
    {
        naaq[i] = temp;
        ++i;
        if (i < ArSize)             // ro
        {
            cout << "Next value: ";
            cin >> temp;            // so
        }
    }
    if (i == 0)
        cout << "No data--bye\n";
```

```
First value: 28
Next value: 72
Next value: 15
Next value: 6
Next value: 130
Next value: 145
Enter your NAAQ: 50
3 of your neighbors have greater awareness
the New Age than you do.
```

The second run terminates after a negative

```
Enter the NAAQs (New Age Awareness Quotien
your neighbors. Program terminates when yo
6 entries or enter a negative value.
First value: 123
Next value: 119
Next value: 4
Next value: 89
Next value: -1
Enter your NAAQ: 123.031
0 of your neighbors have greater awareness
the New Age than you do.
```

ber. Note that the loop reads another value i
is, only if there is still room left in the array.

After it gets data, the program uses an if
entered (that is, if the first entry was a negativ
present.

Setting Up Ranges with &&

The && operator also lets you set up a series o
choice corresponding to a particular range of
also shows a useful technique for handling a s
variable can identify a single string by pointi
char can identify a series of strings. You simp
ent array element. Listing 6.6 uses the quali
For example, qualify[1] holds the address o
gram can then use qualify[1] as it would ar
with cout or with strlen() or strcmp(). U
from accidental alterations.

```
    else if (age >= 50 && age < 65)
        index = 2;
    else
        index = 3;

    cout << "You qualify for the " << qual
    return 0;
}
```

Here is a sample run of the program in Lis

```
Enter your age in years: 87
You qualify for the pie-throwing festival.
```

The entered age doesn't match any of the t and then prints the corresponding string.

Program Notes

In Listing 6.6, the expression `age > 17 && ag` values—that is, ages in the range 18–34. The e `>=` operator to include 35 in its range, which i `age < 50`, the value 35 would be missed by al should check that the ranges don't have holes

The Logical NOT Operator: !

The ! operator negates, or reverses the truth
is, if expression is true, then !expression
expression is true, or nonzero, then !expre
call the exclamation point *bang*, making !x "b

Usually you can more clearly express a rel

```
if (!(x > 5))                   // if (x <=
```

But the ! operator can be useful with func
that can be interpreted that way. For example
value if the two C-style strings s1 and s2 are
they are the same. This implies that !strcmp(

Listing 6.7 uses the technique of applying
screen numeric input for suitability to be assi
is_int(), which we'll discuss further in a m
the range of values that can be assigned to ty
while(!is_int(num)) to reject values that d

Listing 6.7 **not.cpp**

```
// not.cpp -- using the not operator
#include <iostream>
#include <climits>
```

```
        return false;
}
```

Here is a sample run of the program in Lis

```
Yo, dude! Enter an integer value: 62341286
Out of range -- please try again: -8000222
Out of range -- please try again: 99999
You've entered the integer 99999
Bye
```

Program Notes

If you enter a too-large value to a program re
tions simply truncate the value to fit, without
gram in Listing 6.7 avoids that by first reading
type has more than enough precision to hold
greater. Another choice for holding the input
ing that it is wider than `int`.

The Boolean function `is_int()` uses the t
`INT_MIN`), defined in the `climits` file (discuss
determine whether its argument is within the
value of `true`; otherwise, it returns `false`.

values true or false, which get converted t

The logical AND operator has a higher pr
this expression:

```
age > 30 && age < 45 || weight > 300
```

means the following:

```
(age > 30 && age < 45) || weight > 300
```

That is, one condition is that `age` be in th
that `weight` be greater than 300. The entire e
of these conditions are `true`.

You can, of course, use parentheses to tell
example, suppose you want to use `&&` to com
50 or `weight` be greater than 300 with the co
1,000. You have to enclose the OR part with

```
(age > 50 || weight > 300) && donation >
```

Otherwise, the compiler combines the `wei`
instead of with the `age` condition.

Although the C++ operator precedence r
pound comparisons without using parenthese
parentheses to group the tests, whether or no
code easier to read, it doesn't force someone

Table 6.3 **Logical Operators: Alternative Rep**

Operator	Alternative Representation
`&&`	`and`
`||`	`or`
`!`	`not`

The `cctype` Library of Ch

C++ has inherited from C a handy package o the `cctype` header file (`ctype.h`, in the older whether a character is an uppercase letter or a `isalpha(ch)` function returns a nonzero valu Similarly, the `ispunct(ch)` function returns a acter, such as a comma or period. (These func but the usual `bool` conversions allow you to t

Using these functions is more convenient t example, here's how you might use AND and alphabetic character:

```
if ((ch >= 'a' && ch <= 'z') || (ch >= 'A'
```

```
char ch;
int whitespace = 0;
int digits = 0;
int chars = 0;
int punct = 0;
int others = 0;

cin.get(ch);                // get fi
while (ch != '@')            // test
{
    if(isalpha(ch))         // is it
        chars++;
    else if(isspace(ch))    // is it
        whitespace++;
    else if(isdigit(ch))    // is it
        digits++;
    else if(ispunct(ch))    // is it
        punct++;
    else
        others++;
    cin.get(ch);            // get ne
}
```

Function Name	Return Value
`isalnum()`	This function returns a letter or a digit).
`isalpha()`	This function returns
`isblank()`	This function returns tal tab.
`iscntrl()`	This function returns
`isdigit()`	This function returns
`isgraph()`	This function returns other than a space.
`islower()`	This function returns
`isprint()`	This function returns including a space.
`ispunct()`	This function returns character.
`isspace()`	This function returns space character (tha return, horizontal tab
`isupper()`	This function returns

expression3. Here are two examples that sh

```
5 > 3 ? 10 : 12  // 5 > 3 is true, so exp
3 == 9? 25 : 18  // 3 == 9 is false, so e
```

We can paraphrase the first example this w uates to 10; otherwise, it evaluates to 12. In re expressions would involve variables.

Listing 6.9 uses the conditional operator t

Listing 6.9 **condit.cpp**

```
// condit.cpp -- using the conditional op
#include <iostream>
int main()
{
    using namespace std;
    int a, b;
    cout << "Enter two integers: ";
    cin >> a >> b;
    cout << "The larger of " << a << " an
    int c = a > b ? a : b;   // c = a if
    cout << " is " << c << endl;
    return 0;
}
```

the following mild example shows:

```
const char x[2] [20] = {"Jason ","at your
const char * y = "Quillstone ";

for (int i = 0; i < 3; i++)
    cout << ((i < 2)? !i ? x [i] : y : x[1
```

This is merely an obscure (but, by no mear
strings in the following order:

```
Jason Quillstone at your service
```

In terms of readability, the conditional ope
and simple expression values:

```
x = (x > y) ? x : y;
```

If the code becomes more involved, it can
`else` statement.

The `switch` Statement

Suppose you create a screen menu that asks th
example, Cheap, Moderate, Expensive, Extrava
`else if else` sequence to handle five alterna

a very important way. Each C++ case label
ary between choices. That is, after a program
sequentially executes all the statements follow
itly direct it otherwise. Execution does *not* au
execution stop at the end of a particular grou
statement. This causes execution to jump to t

Listing 6.10 shows how to use `switch` and
menu for executives. The program uses a sho
A `switch` statement then selects an action ba

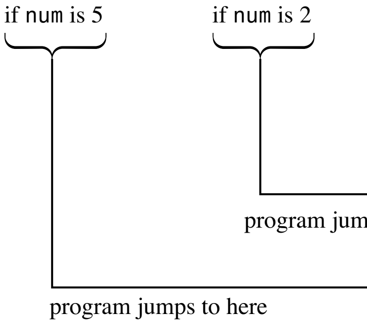


Figure 6.3 The structu

```
            case 2  :   report();
                        break;
            case 3  :   cout << "The boss
                        break;
            case 4  :   comfort();
                        break;
            default :   cout << "That's no
        }
        showmenu();
        cin >> choice;
    }
    cout << "Bye!\n";
    return 0;
}

void showmenu()
{
    cout << "Please enter 1, 2, 3, 4, or 5
            "1) alarm           2) report\
            "3) alibi           4) comfort
            "5) quit\n";
}
void report()
```

```
3) alibi              4) comfort
5) quit
2
It's been an excellent week for business.
Sales are up 120%. Expenses are down 35%.
Please enter 1, 2, 3, 4, or 5:
1) alarm              2) report
3) alibi              4) comfort
5) quit
6
That's not a choice.
Please enter 1, 2, 3, 4, or 5:
1) alarm              2) report
3) alibi              4) comfort
5) quit
5
Bye!
```

The `while` loop terminates when the user
corresponding choice from the `switch` list, a

Note that input has to be an integer for th
you enter a letter, the input statement will fai
kill the program. To deal with those who do
acter input.

```
                break;
        case 'c':
        case 'C': comfort();
                break;
        default : cout << "That's not a ch
    }
    showmenu();
    cin >> choice;
}
```

Because there is no `break` immediately fol
on to the next line, which is the statement fol

Using Enumerators as Labels

Listing 6.11 illustrates using `enum` to define a s
constants in a `switch` statement. In general, `ci`
can't know how you will define them), so the
the `switch` statement compares the `int` value
enumerator to `int`. Also the enumerators are p
condition.

```
            case indigo  : cout << "Her n
        }
        cout << "Enter color code (0-6):
        cin >> code;
    }
    cout << "Bye\n";
    return 0;
}
```

Here's sample output from the program in

```
Enter color code (0-6): 3
Her nails were green.
Enter color code (0-6): 5
Her eyes were violet.
Enter color code (0-6): 2
Her shoes were yellow.
Enter color code (0-6): 8
Bye
```

`switch` and `if else`

Both the `switch` statement and the `if else` alternatives. The `if else` is the more versatil ranges, as in the following:

is to use switch if you have three or more al

The break and continu

The break and continue statements enable a
can use the break statement in a switch state
gram execution to pass to the next statement
continue statement is used in loops and cause
the loop and then start a new loop cycle (see

Listing 6.12 shows how the two statements
text. The loop echoes each character and uses
is a period. This shows how you can use breal
some condition becomes true. Next the prog
The loop uses continue to skip over the cour
isn't a space.

Listing 6.12 **jump.cpp**

```
// jump.cpp -- using continue and break
#include <iostream>
const int ArSize = 80;
int main()
{
    using namespace std;
```

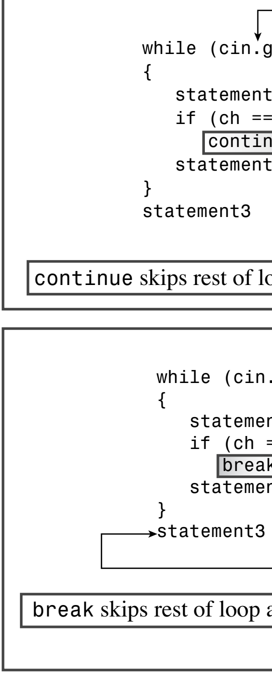


Figure 6.4 The structure of c

However, the continue statement can mak
statements follow the continue. That way, yo
part of an if statement.

C++, like C, also has a goto statement. A s
location bearing the paris: label:

```
goto paris;
```

That is, you can have code like this:

```
char ch;
cin >> ch;
if (ch == 'P')
      goto paris;
cout << ...
...
paris: cout << "You've just arrived at Par
```

In most circumstances (some would say in
and you should use structured controls, such a
to control program flow.

look at a couple examples that illustrate these

Say you want to write a program that calc
of fish. There's a five-fish limit, so a five-elem
ble that you could catch fewer fish. Listing 6.
full or if you enter non-numeric input.

Listing 6.13 **cinfish.cpp**

```
// cinfish.cpp -- non-numeric input termi
#include <iostream>
const int Max = 5;
int main()
{
    using namespace std;
// get data
    double fish[Max];
    cout << "Please enter the weights of
    cout << "You may enter up to " << Max
            << " fish <q to terminate>.\n
    cout << "fish #1: ";
    int i = 0;
    while (i < Max && cin >> fish[i]) {
        if (++i < Max)
            cout << "fish #" << i+1 << ":
```

```
    cin.clear();  // reset input
    cin.get();    // read q
}
cin.get();        // read end of line a
cin.get();        // wait for user to p
```

You also could use code similar to this in Lis
more input after exiting the loop.

Listing 6.14 further illustrates using the `cin`

The expression `cin >> fish[i]` in Listing
and the function returns `cin`. If `cin` is part of
The conversion value is `true` if input succeed
the expression terminates the loop. By the wa

```
Please enter the weights of your fish.
You may enter up to 5 fish <q to terminate
fish #1: 30
fish #2: 35
fish #3: 25
fish #4: 40
fish #5: q
32.5 = average weight of 4 fish
Done.
```

Listing 6.14 **cingolf.cpp**

```
// cingolf.cpp -- non-numeric input skipp
#include <iostream>
const int Max = 5;
int main()
{
    using namespace std;
// get data
    int golf[Max];
    cout << "Please enter your golf score
    cout << "You must enter " << Max << "
    int i;
    for (i = 0; i < Max; i++)
    {
        cout << "round #" << i+1 << ": ";
        while (!(cin >> golf[i])) {
            cin.clear();     // reset inp
            while (cin.get() != '\n')
                continue;    // get rid o
            cout << "Please enter a numbe
        }
    }
// calculate average
```

Program Notes

The heart of the error-handling code in Listin

```
while (!(cin >> golf[i])) {
    cin.clear();     // reset input
    while (cin.get() != '\n')
        continue; // get rid of bad input
    cout << "Please enter a number: ";
}
```

If the user enters **`88`**, the `cin` expression is thermore, because `cin` is `true`, the expression inner loop terminates. But if the user enters **`m`** is placed into the array, the expression `!(cin` enters the inner `while` loop. The first statemen reset input. If you omit this statement, the pro the program uses `cin.get()` in a `while` loop of the line. This gets rid of the bad input, alon approach is to read to the next whitespace, wh instead of one line at a time. Finally, the progr

Let's see how this line of input is handled
First, let's try type `char`:

```
char ch;
cin >> ch;
```

The first character in the input line is assi
the digit `3`, and the character code (in binary
the destination are both characters, so no tra
numeric value `3` that is stored; rather, it is the
input statement, the digit character `8` is the n
the next character examined by the next inpu

Next, let's try the `int` type with the same

```
int n;
cin >> n;
```

In this case, `cin` reads up to the first non-
the `8` digit, leaving the period as the next ch
putes that these two characters correspond to
for `38` is copied to `n`.

Next, let's try the `double` type:

```
double x;
cin >> x;
```

On output, the opposite translations take p
sequences of digit characters, and floating-poi
digits and other characters (for example, 284.5
no translation.

The main point to this is that all the input
lent to console input is a text file—that is, a fi
code. Not all files are text files. For example, d
in numeric forms—that is, in binary integer o
processing files may contain text information,
describe formatting, fonts, printers, and the lik

The file I/O discussed in this chapter paral
with text files. To create a text file for input, y
Windows, or vi or emacs for Unix/Linux. You
save the file in text format. The code editors t
indeed, the source code files are examples of t
to look at files created with text output.

Writing to a Text File

For file output, C++ uses analogs to `cout`. So
basic facts about using `cout` for console outpu

Note that although the iostream header
called cout, you have to declare your own of
associating it with a file. Here's how you decl

```
ofstream outFile;           // outFile an
ofstream fout;              // fout an of
```

Here's how you can associate the objects w

```
outFile.open("fish.txt");   // outFile us
char filename[50];
cin >> filename;            // user speci
fout.open(filename);        // fout used
```

Note that the open() method requires a C
literal string or a string stored in an array.

Here's how you can use these objects:

```
double wt = 125.8;
outFile << wt;          // write a number
char line[81] = "Objects are closer than
fout << line << endl;   // write a line o
```

The important point is that after you've d
with a file, you use it exactly as you would us

```
    char automobile[50];
    int year;
    double a_price;
    double d_price;

    ofstream outFile;               // cre
    outFile.open("carinfo.txt");    // ass

    cout << "Enter the make and model of a
    cin.getline(automobile, 50);
    cout << "Enter the model year: ";
    cin >> year;
    cout << "Enter the original asking pri
    cin >> a_price;
    d_price = 0.913 * a_price;

// display information on screen with cout

    cout << fixed;
    cout.precision(2);
    cout.setf(ios_base::showpoint);
    cout << "Make and model: " << automobi
    cout << "Year: " << year << endl;
```

```
Enter the original asking price: 13500
Make and model: Flitz Perky
Year: 2009
Was asking $13500.00
Now asking $12325.50
```

The screen output comes from using `cout`. If you ch
contains the executable program, you should find a nev
may be in some other folder, depending on how the co
the output generated by using `outFile`. If you open it
the following contents:

```
Make and model: Flitz Perky
Year: 2009
Was asking $13500.00
Now asking $12325.50
```

As you can see, `outFile` sends precisely the same seque
`carinfo.txt` file that `cout` sends to the display.

Program Notes

After the program in Listing 6.15 declares an `ofstream`
method to associate the object with a particular file:

```
ofstream outFile;              // create object f
outFile.open("carinfo.txt");   // associate with
```

tents. The contents are then replaced with the
override this default behavior.

> **Caution**
>
> When you open an existing file for output, by
> so the contents are lost.

It is possible that an attempt to open a file
having the requested name might already exis
careful programmer would check to see if the
nique for this in the next example.

Reading from a Text File

Next, let's examine text file input. It's based o
So let's begin with a summary those elements

- You must include the `iostream` header
- The `iostream` header file defines an `is`
- The `iostream` header file declares an `i`
- You must account for the `std` namespac
  tive or the `std::` prefix for elements su

- You can use an ifstream object with t
- You can use an ifstream object with t ters and with the getline() method t
- You can use an ifstream object with r tor the success of an input attempt.
- An ifstream object itself, when used a Boolean value true if the last read atte

Note that although the iostream header called cin, you have to declare your own ifs associating it with a file. Here's how you decl

```
ifstream inFile;             // inFile an
ifstream fin;                // fin an if
```

Here's how you can associate them with p

```
inFile.open("bowling.txt");  // inFile us
char filename[50];
cin >> filename;             // user spec
fin.open(filename);          // fin used
```

Note that the open() method requires a C literal string or a string stored in an array.

the cstdlib header file, which also defines EX
communicate with the operating system. The

The is_open() method is relatively new t
you can use the older good() method instead
check quite as extensively as is_open() for p

The program in Listing 6.16 opens a file sp
file, and reports the number of values, their su
design the input loop correctly, and the follow
in more detail. Notice that this program bene

Listing 6.16 **sumafile.cpp**

```
// sumafile.cpp -- functions with an array
#include <iostream>
#include <fstream>          // file I/O su
#include <cstdlib>          // support for
const int SIZE = 60;
int main()
{
    using namespace std;
    char filename[SIZE];
    ifstream inFile;        // object for
```

```
        cout << "Input terminated by data
    else
        cout << "Input terminated for unk
    if (count == 0)
        cout << "No data processed.\n";
    else
    {
        cout << "Items read: " << count <
        cout << "Sum: " << sum << endl;
        cout << "Average: " << sum / coun
    }
    inFile.close();         // finished w
    return 0;
}
```

To use the program in Listing 6.16, you fi numbers. You can use a text editor, such as th to create this file. Let's assume that the file is contents:

```
18 19 18.5 13.5 14
16 19.5 20 18 12 18.5
17.5
```

name in the character array `filename`. Then t

```
inFile.open(filename);
```

As discussed earlier in this chapter, it's vital succeeded. Here are a few of the things that m file might be located in another directory or f user might mistype the name or omit a file ex time trying to figure what's wrong with a file that the program didn't open the file. Testing misspent effort.

You need to pay close attention to the pro several things to test for when reading from a read past the EOF. The `eof()` method returns data ran into the EOF. Second, the program m instance, Listing 6.16 expects a file containing `true` if the most recent read attempt encount returns `true` if the EOF is encountered.) Fina for example, a corrupted file or a hardware fa most recent read attempt encountered such a tions individually, it's simpler to use the `good(` when wrong:

immediately before the loop, just before the
input statement at the end of the loop, just b

```
// standard file-reading loop design
inFile >> value;        // get first valu
while (inFile.good())   // while input go
{
    // loop body goes here
    inFile >> value;    // get next value
}
```

You can condense this somewhat by using
ates to `inFile` and that `inFile`, when placed
expected, evaluates to `inFile.good()`—that

```
inFile >> value
```

Thus, you can replace the two input stater
loop test. That is, you can replace the precedi

```
// abbreviated file-reading loop design
// omit pre-loop input
while (inFile >> value)   // read and tes
{
    // loop body goes here
    // omit end-of-loop input
}
```

The conditional operator (?:) provides a com

The cctype library of character functions tools for analyzing character input.

Loops and selection statements are useful t console I/O. After you declare ifstream and files, you can use these objects in the same ma

With C++'s loops and decision-making st interesting, intelligent, and powerful programs real powers of C++. Next, we'll look at funct

Chapter Review

1. Consider the following two code fragm

```
// Version 1
while (cin.get(ch))    // quit on eo
{
      if (ch == ' ')
             spaces++;
      if (ch == '\n')
            newlines++;
}

// Version 2
```

```
        cout << ch;
        ct1++;
        if (ch = '$')
            ct2++;
        cout << ch;
    }
    cout <<"ct1 = " << ct1 << ", ct
    return 0;
}
```

Suppose you provide the following inp
line:

Hi!

Send $10 or $20 now!

What is the output? (Recall that input

4. Construct logical expressions to repres

   a. `weight` is greater than or equal t

   b. `ch` is `q` or `Q`.

   c. `x` is even but is not `26`.

   d. `x` is even but is not a multiple of

   e. `donation` is in the range 1,000–

```
        f_grade++;
```

8. In Listing 6.10, what advantage would t
   and c, instead of numbers for the menu
   about what happens if the user types q i
   types **5** in either case.)

9. Consider the following code fragment:

```
int line = 0;
char ch;
while (cin.get(ch))
{
    if (ch == 'Q')
           break;
    if (ch != '\n')
           continue;
    line++;
}
```

Rewrite this code without using break

```
Please enter a c, p, t, or g: t
A maple is a tree.
```

4. When you join the Benevolent Order meetings by your real name, your job t gram that can list members by real nam ber's preference. Base the program on t

```
// Benevolent Order of Programmers 
struct bop {
    char fullname[strsize]; // real
    char title[strsize];    // job 
    char bopname[strsize];  // secr
    int preference;         // 0 = 
};
```

In the program, create a small array of values. Have the program run a loop th tives:

```
a. display by name     b. display by
c. display by bopname  d. display by
q. quit
```

5. The Kingdom of Neutronia, where the
lowing income tax code:

   First 5,000 tvarps: 0% tax

   Next 10,000 tvarps: 10% tax

   Next 20,000 tvarps: 15% tax

   Tvarps after 35,000: 20% tax

   For example, someone earning 38,000 t
   0.10 + 20,000   0.15 + 3,000   0.20, o
   loop to solicit incomes and to report ta
   the user enters a negative number or no

6. Put together a program that keeps track
   for the Preservation of Rightful Influen
   ber of contributors and then solicit the
   each contributor. The information shou
   of structures. Each structure should hav
   `string` object) to store the name and a
   contribution. After reading all the data,
   amounts donated for all donors who co

9. Do Programming Exercise 6 but modi
item in the file should be the number
consist of pairs of lines, with the first li
and the second line being a contributi

```
4
Sam Stone
2000
Freida Flass
100500
Tammy Tubbs
5000
Rich Raptor
55000
```

- Designing functions to process arrays
- Using `const` pointer parameters
- Designing functions to process text stri
- Designing functions to process structur
- Designing functions to process objects
- Functions that call themselves (recursio
- Pointers to functions

Fun is where you find it. Look closely, and
with a large library of useful functions (the st
classes), but real programming pleasure come
other hand, real programming productivity c
can do with the STL and the BOOST C++
"Adventures in Functions," examine how to
them, and retrieve information from them. A
chapter concentrates on how to use function
structures. Finally, it touches on recursion an
dues, you'll find much of this chapter familia
expertise. C++ has made several additions to
deals primarily with those. Meanwhile, let's a

```
// calling.cpp -- defining, prototyping, a
#include <iostream>

void simple();    // function prototype

int main()
{
    using namespace std;
    cout << "main() will call the simple()
    simple();     // function call
        cout << "main() is finished with t
    // cin.get();
    return 0;
}

// function definition
void simple()
{
    using namespace std;
    cout << "I'm but a simple function.\n"
}
```

```
}
```

Here *parameterList* specifies the types a to the function. This chapter more fully inves statement marks the end of the function. Oth ing brace. Type `void` functions correspond to and modern BASIC subprogram procedures. form some sort of action. For example, a fun times could look like this:

```
void cheers(int n)          // no return
{

    for (int i = 0; i < n; i++)
        std::cout << "Cheers! ";
    std::cout << std::endl;
}
```

The `int n` parameter list means that `chee` it as an argument when you call this function

A function with a return value produces a called it. In other words, if the function retur

return value to a specified CPU register or m
examines that location. Both the returning fu
agree on the type of data at that location. The
what to expect, and the function definition te
Figure 7.1). Providing the same information i
seem like extra work, but it makes good sense
something from your desk at the office, you e
right if you provide a description of what you
at the office.

A function terminates after executing a ret
one return statement—for example, as alterna
function terminates after it executes the first r
the following example, the `else` isn't needed,
stand the intent:

```
int bigger(int a, int b)
{
    if (a > b )
        return a;  // if a > b, function t
    else
        return b;  // otherwise, function
}
```

(Usually, having multiple return statemen
fusing, and some compilers might issue a wa
enough to understand.)

Functions with return values are much lik
BASIC. They return a value to the calling pr
variable, display the value, or otherwise use i
cube of a type double value:

```
double cube(double x)    // x times x tim
{
    return x * x * x; // a type double va
}
```

For example, the function call cube(1.2)
return statement uses an expression. The fun
(1.728, in this case) and returns the value.

Prototyping and Calling a Functi

By now you are familiar with making functi
with function prototyping because that's ofte
7.2 shows the cheers() and cube() functio
prototypes.

```
    for (int i = 0; i < n; i++)
        cout << "Cheers! ";
    cout << endl;
}

double cube(double x)
{
    return x * x * x;
}
```

The program in Listing 7.2 places a using
the members of the std namespace. Here's a s

```
Cheers! Cheers! Cheers! Cheers! Cheers!
Give me a number: 5
A 5-foot cube has a volume of 125 cubic fe
Cheers! Cheers! Cheers! Cheers! Cheers! Ch
```

Note that main() calls the type void funct
and arguments followed by a semicolon: chee
statement. But because cube() has a return va
ment statement:

```
double volume = cube(side);
```

could only guess, and that is something comp

Still, you might wonder, why does the co further in the file and see how the functions approach is that it is not very efficient. The c `main()` on hold while searching the rest of t fact that the function might not even be in t over several files, which you can compile ind case, the compiler might not have access to t `main()`. The same is true if the function is pa function prototype is to place the function d possible. Also the C++ programming style is vides the structure for the whole program.

Prototype Syntax

A function prototype is a statement, so it mu way to get a prototype is to copy the functio add a semicolon. That's what the program in

```
double cube(double x); // add ; to header
```

```
void say_bye(...);   // C++ abdication
```

Normally this use of an ellipsis is needed onl
able number of arguments, such as `printf(`

What Prototypes Do for You

You've seen that prototypes help the compiler
reduce the chances of program errors. In parti

- The compiler correctly handles the fun
- The compiler checks that you use the c
- The compiler checks that you use the c
converts the arguments to the correct ty

We've already discussed how to correctly h
what happens when you use the wrong numb
make the following call:

```
double z = cube();
```

A compiler that doesn't use function proto
called, it looks where the call to `cube()` shoul
value happens to be there. This is how C worl

verted correctly to a mere `int`. Some compil
is an automatic conversion from a larger type

Also prototyping results in type conversio
example, convert an integer to a structure or

Prototyping takes place during compile ti
type checking, as you've just seen, catches ma
catch during runtime.

Function Arguments and

It's time to take a closer look at function argu
value. That means the numeric value of the ar
assigned to a new variable. For example, Listi

```
double volume = cube(side);
```

Here `side` is a variable that, in the sample
`cube()`, recall, was this:

```
double cube(double x)
```

When this function is called, it creates a n
izes it with the value `5`. This insulates data in
`cube()` because `cube()` works with a copy o

Variables, including parameters, declared w
When a function is called, the computer alloc
When the function terminates, the computer
variables. (Some C++ literature refers to this a
and destroying variables. That does make it soun
called *local variables* because they are localized
this helps preserve data integrity. It also means
`main()` and another variable called `x` in some
lated variables, much as the Albany in Californ
(see Figure 7.3). Such variables are also termed
cated and deallocated automatically during pr

Multiple Arguments

A function can have more than one argument.
arguments with commas:

```
n_chars('R', 25);
```

This passes two arguments to the function n i y

variables in `main()`

Figure 7.3

Similarly, when you define the function, y
declarations in the function header:

```
void n_chars(char c, int n)  // two argum
```

This function header states that the functi
and one type `int` argument. The parameters
to the function. If a function has two parame
type of each parameter separately. You can't c
you declare regular variables:

```
void fifi(float a, float b)  // declare e
void fufu(float a, b)        // NOT accep
```

As with other functions, you just add a se

```
void n_chars(char c, int n); // prototype
```

As with single arguments, you don't have
type as in the definition, and you can omit th

```
void n_chars(char, int);     // prototype
```

```
    {
        cout << "Enter an integer: ";
        cin >> times;
        n_chars(ch, times); // function wi
        cout << "\nEnter another character
                " q-key to quit: ";
           cin >> ch;
    }
    cout << "The value of times is " << ti
    cout << "Bye\n";
    return 0;
}

void n_chars(char c, int n) // displays c
{
    while (n-- > 0)         // continue un
        cout << c;
}
```

The program in Listing 7.3 illustrates placi
initions rather than within the functions. Here

```
Enter a character: W
Enter an integer: 50
```

```
while (n-- > 0)          // continue until
    cout << c;
```

Notice that the program keeps count by d
formal parameter from the argument list. Thi
variable in `main()`. The `while` loop then dec
strates, changing the value of `n` has no effect
of `times` in `main()`, the value of `n` in `main()`
`n_chars()`.

Another Two-Argument Function

Let's create a more ambitious function—one
Also the function illustrates the use of local
arguments.

Many states in the United States now spo
Lotto lets you pick a certain number of choi
to pick six numbers from a card having 51 n
numbers at random. If your choice exactly m
or so. Our function will calculate the probab
function that successfully predicts the winnin
C++, although powerful, has yet to impleme

For example, compare

```
(10 * 9) / (2 * 1)
```

with

```
(10 / 2) * (9 / 1)
```

The first evaluates to 90 / 2 and then to 45
then to 45. Both give the same answer, but th
ate value (90) than does the second. The more
gets. For large numbers, this strategy of altern
the calculation from overflowing the maximu

Listing 7.4 incorporates this formula into
number of picks and the total number of cho
uses the `unsigned int` type (`unsigned`, for s
eral integers can produce pretty large results,
for the function's return value. Also terms suc
integer types.

> **Note**
>
> Some C++ implementations don't support typ
> into that category, try ordinary `double` instea

```
// the following function calculates the
// numbers correctly from numbers choices
long double probability(unsigned numbers,
{
    long double result = 1.0;  // here co
    long double n;
    unsigned p;

    for (n = numbers, p = picks; p > 0; n
        result = result * n / p ;
    return result;
}
```

Here's a sample run of the program in Lis

```
Enter the total number of choices on the
the number of picks allowed:
49 6
You have one chance in 1.39838e+007 of wi
Next two numbers (q to quit): 51 6
You have one chance in 1.80095e+007 of wi
Next two numbers (q to quit): 38 6
You have one chance in 2.76068e+006 of wi
Next two numbers (q to quit): q
bye
```

corresponds to the number of cookies that pe
That's easy to find; you just use a loop to add
ments is such a common task that it makes se
you won't have to write a new loop every tim

Let's consider what the function interface i
sum, it should return the answer. If you keep y
with a type int return value. So that the func
pass the array name as an argument. And to m
restricted to an array of a particular size, you p
ingredient here is that you have to declare tha
name. Let's see what that and the rest of the fu

```
int sum_arr(int arr[], int n) // arr = arr
```

This looks plausible. The brackets seem to
the brackets are empty seems to indicate that
any size. But things are not always as they seem
The good news is that you can write the rest
First, let's use an example to check that this ap
it works.

Listing 7.5 illustrates using a pointer as if it
the array to some values and uses the sum_arr
the sum_arr() function uses arr as if it were

```
    for (int i = 0; i < n; i++)
        total = total + arr[i];
    return total;
}
```

Here is the output of the program in Listi

```
Total cookies eaten: 255
```

As you can see, the program works. Now I

How Pointers Enable Array-Proce

The key to the program in Listing 7.5 is that name of an array as if it were a pointer. Reca C++ interprets an array name as the address

```
cookies == &cookies[0]  // array name is
```

(There are a few exceptions to this rule. F to label the storage. Second, applying `sizeof` array, in bytes. Third, as mentioned in Chapte array name returns the address of the whole a address of a 32-byte block of memory if `int`

of an array. Whether arr is a pointer or an arr
fourth element of the array. And it probably w
of the following two identities:

```
arr[i] == *(ar + i)    // values in two no
&arr[i] == ar + i      // addresses in two
```

Remember that adding one to a pointer, i
value equal to the size, in bytes, of the type to
and array subscription are two equivalent way
of an array.

The Implications of Using Arrays a

Let's look at the implications of Listing 7.5. Th
ArSize) passes the address of the first elemen
elements of the array to the sum_arr() functi
cookies address to the pointer variable arr an
This means Listing 7.5 doesn't really pass the a
the function where the array is (the address), w
how many elements it has (the n variable). (Se
the function then uses the original array. If yo
works with a copy. But if you pass an array, the
this difference doesn't violate C++'s pass-by-v

memory, but it has to spend time copying lar
ing with the original data raises the possibilit
problem in classic C, but ANSI C and C++'s
soon see an example. But first, let's alter Listin
array functions operate. Listing 7.6 demonstra
value. It also shows how the pointer concept
than it may have appeared at first. To provide
looks like, the program uses the `std::` qualifi
access to `cout` and `endl`.

Listing 7.6 **arrfun2.cpp**

```
// arrfun2.cpp -- functions with an array
#include <iostream>
const int ArSize = 8;
int sum_arr(int arr[], int n);
// use std:: instead of using directive
int main()
{
    int cookies[ArSize] = {1,2,4,8,16,32,
//  some systems require preceding int wi
//  enable array initialization
```

```
    return total;
}
```

Here's the output of the program in Listing

```
003EF9FC = array address, 32 = sizeof cook
003EF9FC = arr, 4 = sizeof arr
Total cookies eaten: 255
003EF9FC = arr, 4 = sizeof arr
First three eaters ate 7 cookies.
003EFA0C = arr, 4 = sizeof arr
Last four eaters ate 240 cookies.
```

Note that the address values and the array
system. Also some implementations will displa
of in hexadecimal. Others will use hexadecim

Program Notes

Listing 7.6 illustrates some very interesting po
`cookies` and `arr` both evaluate to the same ad
`cookies` is `32`, whereas `sizeof arr` is only `4`.
of the whole array, whereas `sizeof arr` is the
execution takes place on a system that uses 4-

```
void fillArray(int arr[], int size);
```

Don't try to pass the array size by using brac

```
void fillArray(int arr[size]);
```

More Array Function Examples

When you choose to use an array to represen design decisions should go beyond how data data is used. Often you'll find it profitable to data operations. (The profits here include inc tion, and ease of debugging.) Also when you operations when you think about a program, OOP mind-set; that, too, might prove profita

Let's examine a simple case. Suppose you lar values of your real estate. (If necessary, sup what type to use. Certainly, `double` is less res provides enough significant digits to represen decide on the number of array elements. (Wi put off that decision, but let's keep things sim five properties, so you can use an array of five

Now consider the possible operations you array. Two very basic ones are reading values

number of items to be read, and the function
example, if you use this function with an array
argument. If you then enter only three values,

You can use a loop to read successive value
the loop early? One way is to use a special val
property should have a negative value, you can
of input. Also the function should do somethi
ther input. Given these considerations, you ca

```
int fill_array(double ar[], int limit)
{
    using namespace std;
    double temp;
    int i;
    for (i = 0; i < limit; i++)
    {
        cout << "Enter value #" << (i + 1)
        cin >> temp;
        if (!cin)    // bad input
        {
            cin.clear();
            while (cin.get() != '\n')
                continue;
           cout << "Bad input; input proce
```

arguments because C++ passes them by value
functions that use an array work with the orig
function is able to do its job. To keep a functio
an array argument, you can use the keyword c
Data") when you declare the formal argumen

```
void show_array(const double ar[], int n)
```

The declaration states that the pointer `ar` can't use `ar` to change the data. That is, you c change that value. Note that this doesn't mea just means that you can't use `ar` in the `show_` `show_array()` treats the array as read-only d restriction by doing something like the follov

```
ar[0] += 10;
```

In this case, the compiler will put a stop to example, gives an error message like this (edit

```
Cannot modify a const object in function
     show_array(const double *,int)
```

Other compilers may choose to express th

```
            ar[i] *= r;
}
```

Because this function is supposed to alter t
you declare ar.

Putting the Pieces Together

Now that you've defined a data type in terms
used (three functions), you can put together a
you've already built all the array-handling too
main(). The program does check to see if the
tion factor with a number. In this case, rather
program uses a loop to ask the user to do the
ming work consists of having main() call the
shows the result. It places a using directive in
facilities.

Listing 7.7 **arrfun3.cpp**

```
// arrfun3.cpp -- array functions and cons
#include <iostream>
const int Max = 5;
```

```
        revalue(factor, properties, size)
        show_array(properties, size);
    }
    cout << "Done.\n";
    cin.get();
    cin.get();
    return 0;
}

int fill_array(double ar[], int limit)
{
    using namespace std;
    double temp;
    int i;
    for (i = 0; i < limit; i++)
    {
        cout << "Enter value #" << (i + 1
        cin >> temp;
        if (!cin)    // bad input
        {
            cin.clear();
            while (cin.get() != '\n')
                continue;
           cout << "Bad input; input proc
```

```
{
    for (int i = 0; i < n; i++)
        ar[i] *= r;
}
```

Here are two sample runs of the program i

```
Enter value #1: 100000
Enter value #2: 80000
Enter value #3: 222000
Enter value #4: 240000
Enter value #5: 118000
Property #1: $100000
Property #2: $80000
Property #3: $222000
Property #4: $240000
Property #5: $118000
Enter revaluation factor: 0.8
Property #1: $80000
Property #2: $64000
Property #3: $177600
Property #4: $192000
Property #5: $94400
Done.
```

into a program. This is sometimes called *botto*
moves from the component parts to the who
which concentrates on data representation an
programming, on the other hand, leans towar
develop a modular grand design first and the
methods are useful, and both lead to modula

The Usual Array Function Idiom

Suppose you want a function to process an a
is intended to modify the array, the prototyp

```
void f_modify(double ar[], int n);
```

If the function preserves values, the protot

```
void _f_no_change(const double ar[], int
```

Of course, you can omit the variable name
be something other than `void`. The main poi
element of the passed array and that because t
ment, either function can be used with any si

```
double rewards[1000];
double faults[50];
...
f_modify(rewards, 1000);
f_modify(faults, 50);
```

```
double elbuod[20];
```

Then the two pointers `elbuod` and `elbuod` the name of the array, points to the first eleme the last element (that is, `elbuod[19]`), so `elbu` array. Passing a range to a function tells it whi Listing 7.6 to use two pointers to specify a ran

Listing 7.8 **arrfun4.cpp**

```
// arrfun4.cpp -- functions with an array
#include <iostream>
const int ArSize = 8;
int sum_arr(const int * begin, const int *
int main()
{
    using namespace std;
    int cookies[ArSize] = {1,2,4,8,16,32,6
//  some systems require preceding int wit
//  enable array initialization

    int sum = sum_arr(cookies, cookies + A
    cout << "Total cookies eaten: " << sum
    sum = sum_arr(cookies, cookies + 3);
    cout << "First three eaters ate " << s
```

Program Notes

In Listing 7.8, notice the `for` loop in the `sum

```
for (pt = begin; pt != end; pt++)
    total = total + *pt;
```

It sets `pt` to point to the first element to b
and adds `*pt` (the value of the element) to `to
menting it, causing it to point to the next ele
`end`. When `pt` finally equals `end`, it's pointing
the range, so the loop halts.

Second, notice how the different function

```
int sum = sum_arr(cookies, cookies + ArSi
...
sum = sum_arr(cookies, cookies + 3);
...
sum = sum_arr(cookies + 4, cookies + 8);
```

The pointer value `cookies + ArSize` poi
ment. (The array has `ArSize` elements, so `coo
its address is `cookies + ArSize - 1`.) So the
fies the entire array. Similarly, `cookies, cook
and so on.

```
cin >> *pt;        // INVALID for the same
```

Now for a subtle point. This declaration fo
it points to is really a constant; it just means th
cerned. For example, pt points to age, and ag
age directly by using the age variable, but you
pointer:

```
*pt = 20;          // INVALID because pt po
age = 20;          // VALID because age is
```

Previous examples have assigned the addres
This example assigns the address of a regular v
two other possibilities: assigning the address of
assigning the address of a const to a regular p
and the second isn't:

```
const float g_earth = 9.80;
const float * pe = &g_earth;   // VALID

const float g_moon = 1.63;
float * pm = &g_moon;          // INVALID
```

For the first case, you can use neither g_ea
doesn't allow the second case for a simple reas
to pm, then you can cheat and use pm to alter t

allows `p1` to be used to alter `const` data. So t
address or pointer to a `const` pointer works
for example, if the pointer points to a fundan

Note

You can assign the address of either `const`
provided that the data type is not itself a poi
`const` data only to a non-`const` pointer.

Suppose you have an array of `const` data:

```
const int months[12] = {31,28,31,30,31,30
```

The prohibition against assigning the addr
not pass the array name as an argument to a
argument:

```
int sum(int arr[], int n);  // should hav
...
int j = sum(months, 12);    // not allowe
```

This function call attempts to assign a `con`
(`arr`), and the compiler disallows the functio

```
int sloth = 3;
const int * ps = &sloth;       // a pointer
int * const finger = &sloth;  // a const p
```

Note that the last declaration has repositio
ration constrains finger to point only to slo
alter the value of sloth. The middle declaratio
value of sloth, but it permits you to have ps
and *ps are both const, and *finger and ps

If you like, you can declare a const pointe

```
double trouble = 2.0E30;
const double * const stick = &trouble;
```

Here stick can point only to trouble, an
of trouble. In short, both stick and *stick

Typically you use the pointer-to-const for
function arguments. For example, recall the s

```
void show_array(const double ar[], int n);
```

Using const in this declaration means that
any array that is passed to it. This technique w
rection. Here, for example, the array elements
pointers or pointers-to-pointers, you wouldn'

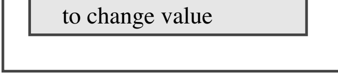


Figure 7.5 Pointers-to-c

Functions and Two-Dimen

To write a function that has a two-dimensior
ber that the name of an array is treated as its
ter is a pointer, just as for one-dimensional ar
correctly. Suppose, for example, that you start

```
int data[3][4] = {{1,2,3,4}, {9,8,7,6}, {
int total = sum(data, 3);
```

What should the prototype for `sum()` look
number of rows (3) as an argument and not a

Well, `data` is the name of an array with th
array of four `int` values. Thus, the type of `dat`
appropriate prototype would be this:

```
int sum(int (*ar2)[4], int size);
```

```
int total3 = sum(a, 10);      // sum first
int total4 = sum(a+10, 20);  // sum next 2
```

Given that the parameter `ar2` is a pointer t
definition? The simplest way is to use `ar2` as i
array. Here's a possible function definition:

```
int sum(int ar2[][4], int size)
{
    int total = 0;
    for (int r = 0; r < size; r++)
        for (int c = 0; c < 4; c++)
            total += ar2[r][c];
    return total;
}
```

Again, note that the number of rows is wh
the number of columns is fixed at four, both i
the inner `for` loop.

Here's why you can use array notation. Be
ment 0) of an array whose elements are array-
to element number `r`. Therefore `ar2[r]` is ele
array-of-four-`int`, so `ar2[r]` is the name of th
to an array name gives an array element, so `ar`
`int`, hence is a single `int` value. The pointer `a`

tions, too. For example, passing a string as an
can use const to protect a string argument fr
twists to strings that we'll unravel now.

Functions with C-Style String Arg

Suppose you want to pass a string as an argur
representing a string:

- An array of char
- A quoted string constant (also called a
- A pointer-to-char set to the address of

All three choices, however, are type pointe
you can use all three as arguments to string-p

```
char ghost[15] = "galloping";
char * str = "galumphing";
int n1 = strlen(ghost);          // ghost
int n2 = strlen(str);            // point
int n3 = strlen("gamboling");    // addre
```

Informally, you can say that you're passing
ing the address of the first character in the str
type should use type char * as the type for th

```
    unsigned int ms = c_in_str(mmm, 'm');
    unsigned int us = c_in_str(wail, 'u');
    cout << ms << " m characters in " << m
    cout << us << " u characters in " << w
    return 0;
}

// this function counts the number of ch c
// in the string str
unsigned int c_in_str(const char * str, ch
{
    unsigned int count = 0;

    while (*str)        // quit when *str
    {
        if (*str == ch)
            count++;
        str++;        // move pointer to n
    }
    return count;
}
```

acter itself. For example, immediately after th
first character in minimum. As long as the char
nonzero, so the loop continues. At the end of
the pointer by 1 byte so that it points to the
points to the terminating null character, maki
code for the null character. That condition te
functions ruthless? Because they stop at nothi

Functions That Return C-Style St

Now suppose you want to write a function t
that. But it can return the address of a string,
example, defines a function called buildstr(
two arguments: a character and a number. Us
length equals the number, and then it initializ
returns a pointer to the new string.

Listing 7.10 **strgback.cpp**

```
// strgback.cpp -- a function that return
#include <iostream>
char * buildstr(char c, int n);     // pr
int main()
{
```

```
        pstr[n] = c;        // fill rest c
    return pstr;
}
```

Here's a sample run of the program in Listi

```
Enter a character: V
Enter an integer: 46
VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
++++++++++++++++++++-DONE-+++++++++++++++
```

Program Notes

To create a string of `n` visible characters, you n
have space for the null character. So the functi
hold the string. Next, it sets the final byte to t
the array from back to front. In Listing 7.10, t
decreases to `0`, filling `n` elements:

```
while (n-- > 0)
    pstr[n] = c;
```

At the start of the final cycle, `n` has the valu
then decrement it, the `while` loop test conditi
`true`, and continues. But after making the test

Let's move from arrays to structures. It's easie
arrays. Although structure variables resemble
items, structure variables behave like basic, sir
tions. That is, unlike an array, a structure ties i
that will be treated as a unit. Recall that you
you can pass structures by value, just as you d
function works with a copy of the original st
ture. There's no funny business like the name
ment. The name of a structure is simply the r
address, you have to use the & address operato
denote the address operator. C++ additionall
variables, to be discussed in Chapter 8.)

The most direct way to program by using
the basic types—that is, pass them as argumen
However, there is one disadvantage to passing
the space and effort involved in making a cop
requirements and slow down the system. For
allow the passing of structures by value), man
of a structure and then using a pointer to acc
third alternative, called *passing by reference*, tha
other two choices now, beginning with passin

```
travel_time sum(travel_time t1, travel_tim
```

To add two times, you first add the minute number of hours to carry over, and the modul utes left. Listing 7.11 incorporates this approa `show_time()` function to display the contents

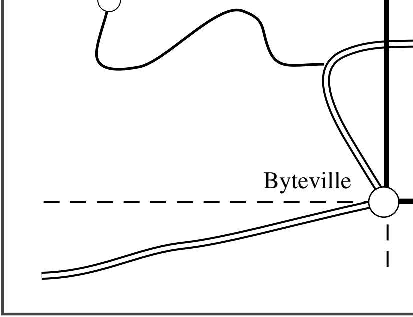
Listing 7.11 **travel.cpp**

```
// travel.cpp -- using structures with fun
#include <iostream>
struct travel_time
{
    int hours;
    int mins;
};
const int Mins_per_hr = 60;

travel_time sum(travel_time t1, travel_tim
void show_time(travel_time t);

int main()
{
    using namespace std;
```

```
void show_time(travel_time t)
{
    using namespace std;
    cout << t.hours << " hours, "
         << t.mins << " minutes\n";
}
```

Here `travel_time` acts just like a standard
ables, function return types, and function arg
and `t1` are `travel_time` structures, you can a
Note that because the `sum()` function return
an argument for the `show_time()` function. I
ments by value, the `show_time(sum(trip, d`
`sum(trip, day3)` function call in order to fi
then passes `sum()`'s return value, not the func
of the program in Listing 7.11:

```
Two-day total: 10 hours, 40 minutes
Three-day total: 15 hours, 12 minutes
```

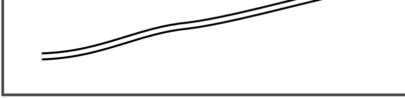


rectangular coordinates of Mi

Figure 7.6 Recta

```
struct rect
{
     double x;          // horizontal di
     double y;          // vertical dist
};
```

A second way to describe the position of a
origin and in what direction it is (for example
mathematicians have measured the angle coun



polar coordinates of Micro

Figure 7.7 Po

Let's construct a function that displays the
functions in the C++ library (borrowed from
need to measure angles in that unit. But for d
measure to degrees. This means multiplying b
57.29577951. Here's the function:

```
// show polar coordinates, converting ang
void show_polar (polar dapos)
{
    using namespace std;
    const double Rad_to_deg = 57.29577951

    cout << "distance = " << dapos.distan
    cout << ", angle = " << dapos.angle *
    cout << " degrees\n";
}
```

```
polar rect_to_polar(rect xypos)   // type
{
    polar answer;

    answer.distance =
        sqrt( xypos.x * xypos.x + xypos.y
    answer.angle = atan2(xypos.y, xypos.x)
    return answer;      // returns a polar
}
```

Now that the functions are ready, writing t
Listing 7.12 presents the result.

Listing 7.12 **strctfun.cpp**

```
// strctfun.cpp -- functions with a struct
#include <iostream>
#include <cmath>

// structure declarations
struct polar
{
    double distance;      // distance from
    double angle;         // direction fro
};
struct rect
```

```
    return 0;
}

// convert rectangular to polar coordinat
polar rect_to_polar(rect xypos)
{
    using namespace std;
    polar answer;

    answer.distance =
        sqrt( xypos.x * xypos.x + xypos.y
    answer.angle = atan2(xypos.y, xypos.x
    return answer;      // returns a pola
}

// show polar coordinates, converting ang
void show_polar (polar dapos)
{
    using namespace std;
    const double Rad_to_deg = 57.29577951

    cout << "distance = " << dapos.distan
    cout << ", angle = " << dapos.angle *
    cout << " degrees\n";
}
```

What really happens when you use cin >> r
that returns a type istream value. If you apply
rplace.x object (as in cin >> rplace.x >>
istream class. Thus, the entire while loop tes
which, as you may recall, when used in the co
bool value of true or false, depending on w
Listing 7.12, for example, cin expects the user
enters q, as shown in the sample output, cin
leaves the q in the input queue and returns a
the loop.

Compare that approach for reading numbe

```
for (int i = 0; i < limit; i++)
{
    cout << "Enter value #" << (i + 1) <<
    cin >> temp;
    if (temp < 0)
        break;
    ar[i] = temp;
}
```

To terminate this loop early, you enter a ne
negative values. This restriction fits the needs

After these changes are made, the function

```
// show polar coordinates, converting ang
void show_polar (const polar * pda)
{
    using namespace std;
    const double Rad_to_deg = 57.29577951

    cout << "distance = " << pda->distanc
    cout << ", angle = " << pda->angle *
    cout << " degrees\n";
}
```

Next, let's alter `rect_to_polar`. This is mo `rect_to_polar` function returns a structure. you should use a pointer instead of a return v ers to the function. The first points to the stru to the structure that's to hold the conversion. function *modifies* an existing structure in the argument is `const` pointer, the second is not ciples used to convert `show_polar()` to poin reworked program.

```
{
    using namespace std;
    rect rplace;
    polar pplace;

    cout << "Enter the x and y values: ";
    while (cin >> rplace.x >> rplace.y)
    {
        rect_to_polar(&rplace, &pplace);
        show_polar(&pplace);         // pas
        cout << "Next two numbers (q to qu
    }
    cout << "Done.\n";
    return 0;
}

// show polar coordinates, converting angl
void show_polar (const polar * pda)
{
    using namespace std;
    const double Rad_to_deg = 57.29577951;

    cout << "distance = " << pda->distance
    cout << ", angle = " << pda->angle * R
```

Functions and `string` C

Although C-style strings and `string` class ob
class object is more closely related to a struct
assign a structure to another structure and an
structure as a complete entity to a function, a
entity. If you need several strings, you can dec
objects instead of a two-dimensional array of

Listing 7.14 provides a short example that
the array to a function that displays the conte

Listing 7.14 **topfive.cpp**

```
// topfive.cpp -- handling an array of st
#include <iostream>
#include <string>
using namespace std;
const int SIZE = 5;
void display(const string sa[], int n);
int main()
{
    string list[SIZE];     // an array ho
    cout << "Enter your " << SIZE << " fa
```

```
3: Saturn
4: Jupiter
5: Moon
Your list:
1: Orion Nebula
2: M13
3: Saturn
4: Jupiter
5: Moon
```

The main point to note in this example is
program treats `string` just as it would treat ar
want an array of `string`, you just use the usua

```
string list[SIZE];     // an array holding
```

Each element of the `list` array, then, is a s

```
getline(cin,list[i]);
```

Similarly, the formal argument `sa` is a poin
object and can be used accordingly:

```
cout << i + 1 << ": " << sa[i] << endl;
```

How can we declare these two functions?
so that's what must appear in the prototypes:

```
void show(std::array<double, 4> da);   //
void fill(std::array<double, 4> * pa); //
```

These considerations form the core of the
more features. First, it replaces `4` with a symb

```
const int Seasons = 4;
```

Second, it adds a `const array` object con
four seasons:

```
const std::array<std::string, Seasons> Sn
    {"Spring", "Summer", "Fall", "Winter"
```

Note that the `array` template is not limit
class types too. Listing 7.15 presents the prog

Listing 7.15 **arrobj.cpp**

```
//arrobj.cpp -- functions with array obje
#include <iostream>
#include <array>
#include <string>
// constant data
```

```
        cin >> (*pa)[i];
    }
}

void show(std::array<double, Seasons> da)
{
    using namespace std;
    double total = 0.0;
    cout << "\nEXPENSES\n";
    for (int i = 0; i < Seasons; i++)
    {
        cout << Snames[i] << ": $" << da[i
        total += da[i];
    }
    cout << "Total Expenses: $" << total <
}
```

Here's a sample run:

```
Enter Spring expenses: 212
Enter Summer expenses: 256
Enter Fall expenses: 208
Enter Winter expenses: 244
```

programming look more complicated.

```
fill(&expenses);    // don't forget the &
...
cin >> (*pa)[i];
```

In the last statement, `pa` is a pointer to an
object, and `(*pa)[i]` is an element in the ob
operator precedence. The logic is straightforw
making errors.

Using references, as discussed in Chapter 8
notational problems.

Recursion

And now for something completely different
teristic that it can call itself. (Unlike C, howe
ability is termed *recursion*. Recursion is an im
such as artificial intelligence, but we'll just tak
how it works.

calls, first the *statements1* section is executed
tions were called, and then the *statements2* s
order from the order in which the functions v
recursion, the program then has to back out th
illustrates this behavior.

Listing 7.16 **recur.cpp**

```
// recur.cpp -- using recursion
#include <iostream>
void countdown(int n);

int main()
{
    countdown(4);           // call the re
    return 0;
}

void countdown(int n)
{
    using namespace std;
    cout << "Counting down ... " << n << e
    if (n > 0)
```

```
cout << n << ": Kaboom!"; << "         (n
```

Doing so produces output like the followi

```
Counting down ... 4 (n at 0012FE0C)
Counting down ... 3 (n at 0012FD34)
Counting down ... 2 (n at 0012FC5C)
Counting down ... 1 (n at 0012FB84)
Counting down ... 0 (n at 0012FAAC)
0: Kaboom!          (n at 0012FAAC)
1: Kaboom!          (n at 0012FB84)
2: Kaboom!          (n at 0012FC5C)
3: Kaboom!          (n at 0012FD34)
4: Kaboom!          (n at 0012FE0C)
```

Note how the `n` having the value `4` is stor
0012FE0C in this example), the `n` having the
ory address 0012FD34), and so on. Also note
during the "Counting down" stage is the sam
"Kaboom!" stage.

Recursion with Multiple Recursiv

Recursion is particularly useful for situations
into two smaller, similar tasks. For example, c

```
        ruler[i] = ' ';
    ruler[Len - 1] = '\0';
    int max = Len - 2;
    int min = 0;
    ruler[min] = ruler[max] = '|';
    std::cout << ruler << std::endl;
    for (i = 1; i <= Divs; i++)
    {
        subdivide(ruler,min,max, i);
        std::cout << ruler << std::endl;
        for (int j = 1; j < Len - 2; j++)
            ruler[j] = ' ';  // reset to b
    }

    return 0;
}

void subdivide(char ar[], int low, int hig
{
    if (level == 0)
        return;
    int mid = (high + low) / 2;
```

ically. That is, one call generates two, which
so on. That's why the level 6 call is able to fill
doubling of the number of function calls (an
make this form of recursion a poor choice if
is an elegant and simple choice if the necessa

Pointers to Functions

No discussion of C or C++ functions would
functions. We'll take a quick look at this topi
ities to more advanced texts.

Functions, like data items, have addresses.
which the stored machine language code for
important nor useful for you or the user to k
program. For example, it's possible to write a
function as an argument. That enables the firs
run it. This approach is more awkward than s
ond one directly, but it leaves open the possib
different times. That means the first function

The `process()` call enables the `process()` from within `process()`. The `thought()` call f passes the return value of `think()` to the `thou`

Declaring a Pointer to a Function

To declare pointers to a data type, the declarat the pointer points. Similarly, a pointer to a fun tion the pointer points. This means the declara type and the function's signature (its argumen the same information about a function that a pose Pam LeCoder has written a time-estimat

```
double pam(int);  // prototype
```

Here's what a declaration of an appropriate

```
double (*pf)(int);   // pf points to a fun
                     // one int argument a
                     // returns type doubl
```

Tip

In general, to declare a pointer to a particular for a regular function of the desired kind and sion in the form `(*pf)`. In this case, `pf` is a p such as the `pam()` function. It could have the

```
void estimate(int lines, double (*pf)(int
```

This declaration says the second argument argument and a `double` return value. To have pass `pam()`'s address to it:

```
estimate(50, pam); // function call telli
```

Clearly, the tricky part about using pointe whereas passing the address is very simple.

Using a Pointer to Invoke a Function

Now we get to the final part of the techniqu pointed-to function. The clue comes in the p the same role as a function name. Thus, all yo tion name:

```
double pam(int);
double (*pf)(int);
pf = pam;           // pf now points to
double x = pam(4);  // call pam() using
double y = (*pf)(5); // call pam() using
```

design facilitates future program development
for estimating time, he doesn't have to rewrite
supply his own `ralph()` function, making sur
type. Of course, rewriting `estimate()` isn't a
to more complex code. Also the function poi
behavior of `estimate()`, even if he doesn't ha
`estimate()`.

Listing 7.18 **fun_ptr.cpp**

```
// fun_ptr.cpp -- pointers to functions
#include <iostream>
double betsy(int);
double pam(int);

// second argument is pointer to a type do
// takes a type int argument
void estimate(int lines, double (*pf)(int)

int main()
{
    using namespace std;
    int code;
```

```
    cout << (*pf)(lines) << " hour(s)\n";
}
```

Here is a sample run of the program in Li

```
How many lines of code do you need? 30
Here's Betsy's estimate:
30 lines will take 1.5 hour(s)
Here's Pam's estimate:
30 lines will take 1.26 hour(s)
```

Here is a second sample run of the progra

```
How many lines of code do you need? 100
Here's Betsy's estimate:
100 lines will take 5 hour(s)
Here's Pam's estimate:
100 lines will take 7 hour(s)
```

Variations on the Theme of Funct

With function pointers, the notation can get i
illustrates some of the challenges of function p
begin, here are prototypes for some functions

Now consider the following statements:

```
cout <<  (*p1)(av,3) << ": " << *(*p1)(av,
cout << p2(av,3) << ": " << *p2(av,3) << e
```

Both `(*p1)(av,3)` and `p2(av,3)`, recall, re (`f1()` and `f2()`, in this case) with `av` and `3` as the return values of these two functions. The r is, address of `double` values). So the first part c address of a `double` value. To see the actual va the `*` operator to these addresses, and that's wh `*p2(av,3)` do.

With three functions to work with, it coul pointers. Then one can use a `for` loop to call would that look like? Clearly, it should look s pointer, but there should be a `[3]` somewhere question is where. And here's the answer (incl

```
const double * (*pa[3])(const double *, in
```

Why put the `[3]` there? Well, `pa` is an array declaring an array of three things is this: `pa[3` kind of thing is to be placed in the array. Ope `*pa[3]` says `pa` is an array of three pointers. T each pointer points to: a function with a signa because the result can be initialed with a sing

```
auto pc = &pa;  // C++11 automatic type d
```

What if you prefer to do it yourself? Clea
ration for pa, but because there is one more l
stuck somewhere. In particular, if we call the
pointer, not an array name. This suggests the
The parentheses bind the pd identifier to the

```
*pd[3]    // an array of 3 pointers
(*pd)[3]  // a pointer to an array of 3 e
```

In other words, pd is a pointer, and it poin
things are is described by the rest of the orig
the following:

```
const double *(*(*pd)[3])(const double *,
```

To call a function, realize that if pd points
(*pd)[i] is an array element, which is a poi
for the function call is (*pd)[i](av,3), and
the returned pointer points to. Alternatively,
function with a pointer and use (*(*pd)[i]
(av,3) for the pointed-to double value.

```
const double * f3(const double *, int);

int main()
{
    using namespace std;
    double av[3] = {1112.3, 1542.6, 2227.9

    // pointer to a function
    const double *(*p1)(const double *, in
    auto p2 = f2;  // C++11 automatic type
    // pre-C++11 can use the following cod
    // const double *(*p2)(const double *,
    cout << "Using pointers to functions:\
    cout << " Address  Value\n";
    cout <<  (*p1)(av,3) << ": " << *(*p1)
    cout << p2(av,3) << ": " << *p2(av,3)

    // pa an array of pointers
    // auto doesn't work with list initial
    const double *(*pa[3])(const double *,
    // but it does work for initializing t
    // pb a pointer to first element of pa
    auto pb = pa;
    // pre-C++11 can use the following cod
```

```
    // alternative notation
    cout << (*(*pd)[2])(av,3) << ": " <<
    // cin.get();
    return 0;
}

// some rather dull functions

const double * f1(const double * ar, int
{
    return ar;
}
const double * f2(const double ar[], int
{
    return ar+1;
}
const double * f3(const double ar[], int
{
    return ar+2;
}
```

```
002AF9F0: 2227.9
```

The addresses shown are the locations of th

This example may seem esoteric, but point
unheard of. Indeed, the usual implementation
"Class Inheritance") uses this technique. Fortu

Appreciating `auto`

One of the goals of C++11 is to make C++ ea
trate more on design and less on details. List

```
auto pc = &pa;
const double *(*(*pd)[3])(const double
```

The automatic type deduction feature reflects
piler. In C++98, the compiler uses its knowled
at least with this feature, it uses its knowledg

There is a potential drawback. Automatic type
able matches the type of the initializer, but it
wrong type of initializer:

```
auto pc = *pa;   // oops! used *pa inst
```

This declaration would make `pc` match the typ
time error when Listing 7.19 later uses `pc`, as

Functions are the C++ programming modu
definition and a prototype, and you have to u
the code that implements what the function
function interface: how many and what kind
sort of return type, if any, to get from it. The
function arguments to the function and to tr
code.

By default, C++ functions pass argument
eters in the function definition are new varia
by the function call. Thus, C++ functions pr
working with copies.

C++ treats an array name argument as th
Technically, this is still passing by value becau
address, but the function uses the pointer to a
When you declare formal parameters for a fu
declarations are equivalent:

```
typeName arr[];
typeName * arr;
```

Both of these mean that `arr` is a pointer t
code, however, you can use `arr` as if it were a
`arr[i]`. Even when passing pointers, you can
by declaring the formal argument to be a po

Chapter Review

1. What are the three steps in using a func
2. Construct function prototypes that mat
   a. `igor()` takes no arguments and h
   b. `tofu()` takes an `int` argument an
   c. `mpg()` takes two type `double` argu
   d. `summation()` takes the name of a
   returns a `long` value.
   e. `doctor()` takes a string argument
   returns a `double` value.
   f. `ofcourse()` takes a `boss` structur
   g. `plot()` takes a pointer to a `map` st

10. C++ enables you to pass a structure by
structure. If `glitz` is a structure variabl
would you pass its address? What are th

11. The function `judge()` has a type `int` r
address of a function. The function wh
to a `const char` as an argument and re

12. Suppose we have the following structu

```
struct applicant {
    char name[30];
    int credit_ratings[3];
};
```

   a. Write a function that takes an ap
   plays its contents.

   b. Write a function that takes the a
   ment and displays the contents o

13. Suppose the functions `f1()` and `f2()` l

```
void f1(applicant * a);
const char * f2(const applicant * a
```

```
{
      char maker[40];
      float height;
      float width;
      float length;
      float volume;
};
```

a. Write a function that passes a `box`
   value of each member.

b. Write a function that passes the a
   `volume` member to the product o

c. Write a simple program that uses

4. Many state lotteries use a variation of th
these variations you choose several num
numbers. For example, you might select
also pick a single number (called a mega
ond range, such as 1–27. To win the gra
rectly. The chance of winning is the pro
field numbers times the probability of p

show the array.

7. Redo Listing 7.7, modifying the three
pointer parameters to represent a range
returning the actual number of items r
after the last location filled; the other f
argument to identify the end of the da

8. Redo Listing 7.15 without using the a

    a. Use an ordinary array of `const`
    son names, and use an ordinary a

    b. Use an ordinary array of `const`
    son names, and use a structure w
    `double` for the expenses. (This d
    `array` class.)

9. This exercise provides practice in writi
tures. The following is a program skelet
functions:

```
#include <iostream>
using namespace std;
```

```
// display3() takes the address of t
// of student structures and the num
// arguments and displays the conten
void display3(const student pa[], in

int main()
{
     cout << "Enter class size: ";
     int class_size;
     cin >> class_size;
     while (cin.get() != '\n')
         continue;

    student * ptr_stu = new student[
    int entered = getinfo(ptr_stu, c
    for (int i = 0; i < entered; i++
    {
        display1(ptr_stu[i]);
        display2(&ptr_stu[i]);
    }
    display3(ptr_stu, entered);
    delete [] ptr_stu;
    cout << "Done\n";
    return 0;
}
```

by using these pointers. Hint: Here's ho

```
double (*pf[3])(double, double);
```

You can initialize such an array by usin
function names as addresses.

▪ Function templates

▪ Function template specializations

With Chapter 7, "Functions: C++'s Progr
know a lot about C++ functions, but there's
new function features that separate C++ fro
inline functions, by-reference variable passing
ing (polymorphism), and template functions.
read so far, explores features found in C++ b
into plus-plussedness.

C++ Inline Functions

Inline functions are a C++ enhancement desig
tinction between normal functions and inlin
in how the C++ compiler incorporates them
tion between inline functions and normal fu
program's innards than we have so far. Let's d

The final product of the compilation proc
of a set of machine language instructions. Wh
loads these instructions into the computer's r
ular memory address. The computer then go
Sometimes, as when you have a loop or a bra

the non-inline call. On the other hand, you a
quick process, so the absolute time savings ma
called frequently.

To use this feature, you must take at least o

- Preface the function declaration with th
- Preface the function definition with the

A common practice is to omit the prototy
(meaning the function header and all the fun
normally go.

The compiler does not have to honor you
decide the function is too large or notice that
indeed possible for inline functions), or the fe
mented for your particular compiler.

---

[1] It's a bit like having to leave off reading some text
finishing the footnote, returning to where you were

```
      cout << "hubba! ";
   cout << "\n";
}
```

A regular function transfers program
execution to a separate function.

Figure 8.1 Inline functio

Listing 8.1 illustrates the inline technique
squares its argument. Note that the entire de
but if the definition doesn't fit on one or two
identifiers), the function is probably a poor c

Listing 8.1 **inline.cpp**

```
// inline.cpp -- using an inline function
#include <iostream>

// an inline function definition
inline double square(double x) { return x

int main()
{
```

Even though the program doesn't provide
features are still in play. That's because the ent
tion's first use, serves as a prototype. This mea
ment or a `long` argument, and the program a
`double` before passing it to the function.

Inline Versus Macros

The `inline` facility is an addition to C++. C u
provide *macros*, which are crude implementat
macro for squaring a number:

```
#define SQUARE(X) X*X
```

This works not by passing arguments but thro
symbolic label for the "argument":

```
a = SQUARE(5.0); is replaced by a = 5.0
b = SQUARE(4.5 + 7.5); is replaced by b
d = SQUARE(c++); is replaced by d = c++
```

Only the first example here works properly. Yo
tion of parentheses:

```
#define SQUARE(X) ((X)*(X))
```

Creating a Reference Variable

You might recall that C and C++ use the & s
C++ assigns an additional meaning to the & s
ing references. For example, to make rodent
you could do the following:

```
int rats;
int & rodents = rats;    // makes rodents
```

In this context, & is not the address operat
fier. Just as char * in a declaration means poi
int. The reference declaration allows you to
refer to the same value and the same memory
this claim.

Listing 8.2 **firstref.cpp**

```
// firstref.cpp -- defining and using a r
#include <iostream>
int main()
{
    using namespace std;
    int rats = 101;
    int & rodents = rats;   // rodents is
```

```
rats = 101, rodents = 101
rats = 102, rodents = 102
rats address = 0x0065fd48, rodents address
```

As you can see, both `rats` and `rodents` ha
(The address values and display format vary fr
`rodents` by one affects both variables. More p
ments a single variable for which there are tw
this example shows you how a reference work
reference, which is as a function parameter, pa
ments. We'll look into these uses pretty soon.)

References tend to be a bit confusing at fi
they are tantalizingly reminiscent of pointers,
create both a reference and a pointer to refer

```
int rats = 101;
int & rodents = rats;   // rodents a refer
int * prats = &rats;    // prats a pointer
```

Then you could use the expressions `roden`
and use the expressions `&rodents` and `prats`
standpoint, a reference looks a lot like a point
dereferencing operator is understood implicitl
reference is. But there are differences besides t

Listing 8.3 **secref.cpp**

```
// secref.cpp -- defining and using a reference
#include <iostream>
int main()
{
    using namespace std;
    int rats = 101;
    int & rodents = rats;   // rodents is a referen

    cout << "rats = " << rats;
    cout << ", rodents = " << rodents << endl;

    cout << "rats address = " << &rats;
    cout << ", rodents address = " << &rodents << e

    int bunnies = 50;
    rodents = bunnies;       // can we change the r
    cout << "bunnies = " << bunnies;
    cout << ", rats = " << rats;
    cout << ", rodents = " << rodents << endl;

    cout << "bunnies address = " << &bunnies;
    cout << ", rodents address = " << &rodents << e
    return 0;
}
```

```
int & rodents = *pt;
int bunnies = 50;
pt = &bunnies;
```

Initializing `rodents` to `*pt` makes `rodents`
point to `bunnies` does not alter the fact that

References as Function Paramete

Most often, references are used as function pa
tion an alias for a variable in the calling progr
called *passing by reference*. Passing by reference
the calling function. C++'s addition of the fe
value. Passing by value, recall, results in the ca
from the calling program (see Figure 8.2). Of
value limitation by using pointers.

Let's compare using references and using p
swapping the values of two variables. A swapp
variables in the calling program. That means t
value won't work because the function will e
original variables instead of the variables them
function can work with the original data. Alte
access the original data. Listing 8.4 shows all t
work, so that you can compare them.

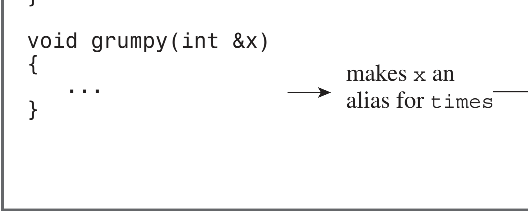


Figure 8.2 Passing by value and pas

Listing 8.4 **swaps.cpp**

```
// swaps.cpp -- swapping with references and with p
#include <iostream>
void swapr(int & a, int & b);   // a, b are aliases
void swapp(int * p, int * q);   // p, q are address
void swapv(int a, int b);       // a, b are new var
int main()
{
    using namespace std;
    int wallet1 = 300;
    int wallet2 = 350;
```

```
    temp = a;       // use a, b for values
    a = b;
    b = temp;
}

void swapp(int * p, int * q)    // use poi
{
    int temp;

    temp = *p;      // use *p, *q for valu
    *p = *q;
    *q = temp;
}

void swapv(int a, int b)        // try usi
{
    int temp;

    temp = a;      // use a, b for values
    a = b;
    b = temp;
}
```

(`(swapp(&wallet1, &wallet2)`). (Recall tha
a pointer to an `int` and therefore the argume
such as `&wallet1`.)

Next, compare the code for the functions
(passing by value). The only outward differen
parameters are declared:

```
void swapr(int & a, int & b)
void swapv(int a, int b)
```

The internal difference, of course, is that i
aliases for `wallet1` and `wallet2`, so swapping
`swapv()`, the variables `a` and `b` are new variab
`wallet2`, so swapping `a` and `b` has no effect o

Finally, compare the functions `swapr()` (p
pointer). The first difference is in how the fu

```
void swapr(int & a, int & b)
void swapp(int * p, int * q)
```

The second difference is that the pointer v
operator throughout when the function uses

Earlier, I said you should initialize a refere
call initializes its parameters with argument v
function arguments are initialized to the argu

```
    cout << cube(x);
    cout << " = cube of " << x << endl;
    cout << refcube(x);
    cout << " = cube of " << x << endl;
    return 0;
}

double cube(double a)
{
    a *= a * a;
    return a;
}

double refcube(double &ra)
{
    ra *= ra * ra;
    return ra;
}
```

Here is the output of the program in Listin

```
27 = cube of 3
27 = cube of 27
```

```
double yo[3] = { 2.2, 3.3, 4.4};
z = cube (yo[2]);            // pass the
```

Suppose you try similar arguments for a fu
seem that passing a reference should be more
name for a variable, then the actual argument
following doesn't appear to make sense becau

```
double z = refcube(x + 3.0);  // should n
```

For example, you can't assign a value to su

```
x + 3.0 = 5.0;  // nonsensical
```

What happens if you try a function call lik
C++, that's an error, and most compilers will
warning along the following lines:

```
Warning: Temporary used for parameter 'ra
```

The reason for this milder response is that
expressions to a reference variable. In some ca
`x + 3.0` is not a type `double` variable, the pr
initializing it to the value of the expression `x`
temporary variable. Let's take a closer look at
and are not created.

Now, to return to our example, suppose yo
stant reference argument:

```
double refcube(const double &ra)
{
    return ra * ra * ra;
}
```

Next, consider the following code:

```
double side = 3.0;
double * pd = &side;
double & rd = side;
long edge = 5L;
double lens[4] = { 2.0, 5.0, 10.0, 12.0};
double c1 = refcube(side);          // ra
double c2 = refcube(lens[2]);       // ra
double c3 = refcube(rd);            // ra
double c4 = refcube(*pd);           // ra
double c5 = refcube(edge);          // ra
double c6 = refcube(7.0);           // ra
double c7 = refcube(side + 10.0);   // ra
```

The arguments `side`, `lens[2]`, `rd`, and `*pd`
it is possible to generate a reference for them,
(Recall that an element of an array behaves lik

leaving `a` and `b` unaltered.

In short, if the intent of a function with re
passed as arguments, situations that create tem
solution is to prohibit creating temporary var
C++ Standard now does. (However, some co
of error messages, so if you see a warning abo

Now think about the `refcube()` function
to modify them, so temporary variables cause
eral in the sorts of arguments it can handle. T
ence is `const`, C++ generates temporary vari
function with a `const` reference formal argum
mimics the traditional passing by value behav
unaltered and using a temporary variable to h

> **Note**
>
> If a function call argument isn't an `lvalue` o
> `const` reference parameter, C++ creates an
> the value of the function call argument to the
> refer to that variable.

semantics. The original reference type (the on
lvalue reference.

Using References with a Structur

References work wonderfully with structures
Indeed, references were introduced primarily
basic built-in types.

The method for using a reference to a stru
the method for using a reference to a basic va
when declaring a structure parameter. For exa
tion of a structure:

```
struct free_throws
{
    std::string name;
    int made;
    int attempts;
    float percent;
};
```

Then a function using a reference to this t

```
void set_pc(free_throws & ft);   // use a
```

If the intent is that the function doesn't alt

```
void display(const free_throws & ft);  //
```

```
// partial initializations - remaining me
    free_throws one = {"Ifelsa Branch", 1
    free_throws two = {"Andor Knott", 10,
    free_throws three = {"Minnie Max", 7,
    free_throws four = {"Whily Looper", 5
    free_throws five = {"Long Long", 6, 1
    free_throws team = {"Throwgoods", 0,
// no initialization
    free_throws dup;

    set_pc(one);
    display(one);
    accumulate(team, one);
    display(team);
// use return value as argument
    display(accumulate(team, two));
    accumulate(accumulate(team, three), f
    display(team);
// use return value in assignment
    dup = accumulate(team,five);
    std::cout << "Displaying team:\n";
    display(team);
    std::cout << "Displaying dup after as
    display(dup);
```

```
free_throws & accumulate(free_throws & tar
{
    target.attempts += source.attempts;
    target.made += source.made;
    set_pc(target);
    return target;
}
```

Here is the program output:

```
Name: Ifelsa Branch
  Made: 13      Attempts: 14    Percent: 9
Name: Throwgoods
  Made: 13      Attempts: 14    Percent: 9
Name: Throwgoods
  Made: 23      Attempts: 30    Percent: 7
Name: Throwgoods
  Made: 35      Attempts: 48    Percent: 7
Displaying team:
Name: Throwgoods
  Made: 41      Attempts: 62    Percent: 6
Displaying dup after assignment:
Name: Throwgoods
```

```
    if (pt->attempts != 0)
        pt->percent = 100.0f *float(pt->m
    else
        pt->percent = 0;
}
```

The next function call is this:

```
display(one);
```

Because `display()` displays the contents
function uses a `const` reference parameter. In
ture by value, but using a reference is more e
a copy of the original structure.

The next function call is this:

```
accumulate(team, one);
```

The `accumulate()` function takes two str
`attempts` and `made` members of the second s
the first structure. Only the first structure is r
whereas the second parameter is a `const` refe

```
free_throws & accumulate(free_throws & ta
```

Therefore, the contents of team get displayed.

```
display(accumulate(team, two));
```

is the same as that of the following:

```
accumulate(team, two);
display(team);
```

The same logic applies to this statement:

```
accumulate(accumulate(team, three), four);
```

This has the same effect as the following:

```
accumulate(team, three);
accumulate(team, four);
```

Next, the program uses an assignment state

```
dup = accumulate(team,five);
```

As you might expect, this copies the values
Finally, the program uses accumulate() in

```
accumulate(dup,five) = four;
```

This statement—that is, assigning a value to
value is a reference. The code won't compile i
the return value is a reference to dup, this cod

If accumulate() returned a structure inst
involve copying the entire structure to a tem
to dup. But with a reference return value, te
approach.

Note

A function that returns a reference is actuall

Being Careful About What a Return Ref

The single most important point to rememb
returning a reference to a memory location t
nates. What you want to avoid is code along

```
const free_throws & clone2(free_throws &
{
    free_throws newguy;  // first step to
    newguy = ft;         // copy info
    return newguy;       // return refere
}
```

This has the unfortunate effect of returnin
(newguy) that passes from existence as soon a
ory Models and Namespaces," discusses the p
ilarly, you should avoid returning pointers to

no longer needed. A call to clone() conceals
to use delete later. The auto_ptr template o
in Chapter 16, "The string Class and the Sta
the deletion process.

Why Use `const` with a Reference Retur

Listing 8.6, as you'll recall, had this statement:

```
accumulate(dup,five) = four;
```

It had the effect of first adding data from f
dup with the contents of four. Why does this
modifiable lvalue on the left. That is, the subex
expression should identify a block of memory
tion returned a reference to dup, which does i
statement is valid.

Regular (non reference) return types, on th
accessed by address. Such expressions can appe
ment but not the left. Other examples of rvalu
sions such as x + y. Clearly, it doesn't make se
such as 10.0, but why is a normal function re
value, you'll recall, resides in a temporary men
even until the next statement.

And of course you still could use `accumul`
statement.

By omitting `const`, you can write shorter

Usually, you're better off avoiding the add
obscure features often expand the opportunit
type a `const` reference therefore protects you
sionally, however, omitting `const` does make s
in Chapter 11, "Working with Classes," is an

Using References with a Class O

The usual C++ practice for passing class obje
instance, you would use reference parameters
`ostream`, `istream`, `ofstream`, and `ifstream` c

Let's look at an example that uses the `str`
choices, some of them bad. The general idea
to each end of another string. Listing 8.7 pro
this. However, one of the designs is so flawed
even not compile.

```
    result = version2(input, "###");
    cout << "Your string enhanced: " << re
    cout << "Your original string: " << in

    cout << "Resetting original string.\n"
    input = copy;
    result = version3(input, "@@@");
    cout << "Your string enhanced: " << re
    cout << "Your original string: " << in

    return 0;
}

string version1(const string & s1, const s
{
    string temp;

    temp = s2 + s1 + s2;
    return temp;
}

const string & version2(string & s1, const
{
```

At this point the program crashed.

Program Notes

Version 1 of the function in Listing 8.7 is the

```
string version1(const string & s1, const
{
    string temp;

    temp = s2 + s1 + s2;
    return temp;
}
```

It takes two `string` arguments and uses st
that has the desired properties. Note that the
ences. The function would produce the same

```
string version4(string s1, string  s2)  /
```

In this case, `s1` and `s2` would be brand-ne
more efficient because the function doesn't h
the old objects to the new. The use of the co
use, but not modify, the original strings.

The `temp` object is a new object, local to t
exist when the function terminates. Thus, ret

actual argument used in the function call can
a quoted string literal, a null-terminated array
char. Hence the following works fine:

```
result = version1(input, "***");
```

The version2() function doesn't create a
the original string:

```
const string & version2(string & s1, const
{
    s1 = s2 + s1 + s2;
// safe to return reference passed to func
    return s1;
}
```

This function is allowed to alter s1 becaus
Because s1 is a reference to an object (inp
erence. Because s1 is a reference to input, the

```
result = version2(input, "###");
```

essentially becomes equivalent to the follow

```
version2(input, "###");     // input alter
result = input;             // reference t
```

The program attempts to refer to memory

Another Object Lesson: Objects,

The ostream and ofstream classes bring an
As you may recall from Chapter 6, "Branchin
of the ofstream type can use ostream metho
same forms as console input/output. The lang
features from one class to another is called *in*
discusses this feature in detail. In brief, ostre
ofstream class is based on it) and ofstream i
from ostream). A derived class inherits the ba
ofstream object can use base class features su
ting methods.

Another aspect of inheritance is that a bas
object without requiring a type cast. The prac
function having a base class reference parame
class objects and also with derived objects. Fo
parameter can accept an ostream object, such
might declare, equally well.

Listing 8.8 demonstrates this point by usin
and to display the data onscreen; only the fun

```
    {
        cout << "Can't open " << fn << ".
        exit(EXIT_FAILURE);
    }
    double objective;
    cout << "Enter the focal length of you
            "telescope objective in mm: ";
    cin >> objective;
    double eps[LIMIT];
    cout << "Enter the focal lengths, in m
         << " eyepieces:\n";
    for (int i = 0; i < LIMIT; i++)
    {
        cout << "Eyepiece #" << i + 1 << "
        cin >> eps[i];
    }
    file_it(fout, objective, eps, LIMIT);
    file_it(cout, objective, eps, LIMIT);
    cout << "Done\n";
    return 0;
}
```

Here is a sample run of the program in Li

```
Enter the focal length of your telescope
Enter the focal lengths, in mm, of 5 eyep
Eyepiece #1: 30
Eyepiece #2: 19
Eyepiece #3: 14
Eyepiece #4: 8.8
Eyepiece #5: 7.5
Focal length of objective: 1800 mm
f.l. eyepiece  magnification
        30.0             60
        19.0             95
        14.0            129
         8.8            205
         7.5            240
Done
```

The following line writes the eyepiece da

```
file_it(fout, objective, eps, LIMIT);
```

And this line writes the identical informat

```
file_it(cout, objective, eps, LIMIT);
```

The setf() method returns a copy of all t
call was made. ios_base::fmtflags is a fancy
information. So the assignment to initial st
the file_it() function was called. The initi
ment to setf() to reset all the formatting sett
tion restores the object to the state it had befo

Knowing more about classes will help you
and, why, for example, ios_base keeps poppin
Chapter 17 to use these methods.

One final point: Each object stores its own
passes cout to file_it(), cout's settings are a
program passes fout to file_it(), fout's sett

When to Use Reference Argument

There are two main reasons for using referenc

- To allow you to alter a data object in th
- To speed up a program by passing a refe

The second reason is most important for la
objects. These two reasons are the same reason
ment. This makes sense because reference argu

- If the data object is a class object, use a

Of course, these are just guidelines, and th
choices. For example, cin uses references for
instead of cin >> &n.

Default Arguments

Let's look at another topic from C++'s bag o
argument is a value that's used automatically if
from a function call. For example, if you set u
value of 1, the function call wow() is the same
you use a function. Suppose you have a funct
characters of a string, with the string and n as
returns a pointer to a new string consisting o
For example, the call left("theory", 3) co
pointer to it. Now suppose you establish a de
call left("theory", 3) would work as befo
default. But the call left("theory"), instead
argument of 1 and return a pointer to the str
program often needs to extract a one-charact
longer strings.

```
beeps = harpo(2);       // same as harpo(2
beeps = harpo(1,8);     // same as harpo(1
beeps = harpo (8,7,6);  // no default argu
```

The actual arguments are assigned to the c right; you can't skip over arguments. Thus, the

```
beeps = harpo(3, ,8);   // invalid, doesn'
```

Default arguments aren't a major programm venience. When you begin working with class number of constructors, methods, and methoc

Listing 8.9 puts default arguments to use. N default. The function definition is the same as

Listing 8.9 **left.cpp**

```
// left.cpp -- string function with a defa
#include <iostream>
const int ArSize = 80;
char * left(const char * str, int n = 1);
int main()
{
    using namespace std;
    char sample[ArSize];
    cout << "Enter a string:\n";
```

```
}
```

Here's a sample run of the program in Lis

```
Enter a string:
forthcoming
fort
f
```

Program Notes

The program in Listing 8.9 uses `new` to creat acters. One awkward possibility is that an un ber of characters. In that case, the function se returns the null string. Another awkward pos request more characters than the string conta using a combined test:

```
i < n && str[i]
```

The `i < n` test stops the loop after `n` char the test, the expression `str[i]`, is the code fo loop reaches the null character, the code is `0`,

```
int m = 0;
while (m <= n && str[m] != '\0')
      m++;
char * p = new char[m+1]:
// use m instead of n in rest of code
```

Remember, the expression `str[m] != '\0` null character and to `false` when it is the nul verted to `true` in an `&&` expression and zero is be written this way:

```
while (m<=n && str[m])
```

Function Overloading

Function polymorphism is a neat C++ additi ments let you call the same function by using *morphism*, also called *function overloading*, lets yo name. The word *polymorphism* means having n function have many forms. Similarly, the expre attach more than one function to the same na sions boil down to the same thing, but we'll u *overloading*—it sounds harder working. You ca of functions that do essentially the same thing

```
print(1999.0, 10);          // use #2
print(1999, 12);            // use #4
print(1999L, 15);           // use #3
```

For example, `print("Pancakes", 15)` us
matches Prototype #1.

When you use overloaded functions, you
types in the function call. For example, consi

```
unsigned int year = 3210;
print(year, 6);          // ambiguous ca
```

Which prototype does the `print()` call m
lack of a matching prototype doesn't automa
because C++ will try to use standard type co
`print()` prototype were #2, the function cal
value to type `double`. But in the earlier code
ber as the first argument, providing three diff
with this ambiguous situation, C++ rejects th

Some signatures that appear to be differen
For example, consider these two prototypes:

```
double cube(double x);
double cube(double & x);
```

```
dabble(p2);       // dabble(char *);
drivel(p1);       // drivel(const char *);
drivel(p2);       // drivel(const char *);
```

The `dribble()` function has two prototyp
ular pointers—and the compiler selects one o
actual argument is `const`. The `dabble()` func
argument, but the `drivel()` function matche
ments. The reason for this difference in behav
it's valid to assign a non-`const` value to a `con`

Keep in mind that the signature, not the fu
For example, the following two declarations a

```
long gronk(int n, float m);      // same s
double gronk(int n, float m);    // hence
```

Therefore, C++ doesn't permit you to ove
different return types, but only if the signatur

```
long gronk(int n, float m);        // diff
double gronk(float n, float m);    // henc
```

After we discuss templates later in this cha
matching.

```
double x = 55.5;
const double y = 32.0;
stove(x);              // calls stove(d
stove(y);              // calls stove(c
stove(x+y);            // calls stove(d
```

If, say, you omit the `stove(double &&)` fur
`stove(const double &)` function instead

An Overloading Example

In this chapter we've already developed a `lef`
first `n` characters in a string. Let's add a secon
digits in an integer. You can use it, for exampl
postal zip code stored as an integer, which is

The integer function is a bit more difficul
you don't have the benefit of each digit being
approach is to first compute the number of d
10 lops off one digit, so you can use division
with a loop, like this:

```
unsigned digits = 1;
while (n /= 10)
    digits++;
```

from that of the old left(), you can use both

Listing 8.10 **leftover.cpp**

```
// leftover.cpp -- overloading the left()
#include <iostream>
unsigned long left(unsigned long num, unsi
char * left(const char * str, int n = 1);

int main()
{
     using namespace std;
    char * trip = "Hawaii!!";   // test va
    unsigned long n = 12345678; // test va
    int i;
    char * temp;

    for (i = 1; i < 10; i++)
    {
        cout << left(n, i) << endl;
        temp = left(trip,i);
        cout << temp << endl;
        delete [] temp; // point to tempor
    }
    return 0;
```

```
// This function returns a pointer to a n
// consisting of the first n characters i
char * left(const char * str, int n)
{
    if(n < 0)
        n = 0;
    char * p = new char[n+1];
    int i;
    for (i = 0; i < n && str[i]; i++)
        p[i] = str[i];  // copy character
    while (i <= n)
        p[i++] = '\0';  // set rest of st
    return p;
}
```

Here's the output of the program in Listin

```
1
H
12
Ha
123
Haw
1234
```

But using the single function with a defaul
tion to write instead of two, and the program
instead of two. If you decide to modify the fu
if you require different types of arguments, de
case, you should use function overloading.

What Is Name Decoration?

How does C++ keep track of which overloade
to each of these functions. When you use the
and compile programs, your C++ compiler pe
name decoration or *name mangling*—through
on the formal parameter types specified in th
undecorated function prototype:

```
long MyFunctionFoo(int, float);
```

This format is fine for us humans; we know th
`int` and `float`, and it returns a value of type
ments this interface by transforming the nam
unsightly appearance, perhaps something lik

```
?MyFunctionFoo@@YAXH
```

The apparent gibberish decorating the origina
tude) encodes the number and types of parar
result in a different set of symbols being add
conventions for their efforts at decorating.

```
double x;           // intended change o
short doubleerval;  // unintended change
```

C++'s function template capability autom
ing greater reliability.

Function templates enable you to define a
example, you can set up a swapping template

```
template <typename AnyType>
void Swap(AnyType &a, AnyType &b)
{
    AnyType temp;
    temp = a;
    a = b;
    b = temp;
}
```

The first line specifies that you are setting
arbitrary type `AnyType`. The keywords `templa`
you can use the keyword `class` instead of `ty`
ets. The type name (`AnyType`, in this example
usual C++ naming rules; many programmers
admit, is simple indeed. The rest of the code
ues of type `AnyType`. The template does not
compiler with directions about how to defin

You should use templates if you need functio
types. If you aren't concerned with backward
typing a longer word, you can use the keyword
declare type parameters.

To let the compiler know that you need a
use a function called `Swap()` in your program
you use and then generates the corresponding
works. The program layout follows the usual p
plate function prototype near the top of the fi
lowing `main()`. The example follows the more
`AnyType` as the type parameter.

Listing 8.11 **funtemp.cpp**

```
// funtemp.cpp -- using a function templat
#include <iostream>
// function template prototype
template <typename T>  // or class T
void Swap(T &a, T &b);

int main()
{
    using namespace std;
```

```
    a = b;
    b = temp;
}
```

The first `Swap()` function in Listing 8.11 I
erates an `int` version of the function. That is,
a definition that looks like this:

```
void Swap(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

You don't see this code, but the compiler
The second `Swap()` function has two `double`
`double` version. That is, it replaces `T` with `dou`

```
void Swap(double &a, double &b)
{
    double temp;
    temp = a;
```

You use templates when you need functions t
types, as in Listing 8.11. It might be, however,
rithm. To handle this possibility, you can overl
regular function definitions. As with ordinary
tinct function signatures. For example, Listing
for swapping elements of two arrays. The orig
whereas the new template has the signature (T
parameter in this case happens to be a specific
all template arguments have to be template pa

When, in `twotemps.cpp`, the compiler enc
that it has two `int` arguments and matches Sw
use, however, has two `int` arrays and an `int` v
template.

Listing 8.12 **twotemps.cpp**

```
// twotemps.cpp -- using overloaded templa
#include <iostream>
template <typename T>     // original temp
void Swap(T &a, T &b);

template <typename T>     // new template
void Swap(T *a, T *b, int n);
```

```
}

template <typename T>
void Swap(T &a, T &b)
{
    T temp;
    temp = a;
    a = b;
    b = temp;
}

template <typename T>
void Swap(T a[], T b[], int n)
{
    T temp;
    for (int i = 0; i < n; i++)
    {
        temp = a[i];
        a[i] = b[i];
        b[i] = temp;
    }
}

void Show(int a[])
{
```

Suppose you have a template function:

```
template <class T>     // or template <typ
void f(T a, T b)
{...}
```

Often the code makes assumptions about v
instance, the following statement assumes that
true if type `T` is a built-in array type:

```
a = b;
```

Similarly, the following assumes > is define
structure:

```
if (a > b)
```

Also the > operator is defined for array nar
compares the addresses of the arrays, which m
following assumes the multiplication operator
if `T` is an array, a pointer, or a structure:

```
T c = a*b;
```

In short, it's easy to write a template functi
other hand, sometimes a generalization makes
allow for it. For example, it could make sense
dinates, even though the + operator isn't defin

Because C++ allows you to assign one str
type `T` is a `job` structure. But suppose you on
bers, keeping the `name` members unchanged.
ments to `Swap()` would be the same as for th
so you can't use template overloading to supp

However, you can supply a specialized fun
specialization, with the required code. If the co
exactly matches a function call, it uses that de

The specialization mechanism has change
the current form as mandated by the C++ S

Third-Generation Specialization (ISO/AN

After some youthful experimentation with o
on this approach:

- For a given function name, you can hav
tion, and an explicit specialization temp
of all of these.
- The prototype and definition for an ex
`template <>` and should mention the
- A specialization overrides the regular te
rides both.

```
int main()
{
    double u, v;
    ...
    Swap(u,v);  // use template
    job a, b;
    ...
    Swap(a,b);  // use void Swap<job>(job
}
```

The `<job>` in `Swap<job>` is optional becau
this is a specialization for `job`. Thus, the proto

```
template <> void Swap(job &, job &);   //
```

In case you have to work with an older co
dard usage soon, but first, let's see how explici

An Example of Explicit Specialization

Listing 8.13 illustrates how explicit specializat

```
    int i = 10, j = 20;
    cout << "i, j = " << i << ", " << j <
    cout << "Using compiler-generated int
    Swap(i,j);    // generates void Swap(
    cout << "Now i, j = " << i << ", " <<

    job sue = {"Susan Yaffee", 73000.60,
    job sidney = {"Sidney Taffee", 78060.
    cout << "Before job swapping:\n";
    Show(sue);
    Show(sidney);
    Swap(sue, sidney); // uses void Swap(
    cout << "After job swapping:\n";
    Show(sue);
    Show(sidney);
    // cin.get();
    return 0;
}

template <typename T>
void Swap(T &a, T &b)    // general versi
{
    T temp;
    temp = a;
```

```
}
```

Here's the output of the program in Listing

```
i, j = 10, 20.
Using compiler-generated int swapper:
Now i, j = 20, 10.
Before job swapping:
Susan Yaffee: $73000.60 on floor 7
Sidney Taffee: $78060.72 on floor 9
After job swapping:
Susan Yaffee: $78060.72 on floor 9
Sidney Taffee: $73000.60 on floor 7
```

Instantiations and Specializations

To extend your understanding of templates, le *ialization*. Keep in mind that including a funct generate a function definition. It's merely a pl When the compiler uses the template to gene type, the result is termed an *instantiation* of the function call `Swap(i,j)` causes the compiler t `int` as the type. The template *is not* a function

explicitly defined for the int type. These pr
function definitions. The explicit specializatio
plate, whereas the explicit instantiation omits

Caution

It is an error to try to use both an explicit ins
same type(s) in the same file, or, more gener

Explicit instantiations also can be created l
instance, consider the following:

```
template <class T>
T Add(T a, T b)    // pass by value
{
    return a + b;
}
...
int m = 6;
double x = 10.2;
cout << Add<double>(x, m) << endl;  // ex
```

The template would fail to match the fun
expects both function arguments to be of the
forces the type double instantiation, and the
match the second parameter of the Add<doub

```
{
  template void Swap<char>(char &, char &)
  short a, b;
  ...
  Swap(a,b);    // implicit template insta
  job n, m;
  ...
  Swap(n, m);   // use explicit specializa
  char g, h;
  ...
  Swap(g, h);  // use explicit template in
  ...
}
```

When the compiler reaches the explicit ins nition to generate a `char` version of `Swap()`. F piler matches a template to the actual argume when the compiler reaches the function call S `Swap()` because the two arguments are type s `Swap(n,m)`, it uses the separate definition (the type. When the compiler reaches `Swap(g,h)`, generated when it processed the explicit insta

First, the compiler rounds up the suspects,
that have the name may(). Then, it finds thos
example, the following pass muster because th
with one argument:

```
void may(int);
float may(float, float = 3);
void may(char);
char * may(const char *);
char may(const char &);
template<class T> void may(const T &);
template<class T> void may(T *);
```

Note that just the signatures and not the r
didates (#4 and #7), however, are not viable
implicitly (that is, without an explicit type ca
is viable because it can be used to generate a
That leaves five viable functions, each of whi
declared.

Next, the compiler has to determine whic
the conversion required to make the function
argument. In general, the ranking from best t

so these rules include converting `char &` to `c`
entry means that a function name as an actual
formal parameter, as long as both have the san
ber function pointers from Chapter 7. Also re
as an argument to a function that expects a po
`volatile` keyword later in Chapter 9.

Table 8.1 **Trivial Conversions Allowed for an**

From an Actual Argument	To a Form
`Type`	`Type &`
`Type &`	`Type`
`Type []`	`* Type`
`Type (argument-list)`	`Type (*)`
`Type`	`const Ty`
`Type`	`volatile`
`Type *`	`const Ty`
`Type *`	`volatile`

Another case in which one exact match is
a non template function and the other isn't. I
better than a template, including explicit spec

If you wind up with two exact matches th
template function that is the more specialized
for example, that an explicit specialization is
the template pattern:

```
struct blot {int a; char b[10];};
template <class Type> void recycle (Type
template <> void recycle<blot> (blot & t)
...
blot ink = {25, "spots"};
...
recycle(ink);  // use specialization
```

The term *most specialized* doesn't necessari
erally, it indicates that fewer conversions take
to use. For example, consider the following tw

```
template <class Type> void recycle (Type
template <class Type> void recycle (Type
```

Let's examine a complete program that uses th
which template definition to use. Listing 8.14
the contents of an array. The first definition (T
passed as an argument contains the data to be
B) assumes that the array elements are pointer

Listing 8.14 **tempover.cpp**

```
// tempover.cpp -- template overloading
#include <iostream>

template <typename T>            // templa
void ShowArray(T arr[], int n);

template <typename T>            // templa
void ShowArray(T * arr[], int n);

struct debts
{
    char name[50];
    double amount;
};

int main()
```

```
template <typename T>
void ShowArray(T arr[], int n)
{
    using namespace std;
    cout << "template A\n";
    for (int i = 0; i < n; i++)
        cout << arr[i] << ' ';
    cout << endl;
}

template <typename T>
void ShowArray(T * arr[], int n)
{
    using namespace std;
    cout << "template B\n";
    for (int i = 0; i < n; i++)
        cout << *arr[i] << ' ';
    cout << endl;
}
```

*arr[i]—that is, the double values pointed t
plates, Template B is the more specialized beca
array contents are pointers, so it is the templat

Here's the output of the program in Listing

```
Listing Mr. E's counts of things:
template A
13 31 103 301 310 130
Listing Mr. E's debts:
template B
2400 1300 1800
```

If you remove Template B from the progra
listing the contents of pd, so it lists the address

In short, the overload resolution process lo
there's just one, that function is chosen. If mo
is a non template function, that non template
date are otherwise tied and all are template fu
than the rest, that one is chosen. If there are tw
functions, or if there are two or more equally
more specialized than the rest, the function ca
matching calls, of course, that is also an error.

```
}

int main()
{
    using namespace std;
    int m = 20;
    int n = -30;
    double x = 15.5;
    double y = 25.9;

    cout << lesser(m, n) << endl;       /
    cout << lesser(x, y) << endl;       /
    cout << lesser<>(m, n) << endl;     /
    cout << lesser<int>(x, y)  << endl; /

    return 0;
}
```

(The final function call converts `double` t
ings about that.)

should choose a template function rather than
noting that the actual arguments are type int,

Finally, consider this statement:

```
cout << lesser<int>(x, y)  << endl; // use
```

Here we have a request for an explicit insta
tion that gets used. The values of x and y are t
returns an int value, which is why the progra

Functions with Multiple Type Arguments

Where matters really get involved is when a fu
matched to prototypes with multiple type arg
for all the arguments. If it can find a function
tions, it is chosen. For one function to be bett
least as good a match for all arguments and a b

This book does not intend to challenge the
The rules are there so that there is a well-defi
prototypes and templates.

Template Function Evolution

In the early days of C++, most people didn't
functions and template classes would prove to

The proper type might be T1 or T2 or some
be double and T2 could be int, in which cas
be short and T2 could be int, in which case
short and T2 is char. Then addition invokes
resultant type is int. Also the + operator can
plicating the options further. Therefore, in C-
of xpy.

The decltype Keyword (C++11)

The C++11 solution is a new keyword: decl

```
int x;
decltype(x) y;   // make y the same type
```

The argument to decltype can be an exp
this code:

```
decltype(x + y) xpy;  // make xpy the sam
xpy = x + y;
```

Alternatively, we could combine these two

```
decltype(x + y) xpy = x + y;
```

```
decltype(rx) u = y;  // u is type double &
decltype(pd) v;      // v is type const do
```

Stage 2: If *expression* is a function call,
return type:

```
long indeed(int);
decltype (indeed(3)) m; // m is type int
```

> **Note**
>
> The call *expression* isn't evaluated. In this
> get the return type; there's no need to actuall

Stage 3: If *expression* is an lvalue, then
This might seem to imply that earlier exampl
types, given that w is an lvalue. However, keep
Stage 1. For this stage to apply, *expression* c
what can it be? One obvious possibility is a p

```
double xx = 4.4;
decltype ((xx)) r2 = xx;  // r2 is double
decltype(xx) w = xx;      // w is double
```

```
    xytype xpy = x + y;
    xytype arr[10];
    xytype & rxy = arr[2];   // rxy a ref
    ...
}
```

Alternative Function Syntax (C++11 Tra

The `decltype` mechanism by itself leaves and
incomplete template function:

```
template<class T1, class T2>
?type? gt(T1 x, T2 y)
{
    ...
    return x + y;
}
```

Again, we don't know in advance what ty
that we could use `decltype(x + y)` for the
the code, the parameters `x` and `y` have not ye
ible and usable to the compiler). The `declty`
are declared. To make this possible, C++11 a
functions. Here's how it works using built-in

Summary

C++ has expanded C function capabilities. B
definition and by placing that definition ahea
gest to the C++ compiler that it make the fu
program jump to a separate section of code to
the function call with the corresponding cod
only when the function is short.

A reference variable is a kind of disguised
second name) for a variable. Reference variab
tions that process structures and class objects.
ence to a particular type can refer only to dat
derived from another, such as `ofstream` from
also refer to the derived type.

C++ prototypes enable you to define defa
omits the corresponding argument, the progr
includes an argument value, the program uses
arguments can be provided only from right to
vide a default value for a particular argument,
arguments to the right of that argument.

3. Write overloaded versions of `iquote()`
enclosed in double quotation marks. W
one for a `double` argument, and one fo

4. The following is a structure template:

```
struct box
{
    char maker[40];
    float height;
    float width;
    float length;
    float volume;
};
```

   a. Write a function that has a refere
      ment and displays the value of ea

   b. Write a function that has a refere
      ment and sets the `volume` memb
      sions.

5. What changes would need be made to
and `show()` use reference parameters?

8. Given the template of Chapter Review
ter Review Question 4, provide a temp
ments and returns the one with the larg

9. What types are assigned to v1, v2, v3, v
the code is part of a complete program)

```
int g(int x);
...
float m = 5.5f;
float & rm = m;
decltype(m) v1 = m;
decltype(rm) v2 = m;
decltype((m)) v3 = m;
decltype (g(100)) v4;
decltype (2.0 * m) v5;
```

Programming Exercises

1. Write a function that normally takes or
prints that string once. However, if a sec
nonzero, the function should print the s
ber of times that function has been calle
times the string is printed is not equal t

```
Next string (q to quit): q
Bye.
```

4. The following is a program skeleton:

```
#include <iostream>
using namespace std;
#include <cstring>      // for strl
struct stringy {
    char * str;        // points to
    int ct;            // length of
    };

// prototypes for set(), show(), an
int main()
{
    stringy beany;
    char testing[] = "Reality isn't

    set(beany, testing);    // firs
                // allocates space
                // sets str member
                // new block, copie
                // and sets ct memb
```

6. Write a template function `maxn()` that
type `T` and an integer representing the r
returns the largest item in the array. Test
plate with an array of six `int` value and
gram should also include a specializatio
an argument and the number of pointe
address of the longest string. If multiple
length, the function should return the a
the specialization with an array of five s

7. Modify Listing 8.14 so that it uses two
return the sum of the array contents ins
gram now should report the total numb

C++ offers many choices for storing data
data remains in memory (storage duration) a
access to data (scope and linkage). You can al
and placement `new` offers a variation on that
vides additional control over access. Larger p
code files that may share some data in comm
pilation of the program files, and this chapter

Separate Compilation

C++, like C, allows and even encourages you
program in separate files. As Chapter 1, "Gett
compile the files separately and then link the
C++ compiler typically compiles programs a
modify just one file, you can recompile just t
compiled versions of the other files. This faci
Furthermore, most C++ environments provi
agement. Unix and Linux systems, for examp
which files a program depends on and when
and it detects that you've changed one or mo
`make` remembers the proper steps needed to
development environments (IDEs), including

that use those structures

- A source code file that contains the cod
- A source code file that contains the cod

This is a useful strategy for organizing a pr program that uses those same functions, you c functions file to the project or `make` list. Also t approach. One file, the header file, contains th second file contains the function code for ma they form a package you can use for a variety

You shouldn't put function definitions or v might work for a simple setup but usually it le function definition in a header file and then i that are part of a single program, you'd wind u in a single program, which is an error, unless t commonly found in header files:

- Function prototypes
- Symbolic constants defined using `#def`
- Structure declarations
- Class declarations

compiler (bcc32.exe) also behave that way. A
and Microsoft Visual C++ go through essent
Chapter 1, you initiate the process differently
and associate source code files with it. Note
header files, to projects. That's because the #i
Also you shouldn't use #include to include
multiple declarations.

> **Caution**
>
> In IDEs, don't add header files to the project
> code files in other source code files.

Listing 9.1 **coordin.h**

```
// coordin.h -- structure templates and f
// structure templates
#ifndef COORDIN_H_
#define COORDIN_H_

struct polar
{
    double distance;    // distance from
    double angle;        // direction fro
};
```

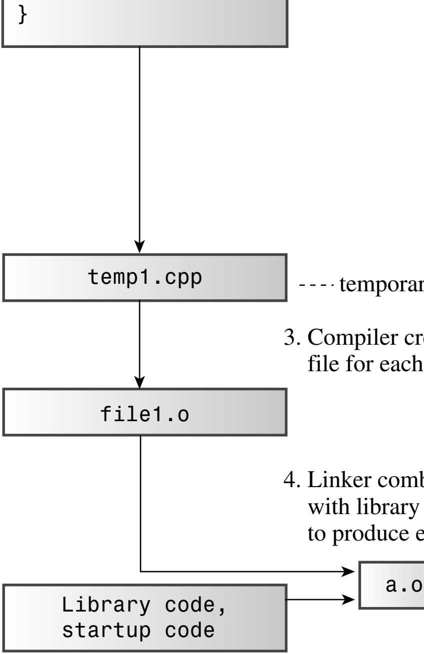


Figure 9.1 Compiling a multifile

```
#define COORDIN_H_
// place include file contents here
#endif
```

The first time the compiler encounters the fi
(I chose a name based on the `include` filen
in to create a name that is unlikely to be def
piler looks at the material between the `#ifn`
In the process of looking at the material, the
If it then encounters a second inclusion of `c`
that `COORDIN_H_` is defined and skips to the
method doesn't keep the compiler from inclu
ignore the contents of all but the first inclus
files use this guarding scheme. Otherwise yo
one file, and that will produce a compile erro

Listing 9.2 **file1.cpp**

```
// file1.cpp -- example of a three-file p
#include <iostream>
#include "coordin.h" // structure templat
using namespace std;
int main()
```

```
// convert rectangular to polar coordinate
polar rect_to_polar(rect xypos)
{
    using namespace std;
    polar answer;

    answer.distance =
        sqrt( xypos.x * xypos.x + xypos.y
    answer.angle = atan2(xypos.y, xypos.x)
    return answer;      // returns a polar
}

// show polar coordinates, converting angl
void show_polar (polar dapos)
{
    using namespace std;
    const double Rad_to_deg = 57.29577951;

    cout << "distance = " << dapos.distanc
    cout << ", angle = " << dapos.angle *
    cout << " degrees\n";
}
```

If you are provided with the source code, you
the source with your compiler.

Storage Duration, Scope,

Now that you've seen a multifile program, it's
memory schemes in Chapter 4, "Compound
information can be shared across files. It migh
Chapter 4, so let's review what it says about m
(four under C++11) for storing data, and the
data in memory:

- **Automatic storage duration**—Varia
including function parameters—have a
when program execution enters the fu
and the memory used for them is freed
block. C++ has two kinds of automatic
- **Static storage duration**—Variables d
by using the keyword `static` have stat
time a program is running. C++ has th
- **Thread storage duration (C++11)**–
mon. These are CPUs that can handle s
allows a program to split computations

A C++ variable can have one of several sc
termed *block scope*) is known only within the b
block is a series of statements enclosed in brac
but you can have other blocks nested within t
scope (also termed *file scope*) is known through
defined. Automatic variables have local scope,
depending on how it is defined. Names used i
within the parentheses enclosing the argumen
what they are or if they are even present.) Me
Chapter 10, "Objects and Classes"). Variables d
(Now that namespaces have been added to th
become a special case of namespace scope.)

C++ functions can have class scope or nam
they can't have local scope. (Because a functio
tion were to have local scope, it could only be
called by another function. Such a function co

The various C++ storage choices are chara
scope, and their linkage. Let's look at C++'s st
begin by examining the situation before name
how namespaces modify the picture.

```
        cout << "hello\n";
        int websight = -2;    // websight
        cout << websight << ' ' << telede
    }                         // websight
    cout << teledeli << endl;
    ...
}  // teledeli expires
```

But what if you name the variable in the i
that you have two variables of the same name
inner block? In this case, the program interpr
block variable while the program executes st
definition *hides* the prior definition. The new
is temporarily out of scope. When the progra
comes back into scope (see Figure 9.2).

Listing 9.4 illustrates how automatic varia
that contain them.

Listing 9.4 **autoscp.cpp**

```
// autoscp.cpp -- illustrating scope of au
#include <iostream>
void oil(int x);
int main()
{
    using namespace std;

    int texas = 31;
    int year = 2011;
    cout << "In main(), texas = " << texas
    cout << &texas << endl;
    cout << "In main(), year = " << year <
    cout << &year << endl;
    oil(texas);
    cout << "In main(), texas = " << texas
    cout << &texas << endl;
    cout << "In main(), year = " << year <
    cout << &year << endl;
    return 0;
}
```

```
In main(), year = 2011, &year = 0012FEC8
In oil(), texas = 5, &texas = 0012FDE4
In oil(), x = 31, &x = 0012FDF4
In block, texas = 113, &texas = 0012FDD8
In block, x = 31, &x = 0012FDF4
Post-block texas = 5, &texas = 0012FDE4
In main(), texas = 31, &texas = 0012FED4
In main(), year = 2011, &year = 0012FEC8
```

Notice that each of the three `texas` variab
and that the program uses only the particular
ing the value `113` to the `texas` in the inner b
variables of the same name. (As usual, the act
fer from system to system.)

Let's summarize the sequence of events. W
space for `texas` and `year`, and these variables
`oil()`, these variables remain in memory but
`texas`, are allocated and come into scope. Wh
block in `oil()`, the new `texas` passes out of
an even newer definition. The variable `x`, how
define a new `x`. When execution exits the blo
freed, and `texas` #2 comes back into scope. V
`texas` and `x` expire, and the original `texas` a

z being initialized:

```
int w;          // value of w is indetermi
int x = 5;      // initialized with a nume
int big = INT_MAX - 1; // initialized with
int y = 2 * x;  // use previously determin
cin >> w;
int z = 3 * w;  // use new value of w
```

Automatic Variables and the Stack

You might gain a better understanding of auto C++ compiler implements them. Because the shrinks as functions start and terminate, the pr it runs. The usual means is to set aside a sectio aging the flow and ebb of variables. It's called stacked atop old data (that is, at an adjacent lo removed from the stack when a program is fir depends on the implementation, but a compil the size. The program keeps track of the stack base of the stack, where the memory set aside top of the stack, which is the next free memo automatic variables are added to the stack, and available free space following the variables. Wh

and compilers developed in sophistication, th
able was heavily used and perhaps the compil
ment. With C++11, even that hint is being d
explicitly identify a variable as being automat
with variables that would be automatic anyw
cate that you really do want to use an automa
as an external variable. This is the same purpo
important reason for retaining `register`, how
that uses that keyword.

Static Duration Variables

C++, like C, provides static storage duration
linkage (accessible across files), internal linkag
file), and no linkage (accessible to just one fu
three last for the duration of the program; the
ables. Because the number of static variables
gram doesn't need a special device such as a s
allocates a fixed block of memory to hold all
present as long as the program executes. Also
able, the compiler sets it to `0`. Static arrays an
or member set to `0` by default.

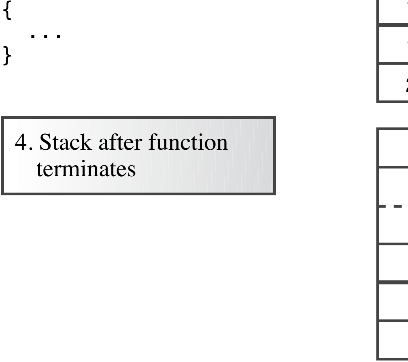


Figure 9.3 Passing arg

Note

Classic K&R C does not allow you to initialize
allow you to initialize static arrays and structu
kinds. But some older C++ translators use C c
you are using such an implementation, you mi
static storage classes for initializing arrays an

```
}
```

As stated previously, all the static duration
this example) persist from the time the progr
variable `count`, which is declared inside `func`
means it can be used only inside the `funct1(`
`llama`. But unlike `llama`, `count` remains in m
not being executed. Both `global` and `one_f`
from the point of declaration until the end o
`main()`, `funct1()`, and `funct2()`. Because `o`
only in the file containing this code. Because
used in other files that are part of the progra

All static duration variables share the follo
static variable has all its bits set to `0`. Such a v

Table 9.1 summarizes the storage class feat
we'll examine the static duration varieties in

Note that the keyword `static` has somew
shown in Table 9.1. When used with a local d
no linkage, `static` indicates the kind of stor
outside of a block, `static` indicates internal
tion. One might term this *keyword overloading*
context.

appropriate type. For example, the null pointe
may have a nonzero internal representation, so
that value. Structure members are zero-initiali

Zero-initialization and constant-expression
tialization. This means the variable is initialized
translation unit). Dynamic initialization means

So what determines which form of initiali
ables are zero-initialized, whether or not any i
able is initialized using a constant expression t
the file contents (including included header fi
tialization. The compiler is prepared to do sim
enough information at this time, the variable
following:

```
#include <cmath>
int x;                              // zer
int y = 5;                          // con
long z = 13 * 13;                   // con
const double pi = 4.0 * atan(1.0);  // dyn
```

First, `x`, `y`, `z`, and `pi` are zero-initialized. The
expressions and initializes `y` and `z` to `5` and `16`

On the other hand, C++ has the "one defini
that there can be only one definition of a var
two kinds of variable declarations. One is the
causes storage for the variable to be allocated
simply, a *declaration*. It does not cause storage
variable.

A referencing declaration uses the keywor
Otherwise, a declaration is a definition and ca

```
double up;             // definition, up
extern int blem;       // blem defined el
extern char gr = 'z';  // definition beca
```

If you use an external variable in several fi
that variable (per the one definition rule). Bu
declare that variable using the keyword exte

```
// file01.cpp
extern int cats = 20;  // definition beca
int dogs = 22;         // also a definiti
int fleas;             // also a definiti
...
// file02.cpp
// use cats and dogs from file01.cpp
extern int cats;       // not definitions
```

```
void promise ();
int main()
{
   ...
}

void promise ()
{
   ...
}
```

This file defines the variable `process_status`, causing the compiler to allocate space for it.

T
to
w

Figure 9.4 Defining declarati

Note that the one definition rule doesn't n
with a given name. For example, automatic va
in different functions are separate variables, in
its own address. Also as later examples show, a
the same name. However, although a program
name, each version can have only one definiti

```
{
    cout << "Global warming is " << warmi
    update(0.1);            // call funct
    cout << "Global warming is " << warmi
    local();                // call funct
    cout << "Global warming is " << warmi
    return 0;
}
```

Listing 9.6 **support.cpp**

```
// support.cpp -- use external variable
// compile with external.cpp
#include <iostream>
extern double warming;  // use warming fr

// function prototypes
void update(double dt);
void local();

using std::cout;
void update(double dt)      // modifies g
{
    extern double warming;  // optional r
```

```
Global warming is 0.4 degrees.
```

Program Notes

The output of the program in Listings 9.5 anc
update() can access the external variable war
makes to warming shows up in subsequent use

The definition for warming is in Listing 9.5

```
double warming = 0.3;       // warming def
```

Listing 9.6 uses extern to make the warmi
that file:

```
extern double warming;  // use warming fro
```

As the comment indicates, this declaration
externally elsewhere."

In addition, the update() function re-decl
word extern. This keyword means "Use the v
externally." Because that is what update() wo
declaration, this declaration is optional. It serv
to use the external variable.

The local() function demonstrates that w
same name as a global variable, the local versio

constant data because you can use the keyw

```
const char * const months[12] =
{
    "January", "February", "March", "A
    "June", "July", "August", "Septemb
    "November", "December"
};
```

In this example, the first `const` protects the
makes sure that each pointer in the array re
pointed initially.

Static Duration, Internal Linkage

Applying the `static` modifier to a file-scope
ence between internal linkage and external li
grams. In that context, a variable with intern
But a regular external variable has external li
ent files, as the previous example showed.

What if you want to use the same name to
Can you just omit the `extern`?

```
void froobish()
{
     cout << errors;   // uses errors def
     ...
```

This doesn't violate the one definition rule
that the identifier `errors` has internal linkage
nal definition.

> **Note**
>
> In a multifile program, you can define an exte
> files using that variable have to declare that v

You can use an external variable to share d
gram. You can use a static variable with intern
found in just one file. (Namespaces offer an al
a file-scope variable static, you needn't worry
variables found in other files.

Listings 9.7 and 9.8 show how C++ handl
age. Listing 9.7 (`twofile1.cpp`) defines the ex
external variable `harry`. The `main()` function
variables and then calls the `remote_access()`
Listing 9.8 (`twofile2.cpp`) shows that file. In
file uses the `extern` keyword to share `tom` wit

```
    cout << &tom << " = &tom, " << &dick
    cout << &harry << " = &harry\n";
    remote_access();
    return 0;
}
```

Listing 9.8 **twofile2.cpp**

```
// twofile2.cpp -- variables with interna
#include <iostream>
extern int tom;         // tom defined el
static int dick = 10;   // overrides exte
int harry = 200;        // external varia
                        // no conflict wi

void remote_access()
{
    using namespace std;
    cout << "remote_access() reports the
    cout << &tom << " = &tom, " << &dick
    cout << &harry << " = &harry\n";
}
```

tion—you could use them to pass secret accou
appearance.) Also if you initialize a static local
once, when the program starts up. Subsequent
variable the way they do for automatic variabl

Listing 9.9 **static.cpp**

```
// static.cpp -- using a static local vari
#include <iostream>
// constants
const int ArSize = 10;

// function prototype
void strcount(const char * str);

int main()
{
    using namespace std;
    char input[ArSize];
    char next;

    cout << "Enter a line:\n";
    cin.get(input, ArSize);
    while (cin)
    {
```

```
}
```

Incidentally, the program in Listing 9.9 sh
exceed the size of the destination array. Reca
method reads up to the end of the line or up
first. It leaves the newline character in the inp
to read the character that follows the line inp
preceding call to `cin.get(input, ArSize)`
newline character, there are more characters
to reject the rest of the line, but you can mod
next input cycle. The program also uses the f
with `get(char *, int)` causes `cin` to test as

Here is the output of the program in Listi

```
Enter a line:
nice pants
"nice pant" contains 9 characters
9 characters total
Enter next line (empty line to quit):
thanks
"thanks" contains 6 characters
15 characters total
```

```
static
extern
thread_local (added by C++11)
mutable
```

You've already seen most of these, and you gle declaration, except that `thread_local` can prior to C++11, the keyword `auto` could be u variable is an automatic variable. (In C++11, a The keyword `register` is used in a declaratio in C++11, simply is an explicit way of saying `static`, when used with a file-scope declaratio with a local declaration, it indicates static stora `extern` indicates a reference declaration—that defined elsewhere. The keyword `thread_loca` is the duration of the containing thread. A `thr` regular static variable is to the whole program of `const`, so let's look at the cv-qualifiers first

Cv-Qualifiers

Here are the cv-qualifiers:

```
const
volatile
```

```
{
    char name[30];
    mutable int accesses;
    ...
};
const data veep = {"Claybourne Clodde", 0
strcpy(veep.name, "Joye Joux");   // not
veep.accesses++;                  // allo
```

The const qualifier to veep prevents a pr
the mutable specifier to the accesses memb

This book doesn't use volatile or mutab

More About const

In C++ (but not C), the const modifier alte
a global variable has external linkage by defa
age by default. That is, C++ treats a global co
fragment, as if the static specifier had been

```
const int fingers = 10;    // same as sta
int main(void)
{
    ...
```

ing them. Each definition is private to the file
to put constant definitions in a header file. Th
header file in two source code files, they recei

If, for some reason, you want to make a co
`extern` keyword to override the default intern

```
extern const int states = 50;   // definit
```

You then must use the `extern` keyword to
constant. This differs from regular external var
keyword `extern` when you define a variable, b
variable. Keep in mind, however, now that a si
one file can use initialization.

When you declare a `const` within a functi
the constant is usable only when the program
means that you can create constants within a f
about the names conflicting with constants de

Functions and Linkage

Like variables, functions have linkage properti
than for variables. C++, like C, does not allow
so all functions automatically have static storag
long as the program is running. By default, fun
can be shared across files. You can, in fact, use

each file that uses the function should have t

Inline functions are excepted from this rul nitions in a header file. Thus, each file that in inline function definition. However, C++ do particular function be identical.

Where C++ Finds Functions

Suppose you call a function in a particular fi function definition? If the function prototype the compiler looks only in that file for the fun the linker, too) looks in all the program files. you an error message because you can have fails to find any definition in the files, the fun that if you define a function that has the sam uses your version rather than the library vers standard library functions, so you shouldn't r explicit instructions to identify which libraries

Language Linking

Another form of linking, called *language linki* A linker needs a different symbolic name for implement because there can be only one C purposes, a C compiler might translate a C fu

C and C++ language linkage are the only
But implementations have the option of provi

Storage Schemes and Dynamic Al

You've seen the five schemes, excluding threa
for variables (including arrays and structures).
using the C++ new operator (or by using the
kind of memory *dynamic memory*. As you saw i
by the new and delete operators, not by scop
can be allocated from one function and freed
memory, dynamic memory is not LIFO; the o
when and how new and delete are used. Typi
ory chunks: one for static variables (this chunk
variables, and one for dynamic storage.

Although the storage scheme concepts don
to automatic and static pointer variables used
example, suppose you have the following state

```
float * p_fees = new float [20];
```

The 80 bytes (assuming that a float is 4 b
memory until the delete operator frees it. Bu
when program execution exits the block cont

int or double, you can do so by following th
enclosed in parentheses:

```
int *pi = new int (6);   // *pi set to 6
double * pd = new double (99.99);  // *pd
```

The parentheses syntax also can be used w
we haven't got that far yet.

To initialize an ordinary structure or an ar
initialization using braces. The new standard a

```
struct where {double x; double y; double
where * one = new where {2.5, 5.3, 7.2};
int * ar = new int [4] {2,4,6,7};
```

With C++11, you also can use the brace i

```
int *pin = new int {});   // *pi set to 6
double * pdo = new double {99.99};  // *p
```

When `new` Fails

It may be that `new` can't find the requested am
handled that eventuality by having `new` return
throws a `std::bad_alloc` exception. Chapte
vides some short examples showing how each

```
int * pa = new(40 * sizeof(int));
```

As you've seen, a statement with a `new` ope
so, in general, using the `new` operator may do
Similarly,

```
delete pi;
```

invokes the following function call:

```
delete (pi);
```

Interestingly, C++ terms these functions *re*
expertise and desire, you can supply replacem
them to meet your specific requirements. One
ment functions with class scope so that they c
particular class. Your code would use the `new` o
would call upon the replacement `new()` funct

The Placement `new` Operator

Normally, the `new` operator has the responsibi
ory that is large enough to handle the amoun
has a variation, called *placement* `new`, that allow
A programmer might use this feature to set up

```
// now, the two forms of placement new
    p2 = new (buffer1) chaff;    // place
    p4 = new (buffer2) int[20];  // place
...
```

For simplicity, this example uses two static
ment new. So this code allocates space for a c
array of 20 ints in buffer2.

Now that you've made your acquaintance
program. Listing 9.10 uses both new and plac
arrays. This program illustrates some importa
new that we'll discuss after seeing the output.

Listing 9.10 **newplace.cpp**

```
// newplace.cpp -- using placement new
#include <iostream>
#include <new>          // for placement n
const int BUF = 512;
const int N = 5;
char buffer[BUF];       // chunk of memory
int main()
{
    using namespace std;
```

```
    for (i = 0; i < N; i++)
    {
        cout << pd3[i] << " at " << &pd3[i
        cout << pd4[i] << " at " << &pd4[i
    }

    cout << "\nCalling new and placement n
    delete [] pd1;
    pd1= new double[N];
    pd2 = new (buffer + N * sizeof(double)
    for (i = 0; i < N; i++)
        pd2[i] = pd1[i] = 1000 + 60.0 * i;
    cout << "Memory contents:\n";
    for (i = 0; i < N; i++)
    {
        cout << pd1[i] << " at " << &pd1[i
        cout << pd2[i] << " at " << &pd2[i
    }
    delete [] pd1;
    delete [] pd3;
    return 0;
}
```

```
1060 at 006E4AB8; 1060 at 00FD9168
1120 at 006E4AC0; 1120 at 00FD9170
1180 at 006E4AC8; 1180 at 00FD9178
1240 at 006E4AD0; 1240 at 00FD9180
```

Program Notes

The first thing to note about Listing 9.10 is t
array in the `buffer` array; both `p2` and `buffe`
ever, of different types; `p1` is pointer-to-`doub`
the way, that's why the program uses a `(void`
try to display a string.) Meanwhile, regular `ne`
memory, at location `006E4AB0`, which is part

The second point to note is that the secon
new block of memory—one beginning at `00`
`new` results in the same block of memory bei
ning at `00FD9138`. The important fact here is
that is passed to it; it doesn't keep track of wh
and it doesn't search the block for unused me
memory management to the programmer. Fo
provides an offset into the buffer array so tha

```
pd2 = new (buffer + N * sizeof(double)) d
```

Other Forms of Placement new

Just as regular new invokes a new function wit
invokes a new function with two arguments:

```
int * pi = new int;              // invokes
int * p2 = new(buffer) int;      // invokes
int * p3 = new(buffer) int[40]; // invokes
```

The placement new function is not replace
least two parameters, the first of which always
number of bytes requested. Any such overload
if the additional parameters don't specify a loc

Namespaces

Names in C++ can refer to variables, functio
and structure members. When programming p
conflicts increases. When you use class librarie
name conflicts. For example, two libraries mig
and Node, but in incompatible ways. You migh
the Tree from the other, and each might expe
are termed *namespace problems*.

the declarative region is the function) hides t
this variable, the declarative region is the file)
ally see the variable is termed the *scope*, which
along. Figures 9.5 and 9.6 illustrate the terms

C++'s rules about global and local variabl
declarative region can declare names that are
declarative regions. A local variable declared i
variable declared in a second function.

New Namespace Features

C++ now adds the ability to create named n
tive region, one whose main purpose is to pr
names in one namespace don't conflict with
spaces, and there are mechanisms for letting o
a namespace. The following code, for exampl
two namespaces, `Jack` and `Jill`:

```
namespace Jack {
    double pail;                    //
    void fetch();                   //
    int pal;                        //
    struct Well { ... };            //
}
namespace Jill {
```

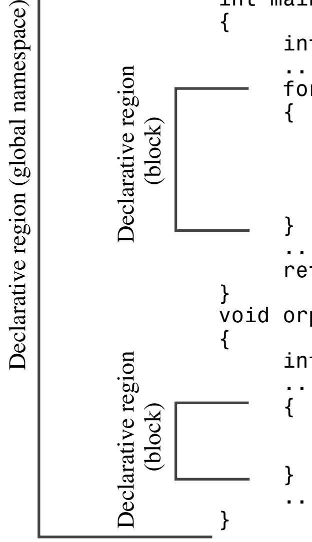


Figure 9.5 Dec

Figure 9.6 Potent

Namespaces are *open*, meaning that you ca
example, the following statement adds the na

```
namespace Jill {
    char * goose(const char *);
}
```

Similarly, the original `Jack` namespace pro
You can provide the code for the function la
`Jack` namespace again:

```
namespace Jack {
    void fetch()
    {
        ...
    }
}
```

```
namespace Jill {
    double bucket(double n) { ... }
    double fetch;
    struct Hill { ... };
}
char fetch;
int main()
{
    using Jill::fetch;   // put fetch into
    double fetch;        // Error! Already
    cin >> fetch;        // read a value i
    cin >> ::fetch;      // read a value i
    ...
}
```

Because a using declaration adds the name
precludes creating another local variable by th
variable, fetch would override a global variab

Placing a using declaration at the external

```
void other();
namespace Jill {
    double bucket(double n) { ... }
    double fetch;
    struct Hill { ... };
```

```
#include <iostream>    // places names in
using namespace std;   // make names avai
```

Placing a `using` directive in a particular fu
function. Here's an example:

```
int main()
{
    using namespace jack; // make names a
...
}
```

You've seen this form often in this book v

One thing to keep in mind about `using` d
increase the possibility of name conflicts. That
namespace `jill` available, and you use the sco

```
jack::pal = 3;
jill::pal =10;
```

The variables `jack::pal` and `jill::pal`
locations. However, if you employ `using` dec

```
using jack::pal;
using jill::pal;
pal = 4;          // which one? now have
```

```
int main()
{
    using namespace Jill;      // import a
    Hill Thrill;               // create a
    double water = bucket(2);  // use Jill
    double fetch;              // not an e
    cin >> fetch;              // read a v
    cin >> ::fetch;            // read a v
    cin >> Jill::fetch;        // read a v
    ...
}

int foom()
{
    Hill top;                 // ERROR
    Jill::Hill crest;         // valid
}
```

Here, in `main()`, the name `Jill::fetch` is
have local scope, so it doesn't override the glo
hides both `Jill::fetch` and the global `fetc`
ables are available if you use the scope-resolut
this example to the preceding one, which uses

This is the approach used for most of this

```
#include <iostream>
int main()
{
    using namespace std;
```

First, the `iostream` header file puts everyt
directive makes the names available within ma

```
#include <iostream>
using namespace std;
int main()
{
```

This exports everything from the `std` nar
main rationale for this approach is expedienc
have namespaces, you can replace the first tw
inal form:

```
#include <iostream.h>
```

However, namespace proponents hope th
the scope-resolution operator or the `using` d
following:

```
using namespace std;  // avoid as too ind
```

```
    {
        int flame;
        ...
    }
    float water;
}
```

In this case, you refer to the `flame` variable
can make the inner names available with this

```
using namespace elements::fire;
```

Also you can use `using` directives and `usi`

```
namespace myth
{
    using Jill::fetch;
    using namespace elements;
    using std::cout;
    using std::cin;
}
```

Suppose you want to access `Jill::fetch`.
`myth` namespace, where it can be called `fetc`

```
std::cin >> myth::fetch;
```

You can make mvft an alias for my_very_
statement:

```
namespace mvft = my_very_favorite_things;
```

You can use this technique to simplify usi

```
namespace MEF = myth::elements::fire;
using MEF::flame;
```

Unnamed Namespaces

You can create an *unnamed namespace* by omi

```
namespace        // unnamed namespace
{
    int ice;
    int bandycoot;
}
```

This code behaves as if it were followed b
declared in this namespace are in potential sc
that contains the unnamed namespace. In thi
are like global variables. However, if a names
using directive or using declaration to mak

```
{
...
}

int other()
{
...
}
```

A Namespace Example

Let's take a look at a multifile example that de
spaces. The first file in this example (see Listin
items normally found in header files—constan
types. In this case, the items are placed in two
tains a definition of a `Person` structure, plus p
with a person's name and a function that disp
namespace, `debts`, defines a structure for stori
money owed to that person. This structure us
space has a `using` directive to make the name
`debts` namespace. The `debts` namespace also

```
    };
    void getDebt(Debt &);
    void showDebt(const Debt &);
    double sumDebts(const Debt ar[], int
}
```

The second file in this example (see Listin
source code file provide definitions for funct
names, which are declared in a namespace, ha
to be in the same namespace as the declaratio
spaces comes in handy. The original namespa
(refer to Listing 9.11). The file then adds the
as shown in Listing 9.12. Also the `namesp.cp`
`std` namespace with the `using` declaration a

Listing 9.12 **namesp.cpp**

```
// namesp.cpp -- namespaces
#include <iostream>
#include "namesp.h"

namespace pers
{
```

```
    void showDebt(const Debt & rd)
    {
        showPerson(rd.name);
        std::cout <<": $" << rd.amount <<
    }
    double sumDebts(const Debt ar[], int n
    {
        double total = 0;
        for (int i = 0; i < n; i++)
            total += ar[i].amount;
        return total;
    }
}
```

Finally, the third file of this program (see L
the structures and functions declared and defi
several methods of making the namespace ide

Listing 9.13 **usenmsp.cpp**

```
// usenmsp.cpp -- using namespaces
#include <iostream>
#include "namesp.h"
```

```
    int i;
    for (i = 0; i < 3; i++)
        getDebt(zippy[i]);

    for (i = 0; i < 3; i++)
        showDebt(zippy[i]);
    cout << "Total debt: $" << sumDebts(z
    return;
}

void another(void)
{
    using pers::Person;
    Person collector = { "Milo", "Rightsh
    pers::showPerson(collector);
    std::cout << std::endl;
}
```

In Listing 9.13, `main()` begins by using tw

```
using debts::Debt;        // makes the Deb
using debts::showDebt;    // makes the sho
```

```
Enter last name: Binx
Enter debt: 100
Enter first name: Cleve
Enter last name: Delaproux
Enter debt: 120
Enter first name: Eddie
Enter last name: Fiotox
Enter debt: 200
Binx, Arabella: $100
Delaproux, Cleve: $120
Fiotox, Eddie: $200
Total debt: $420
Rightshift, Milo
```

Namespaces and the Future

As programmers become more familiar with n
will emerge. Here are some current guideline

- Use variables in a named namespace ins
- Use variables in an unnamed namespac
- If you develop a library of functions or
  C++ currently already calls for placing

Summary

C++ encourages the use of multiple files in
tional strategy is to use a header file to define
for functions to manipulate the user types. Yo
the function definitions. Together, the header
ment the user-defined type and how it can b
using those functions can go into a third file.

C++'s storage schemes determine how lo
duration) and what parts of a program have a
variables are variables that are defined within
within the body. They exist and are known o
in the block that contains the definition. Aut
the storage class specifier `register` or with n
is automatically automatic. The `register` spe
variable is heavily used, but that use is deprec

Static variables exist for the duration of a
function is known to all functions in the file
made available to other files in the program (
such a variable, that file must declare it by us
shared across files should have a defining decl
but it can be used if combined with initializa

b. The secret variable is to be shar

c. The topsecret variable is to be s
den from other files.

d. beencalled keeps track of how r
been called.

2. Describe the differences between a usi

3. Rewrite the following so that it doesn't

```
#include <iostream>
using namespace std;
int main()
{
    double x;
    cout << "Enter value: ";
    while (! (cin >> x) )
    {
        cout << "Bad input. Please e
        cin.clear();
       while (cin.get() != '\n')
           continue;
```

```
}
```

5. Suppose you want the `average(3,6)` f
`int` arguments when it is called in one
average of the two `int` arguments whe
gram. How could you set this up?

6. What will the following two-file progr

```
// file1.cpp
#include <iostream>
using namespace std;
void other();
void another();
int x = 10;
int y;

int main()
{
    cout << x << endl;
    {
        int x = 4;
        cout << x << endl;
        cout << y << endl;
```

```
    cout << "another(): " << x << ",
}
```

7. What will the following program displa

```
#include <iostream>
using namespace std;
void other();
namespace n1
{
    int x = 1;
}

namespace n2
{
    int x = 2;
}


int main()
{
   using namespace n1;
   cout << x << endl;
    {
```

Programming Exercises

1. Here is a header file:

```
// golf.h -- for pe9-1.cpp

const int Len = 40;
struct golf
{
    char fullname[Len];
    int handicap;
};

// non-interactive version:
//  function sets golf structure to
//  using values passed as argument
void setgolf(golf & g, const char *

// interactive version:
//  function solicits name and hand
//  and sets the members of g to th
//  returns 1 if name is entered, 0
int setgolf(golf & g);
```

prototyped functions. For example, a lo
structures and terminate when the arra
for the golfer's name. The `main()` funct
tions to access the golf structures.

2. Redo Listing 9.9, replacing the charact
should no longer have to check whethe
the input string to `""` to check for an e

3. Begin with the following structure dec

```
struct chaff
{
    char dross[20];
    int slag;
};
```

Write a program that uses placement `ne`
a buffer. Then assign values to the struc
`strcpy()` for the `char` array) and use a
use a static array, like that in Listing 9.1
`new` to allocate the buffer.

}

The first file should be a header file th
should be a source code file that exten
the three prototyped functions. The thi
should use the interactive version of se
ture and the non-interactive version of
ond structure. It should display the con
showSales().

- Class methods (also called class functio
- Creating and using class objects
- Class constructors and destructors
- `const` member functions
- The `this` pointer
- Creating arrays of objects
- Class scope
- Abstract data types

Object-oriented programming (OOP) is a
programs, and C++ has enhanced C with fea
approach. The following are the most import

- Abstraction
- Encapsulation and data hiding
- Polymorphism
- Inheritance
- Reusability of code

Let's see, I want to enter the name, times
those who don't follow baseball or softball
divided by the player's official number of ti
player gets on base or makes an out, but c
count as official times at bat), and all thos
Wait, the computer is supposed to make li
out some of that stuff, such as the batting
the results. How should I organize this? I g
tions. Yeah, I'll make `main()` call a functio
make the calculations, and then call a thir
what happens when I get data from the ne
again. Okay, I can add a function to update
in `main()` to select between entering, calc
Hmmm...how am I going to represent the c
the players' names, another array to hold t
to hold the hits, and so on. No, that's dum
information for a single player and then us
the whole team.

In short, with a procedural approach, you f
follow and then think about how to represent
the program running the whole season, you p
a file and read data from a file.)

Abstraction and Classes

Life is full of complexities, and one way we c
abstractions. You are a collection of more tha
mind would say that your mind is a collectio
much simpler to think of yourself as a single
cial step of representing information in term
abstract the essential operational features of a
terms. In the softball statistics example, the in
updates, and displays the data. From abstractio
which in C++ is a class design that impleme

What Is a Type?

Let's think a little more about what constitut
subscribe to the popular stereotype, you migh
black-rimmed glasses, pocket protector full o
might conclude that a nerd is better defined
he or she responds to an awkward social situa
don't mind stretched analogies, with a proced
think of a data type in terms of its appearanc

information yourself. In exchange for this ext
to custom fit new data types to match real-wo

Classes in C++

A *class* is a C++ vehicle for translating an abst
data representation and methods for manipula
look at a class that represents stocks.

First, you have to think a bit about how to
of stock as the basic unit and define a class to
that you would need 100 objects to represent
you can represent a person's current holdings
number of shares owned would be part of the
would have to maintain records of such thing
chase for tax purposes. Also it would have to
seems a bit ambitious for a first effort at defin
ized, simplified view of matters. In particular,
form to the following:

- Acquire stock in a company.
- Buy more shares of the same stock.
- Sell stock.
- Update the per-share value of a stock.
- Display information about the holdings

definitions supply the details.

What Is an Interface?

An *interface* is a shared framework for intera
between a computer and a printer or betwee
the user might be you and the program migh
processor, you don't transfer words directly f
Instead, you interact with the interface provi
computer shows you a character on the scre
moves a cursor on the screen. You click the
pens to the paragraph you were typing. The
your intentions to specific information store

For classes, we speak of the public interface
the class, the interacting system consists of
the methods provided by whoever wrote the
mer, to write code that interacts with class o
the class objects. For example, to find the n
don't open up the object to what is inside; y
class creators. It turns out that the class de
But the public is allowed to use the `size()`
the public interface between the user and a
method is part of the `istream` class public

Listing 10.1 **stock00.h**

```
// stock00.h -- Stock class interface
// version 00
#ifndef STOCK00_H_
#define STOCK00_H_

#include <string>

class Stock  // class declaration
{
private:
    std::string company;
    long shares;
    double share_val;
    double total_val;
    void set_tot() { total_val = shares *
public:
    void acquire(const std::string & co, l
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};    // note semicolon at the end

#endif
```

tions come later in the implementation file, b
function interfaces. The binding of data and r
feature of the class. Because of this design, cre
the rules governing how that object can be u

You've already seen how the `istream` and
such as `get()` and `getline()`. The function p
demonstrate how member functions are estab
ple, has a `getline()` prototype in the `istrea`

Access Control

Also new are the keywords `private` and `pub`
class members. Any program that uses an obje
portions directly. A program can access the pr
public member functions (or, as you'll see in
friend function). For example, the only way t
class is to use one of the `Stock` member func
as go-betweens between a program and an ob
interface between object and program. This i
gram is called *data hiding*. (C++ provides a th
we'll discuss when we cover class inheritance
Figure 10.1.) Whereas data hiding may be an
prospectus, it's a good practice in computing

that constitute the public interface for
the class (abstraction)

Figure 10.1 T

A class design attempts to separate the publ mentation. The public interface represents the ering the implementation details together and called *encapsulation*. *Data hiding* (putting data i instance of encapsulation, and so is hiding fun private section, as the `Stock` class does with s is the usual practice of placing class function d declaration.

OOP and C++

OOP is a programming style that you can use you can incorporate many OOP ideas into ord vides an example (see Listings 9.1, 9.2, 9.3) totype along with the prototypes for functions function simply defines variables of that struc handle those variables; `main()` does not dire that example defines an abstract type that pl types in a header file, hiding the actual data r

Member Access Control: Public or Priva

You can declare class members, whether they
in the public or the private section of a class.
OOP is to hide the data, data items normally
functions that constitute the class interface go
call those functions from a program. As the s
member functions in the private section. You
gram, but the public methods can use them. T
to handle implementation details that don't f

You don't have to use the keyword priva
default access control for class objects:

```
class World
{
    float mass;          // private by de
    char name[20];       // private by de
public:
    void tellall(void);
    ...
};
```

However, this book explicitly uses the pri
of data hiding.

(::) to indicate to which class the function be
update() member function looks like this:

```
void Stock::update(double price)
```

This notation means you are defining the u
Stock class. Not only does this identify updat
use the same name for a member function for
function for a Buffoon class would have this f

```
void Buffoon::update()
```

Thus, the scope-resolution operator resolve
method definition applies. We say that the ider
ber functions of the Stock class can, if necessa
the scope-resolution operator. That's because t
update() in scope. Using update() outside th
however, requires special measures, which we'

One way of looking at method names is th
includes the class name. Stock::update() is c
simple update(), on the other hand, is an abb
name—one that can be used just in class scop

The second special characteristic of metho
members of a class. For example, the show() r

```
{
    company = co;
    if (n < 0)
    {
        std::cout << "Number of shares ca
                  << company << " shares
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}

void Stock::buy(long num, double price)
{
     if (num < 0)
    {
        std::cout << "Number of shares pu
              << "Transaction is aborted.\
    }
    else
    {
        shares += num;
```

```
    }
}

void Stock::update(double price)
{
    share_val = price;
    set_tot();
}

void Stock::show()
{
    std::cout << "Company: " << company
              << "  Shares: " << shares <<
              << "  Share Price: $" << sha
              << "  Total Worth: $" << tot
}
```

Member Function Notes

The `acquire()` function manages the first ac
whereas `buy()` and `sell()` manage adding to
`buy()` and `sell()` methods make sure that th
negative number. Also if the user attempts to s

the class implementation section:

```
class Stock
{
private:
    ...
    void set_tot();  // definition kept s
public:
    ...
};

inline void Stock::set_tot()  // use inli
{
    total_val = shares * share_val;
}
```

The special rules for inline functions requ they are used. The easiest way to make sure th in a multifile program is to include the inline the corresponding class is defined. (Some dev that allow the inline definitions to go into a s

Incidentally, according to the *rewrite rule*, d equivalent to replacing the method definition definition as an inline function immediately joe.shares and joe.share_val, respectively.

> **Note**
>
> When you call a member function, it uses the
> invoke the member function.

Similarly, the function call `kate.sell()` in
`kate.set_tot()`, causing that function to get
Each new object you create contains storag
members. But all objects of the same class sha
one copy of each method. Suppose, for examp
that case, `kate.shares` occupies one chunk o
ond chunk of memory. But `kate.show()` and
method—that is, both execute the same block
ent data. Calling a member function is what s
Thus, sending the same message to two differe
applies it to two different objects (see Figure 1

Using Classes

In this chapter you've seen how to define a cl
produce a program that creates and uses objec
classes as similar as possible to using the basic,
create a class object by declaring a class variab

function with Kate data

Figure 10.2 Objects, da

class type. You can pass objects as arguments, r
assign one object to another. C++ provides f
and cout to recognize objects, and even prov
objects of similar classes. It will be a while be
now with the simpler properties. Indeed, you
and call a member function. Listing 10.3 prov
implementation files. It creates a Stock objec
simple, but it tests the features built in to the
the techniques for multifile programs describ
C++," and in Chapter 9. In particular, compi
present in the same directory or folder.

Listing 10.3 **usestok0.cpp**

```
// usestck0.cpp -- the client program
// compile with stock00.cpp
#include <iostream>
#include "stock00.h"
```

```
Company: NanoSmart  Shares: 35
  Share Price: $18.125  Total Worth: $634.
Company: NanoSmart  Shares: 300035
  Share Price: $40.125  Total Worth: $1.20
Company: NanoSmart  Shares: 35
  Share Price: $0.125  Total Worth: $4.375
```

Note that `main()` is just a vehicle for testin
`Stock` class works as you want it to, you can u
grams. The critical point in using the new typ
tions do; you shouldn't have to think about th
sidebar, "The Client/Server Model."

The Client/Server Model

OOP programmers often discuss program des
conceptualization, the *client* is a program tha
ing the class methods, constitute the *server*,
grams that need it. The client uses the serve
This means that the client's only responsibilit
only responsibility, is to know that interface. T
the server's designer's responsibility, is to se
forms according to that interface. Any change
design should be to details of implementatio
mers to improve the client and the server ind
the server having unforeseen repercussions o

Listing 8.8, using return values for the setting

```
std::streamsize prec =
    std::cout.precision(3);  // save prec
...
std::cout.precision(prec);         // r

// store original flags
std::ios_base::fmtflags orig = std::cout.
...
// reset to stored values
std::cout.setf(orig, std::ios_base::float
```

As you may recall, `fmtflags` is a type defi in the `std` namespace, hence the rather long the flags, and the reset statement uses that inf `floatfield` section, which includes flags for Third, let's not worry too much about the de changes are confined to the implementation aspects of the program using the class.

After this replacement and leaving the head
recompile the program. Now the output wou

```
Company: NanoSmart  Shares: 20
  Share Price: $12.500  Total Worth: $250.
Company: NanoSmart  Shares: 35
  Share Price: $18.125  Total Worth: $634.
You can't sell more than you have! Transac
Company: NanoSmart  Shares: 35
  Share Price: $18.125  Total Worth: $634.
Company: NanoSmart  Shares: 300035
  Share Price: $40.125  Total Worth: $1203
Company: NanoSmart  Shares: 35
  Share Price: $0.125  Total Worth: $4.38
```

Reviewing Our Story to Date

The first step in specifying a class design is to
tion is modeled after a structure declaration a
members. The declaration has a private sectio
be accessed only through the member functio
and members declared there can be accessed

```
char * Bozo::Retort()
```

In other words, `Retort()` is not just a typ
tion that belongs to the `Bozo` class. The full, c
`Bozo::Retort()`. The name `Retort()`, on th
fied name, and it can be used only in certain
class methods.

Another way of describing this situation is
so the scope-resolution operator is needed to
class declaration and a class method.

To create an object, which is a particular e
it were a type name:

```
Bozo bozetta;
```

This works because a class *is* a user-define

You invoke a class member function, or m
using the dot membership operator:

```
cout << Bozetta.Retort();
```

This invokes the `Retort()` member funct
refers to a particular data member, the functi
`bozetta` object.

ceed in initializing an object. (You could initia
the data members public instead of private, bu
the main justifications for using classes: data hi

In general, it's best that all objects be initial
consider the following code:

```
Stock gift;
gift.buy(10, 24.75);
```

With the current implementation of the St
the `company` member. The class design assume
any other member functions, but there is no v
around this difficulty is to have objects initiali
accomplish this, C++ provides for special me
cially for constructing new objects and assigni
cisely, C++ provides a name for these membe
you provide the method definition. The name
possible constructor for the `Stock` class is a m
structor prototype and header have an interest
no return value, it's not declared type `void`. In

```
    if (n < 0)
    {
        std::cerr << "Number of shares ca
                     << company << " shares
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
```

This is the same code that the `acquire()` ference is that in this case, a program automa declares an object.

Member Names and Parameter Name

Often those new to constructors try to use th in the constructor, as in this example:

```
// NO!
Stock::Stock(const string & company, l
{
```

```
    long shares_;
    ...
```

With either convention, you then can use com
the public interface.

Using Constructors

C++ provides two ways to initialize an object
the constructor explicitly:

```
Stock food = Stock("World Cabbage", 250, 1
```

This sets the `company` member of the `food`
`shares` member to `250`, and so on.

The second way is to call the constructor i

```
Stock garment("Furry Mason", 50, 2.5);
```

This more compact form is equivalent to t

```
Stock garment = Stock("Furry Mason", 50, 2
```

C++ uses a class constructor whenever you
you use `new` for dynamic memory allocation.

```
Stock *pstock = new Stock("Electroshock Ga
```

structor would look like this:

```
Stock::Stock() { }
```

The net result is that the `fluffy_the_cat`
ized, just as the following creates `x` without p

```
int x;
```

The fact that the default constructor has n
appear in the declaration.

A curious fact about default constructors
don't define any constructors. After you defin
ity for providing a default constructor for tha
provide a nondefault constructor, such as `Stc`
`pr)`, and don't provide your own version of a
this becomes an error:

```
Stock stock1;  // not possible with curre
```

The reason for this behavior is that you m
uninitialized objects. If, however, you wish to
you must define your own default constructo
ments. You can define a default constructor tw
all the arguments to the existing constructor:

```
Stock(const string & co = "Error", int n
```

After you've used either method (no argu
create the default constructor, you can declare
explicitly:

```
Stock first;                // calls defau
Stock first = Stock();      // calls it ex
Stock *prelief = new Stock; // calls it im
```

However, you shouldn't be misled by the i

```
Stock first("Concrete Conglomerate");
Stock second();
Stock third;
```

The first declaration here calls the nondefa
arguments. The second declaration states that
object. When you implicitly call the default co

Destructors

When you use a constructor to create an obje
of tracking that object until it expires. At that
cial member function bearing the formidable
up any debris, so it actually serves a useful pur
`new` to allocate memory, the destructor should
constructor doesn't do anything fancy like usi

When should a destructor be called? The
code shouldn't explicitly call a destructor. (Se
`new`" in Chapter 12, "Classes and Dynamic M
create a static storage class object, its destruct
terminates. If you create an automatic storage
doing, its destructor is called automatically w
which the object is defined. If the object is c
ory, or the free store, and its destructor is call
free the memory. Finally, a program can creat
operations; in that case, the program automat
it has finished using it.

Because a destructor is called automaticall
be a destructor. If you don't provide one, the
structor and, if it detects code that leads to th
nition for the destructor.

Improving the `Stock` Class

At this point we need to incorporate the con
method definitions. Given the significance of
from `stock00.h` to `stock10.h`. The class met
Finally, we place the program using these reso

```
// two constructors
    Stock();        // default constructor
    Stock(const std::string & co, long n =
    ~Stock();       // noisy destructor
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show();
};

#endif
```

The Implementation File

Listing 10.5 provides the method definitions f
stock10.h file in order to provide the class de
name in double quotation marks instead of in
at the same location where your source files a
iostream header file to provide I/O support.
and qualified names (such as std::string) to
header files. This file adds the constructor and
methods. To help you see when these method

```
    if (n < 0)
    {
        std::cout << "Number of shares ca
                  << company << " shares
        shares = 0;
    }
    else
        shares = n;
    share_val = pr;
    set_tot();
}
// class destructor
Stock::~Stock()        // verbose class d
{
    std::cout << "Bye, " << company << "!
}

// other methods
void Stock::buy(long num, double price)
{
     if (num < 0)
    {
        std::cout << "Number of shares pu
```

```
    else
    {
        shares -= num;
        share_val = price;
        set_tot();
    }
}

void Stock::update(double price)
{
    share_val = price;
    set_tot();
}

void Stock::show()
{
    using std::cout;
    using std::ios_base;
    // set format to #.###
    ios_base::fmtflags orig =
        cout.setf(ios_base::fixed, ios_bas
    std::streamsize prec = cout.precision(

    cout << "Company: " << company
```

```
// compile with stock10.cpp
#include <iostream>
#include "stock10.h"

int main()
{
  {
    using std::cout;
    cout << "Using constructors to create
    Stock stock1("NanoSmart", 12, 20.0);
    stock1.show();
    Stock stock2 = Stock ("Boffo Objects"
    stock2.show();

    cout << "Assigning stock1 to stock2:\
    stock2 = stock1;
    cout << "Listing stock1 and stock2:\n
    stock1.show();
    stock2.show();

    cout << "Using a constructor to reset
    stock1 = Stock("Nifty Foods", 10, 50.
    cout << "Revised stock1:\n";
```

```
Constructor using Nifty Foods called
Bye, Nifty Foods!
Revised stock1:
Company: Nifty Foods  Shares: 10
  Share Price: $50.00  Total Worth: $500.0
Done
Bye, NanoSmart!
Bye, Nifty Foods!
```

Some compilers may produce a program w
one additional line:

```
Using constructors to create new objects
Constructor using NanoSmart called
Company: NanoSmart  Shares: 12
  Share Price: $20.00  Total Worth: $240.0
Constructor using Boffo Objects called
Bye, Boffo Objects!
Company: Boffo Objects  Shares: 2
  Share Price: $2.00  Total Worth: $4.00
...
```

The following "Program Notes" section ex
this output.

The C++ Standard gives a compiler a cou
is to make it behave exactly like the first synt

```
Constructor using Boffo Objects called
Company: Boffo Objects  Shares: 2
```

The second way is to allow the call to the
is then copied to `stock2`. Then the temporar
option, the destructor is called for the tempo

```
Constructor using Boffo Objects called
Bye, Boffo Objects!
Company: Boffo Objects  Shares: 2
```

The compiler that produced this output d
but it's possible that a compiler might wait lo
would be displayed later.

The following statement illustrates that yo
same type:

```
stock2 = stock1;     // object assignment
```

As with structure assignment, class object
one object to the other. In this case, the origi

Finally, at the end, the program displays thi

```
Done
Bye, NanoSmart!
Bye, Nifty Foods!
```

When the `main()` function terminates, its l
from your plane of existence. Because such au
object created is the first deleted, and the first
`"NanoSmart"` was originally in `stock1` but wa
was reset to `"Nifty Foods"`.)

The output points out that there is a funda
two statements:

```
Stock stock2 = Stock ("Boffo Objects", 2,
stock1 = Stock("Nifty Foods", 10, 50.0); /
```

The first of these statements invokes initial
cated value, and it may or may not create a te
invokes assignment. Using a constructor in an
causes the creation of a temporary object befo

Tip

If you can set object values either through ini
tion. It is usually more efficient.

```
const Stock land = Stock("Kludgehorn Prop
land.show();
```

With current C++, the compiler should c
code for show() fails to guarantee that it wo
because it is const, should not be altered. We
declaring a function's argument to be a cons
a syntax problem: The show() method doesn
Instead, the object it uses is provided implicit
is a new syntax, one that says a function pron
C++ solution is to place the const keyword
show() declaration should look like this:

```
void show() const;          // promises no
```

Similarly, the beginning of the function de

```
void stock::show() const   // promises no
```

Class functions declared and defined this v
as you should use const references and point
appropriate, you should make class methods c
invoking object. We'll follow this rule from h

```
Bozo *pc = new Bozo{"Popo", "Le Peu"};
```

If a constructor has just one argument, that
object to a value that has the same type as the
pose you have this constructor prototype:

```
Bozo(int age);
```

Then you can use any of the following for

```
Bozo dribble = bozo(44);   // primary form
Bozo roon(66);             // secondary fo
Bozo tubby = 32;           // special form
```

Actually, the third example is a new point,
time to tell you about it. Chapter 11 mention
lead to unpleasant surprises.

> **Caution**
>
> A constructor that you can use with a single a
> to initialize an object to a value:
>
> ```
> Classname object = value;
> ```
>
> This feature can cause problems, but it can b

You can do still more with the Stock class. S
with but a single object: the object that invok
need to deal with two objects, and doing so r
this. Let's look at how the need for this car

Although the Stock class declaration displ
example, by looking at the show() output, yo
greatest value, but the program can't tell beca
most direct way of letting a program know a
return values. Typically, you use inline code fo

```
class Stock
{
private:
    ...
    double total_val;
    ...
public:
    double total() const { return total_v
    ...
};
```

This definition, in effect, makes total_va
gram access is concerned. That is, you can use
value, but the class doesn't provide a method

reference to one of those two objects. The co
won't modify the explicitly accessed object, a
states that the function won't modify the imp
returns a reference to one of the two `const` o
reference.

Suppose, then, that you want to compare t
assign the one with the greater total value to
lowing statements to do so:

```
top = stock1.topval(stock2);
top = stock2.topval(stock1);
```

The first form accesses `stock1` implicitly a
accesses `stock1` explicitly and `stock2` implic
compares the two objects and returns a refere

Actually, this notation is a bit confusing. It
the relational operator `>` to compare the two
loading, which Chapter 11 discusses.

Meanwhile, there's still the implementatio
problem. Here's a partial implementation that

```
const Stock & Stock::topval(const Stock &
{
    if (s.total_val > total_val)
```

Figure 10.3 Access
a membe

Here `s.total_val` is the total value for th
`total_val` is the total value for the object to
is greater than `total_val`, the function retur
reference to the object used to evoke the me
which the `topval` message is sent.) Here's th
you make the call `stock1.topval(stock2)`,
alias for `stock2`), but there is no alias for `sto`

The C++ solution to this problem is to u
pointer points to the object used to invoke a
as a hidden argument to the method.) Thus,
sets `this` to the address of the `stock1` object
`topval()` method. Similarly, the function cal
address of the `stock2` object. In general, all c
address of the object that invokes the metho
shorthand notation for `this->total_val`. (R
that you use the `->` operator to access structu
for class members.) (See Figure 10.4.)

so s is joe, this points to kate,
and *this is kate

Figure 10.4 this poin

Note

Each member function, including constructors
cial property of the this pointer is that it poi
refer to the invoking object as a whole, it can
qualifier after the function argument parenthe
const; in that case, you can't use this to ch

What you want to return, however, is not
object. You want to return the object itself, an
applying the dereferencing operator * to a po
points.) Now you can complete the method
invoking object:

```
const Stock & Stock::topval(const Stock &
{
   if (s.total_val > total_val)
      return s;          // argument obj
   else
      return *this;      // invoking obj
}
```

```
    void buy(long num, double price);
    void sell(long num, double price);
    void update(double price);
    void show()const;
    const Stock & topval(const Stock & s)
};

#endif
```

Listing 10.8 presents the revised class met method. Also now that you've seen how the 10.8 replaces them with silent versions.

Listing 10.8 **stock20.cpp**

```
// stock20.cpp -- augmented version
#include <iostream>
#include "stock20.h"

// constructors
Stock::Stock()        // default construc
{
    company = "no name";
    shares = 0;
```

```
}

// other methods
void Stock::buy(long num, double price)
{
     if (num < 0)
    {
        std::cout << "Number of shares pur
             << "Transaction is aborted.\n
    }
    else
    {
        shares += num;
        share_val = price;
        set_tot();
    }
}

void Stock::sell(long num, double price)
{
    using std::cout;
    if (num < 0)
    {
```

```
{
    using std::cout;
    using std::ios_base;
    // set format to #.###
    ios_base::fmtflags orig =
        cout.setf(ios_base::fixed, ios_ba
    std::streamsize prec = cout.precision

    cout << "Company: " << company
        << "  Shares: " << shares << '\n'
    cout << "  Share Price: $" << share_v
    // set format to #.##
    cout.precision(2);
    cout << "  Total Worth: $" << total_v

    // restore original format
    cout.setf(orig, ios_base::floatfield)
    cout.precision(prec);
}

const Stock & Stock::topval(const Stock &
{
    if (s.total_val > total_val)
```

ment—mystuff[0], mystuff[1], and so on—
with the Stock methods:

```
mystuff[0].update();      // apply update
mystuff[3].show();        // apply show()
const Stock * tops = mystuff[2].topval(mys
     // compare 3rd and 2nd elements and
     // to point at the one with a higher
```

You can use a constructor to initialize the
the constructor for each individual element:

```
const int STKS = 4;
Stock stocks[STKS] = {
    Stock("NanoSmart", 12.5, 20),
    Stock("Boffo Objects", 200, 2.0),
    Stock("Monolithic Obelisks", 130, 3.25
    Stock("Fleep Enterprises", 60, 6.5)
    };
```

Here the code uses the standard form for i
of values enclosed in braces. In this case, a call

```
// usestok2.cpp -- using the Stock class
// compile with stock20.cpp
#include <iostream>
#include "stock20.h"

const int STKS = 4;
int main()
{
// create an array of initialized objects
    Stock stocks[STKS] = {
        Stock("NanoSmart", 12, 20.0),
        Stock("Boffo Objects", 200, 2.0),
        Stock("Monolithic Obelisks", 130,
        Stock("Fleep Enterprises", 60, 6.
        };

    std::cout << "Stock holdings:\n";
    int st;
    for (st = 0; st < STKS; st++)
        stocks[st].show();
// set pointer to first element
    const Stock * top = &stocks[0];
    for (st = 1; st < STKS; st++)
        top = &top->topval(stocks[st]);
```

class. When that's done, writing the program r

Incidentally, knowing about the `this` poin
under the skin. For example, the original Uni
`cfront` that converted C++ programs to C p
had to do is convert a C++ method definitio

```
void Stock::show() const
{
    cout << "Company: " << company
         << "  Shares: " << shares << '\n'
         << "  Share Price: $" << share_va
         << "  Total Worth: $" << total_va
}
```

to the following C-style definition:

```
void show(const Stock * this)
{
    cout << "Company: " << this->company
         << "  Shares: " << this->shares <
         << "  Share Price: $" << this->sh
         << "  Total Worth: $" << this->to
}
```

without conflict. For example, the `shares` m
`shares` member of a `JobRide` class. Also clas
bers of a class from the outside world. This is
is, to invoke a public member function, you

```
Stock sleeper("Exclusive Ore", 100, 0.25)
sleeper.show();      // use object to inv
show();              // invalid -- can't
```

Similarly, you have to use the scope-resol
functions:

```
void Stock::update(double price)
{
    ...
}
```

In short, within a class declaration or a m
unadorned member name (the unqualified n
member function. A constructor name is rec
the same as the class name. Otherwise, you m
the indirect membership operator (`->`), or th
on the context, when you use a class membe
trates how identifiers with class scope can be

Class Scope Constants

Sometimes it would be nice to have symbolic class declaration might use the literal 30 to sp the same for all objects, it would be nice to c You might think the following would be a so

```
class Bakery
{
private:
    const int Months = 12;    // declare a
    double costs[Months];
    ...
```

But this won't work because declaring a cl doesn't create an object. Hence, until you crea value. (Actually, C++11 provides for member make the preceding array declaration work; C however, a couple ways to achieve essentially

First, you can declare an enumeration with declaration has class scope, so you can use enu names for integer constants. That is, you can s

...

This creates a single constant called `Month`
rather than in an object. Thus, there is only o
objects. Chapter 12 looks further into static c
technique only for declaring static constants
C++98 doesn't allow you to store a `double`
restriction.

Scoped Enumerations (C++11)

Traditional enumerations have some problem
ent `enum` definitions can conflict. Suppose yo
and T-shirts. You might try something like th

```
enum egg {Small, Medium, Large, Jumbo};
enum t_shirt {Small, Medium, Large, Xlarg
```

This won't fly because the `egg Small` and
same scope, and the names conflict. C++11 p
avoids this problem by having class scope for
form look like this:

```
enum class egg {Small, Medium, Large, Jum
enum class t_shirt {Small, Medium, Large,
```

But you can do an explicit type conversion

```
int Frodo = int(t_shirt::Small); // Frodo
```

Enumerations are represented by some und
choice was implementation-dependent. Thus,
might be of different sizes on different system
scoped enumerations. By default, the underlyi
`int`. Furthermore, there's a syntax for indicati

```
// underlying type for pizza is short
enum class : short pizza {Small, Medium, I
```

The `: short` specifies the underlying type
an integer type. Under C++11, you also can u
type for an unscoped enumeration, but if you
piler makes is implementation-dependent.

Abstract Data Types

The `Stock` class is pretty specific. Often, howe
more general concepts. For example, using cla
puter scientists describe as *abstract data types* (A
describes a data type in a general fashion with
tion details. Consider, for example, the stack. I

tion. Instead, it should be expressed in gener
item, and so on. Listing 10.10 shows one app
implemented. If it hasn't been implemented
rather than bool, false, and true.

Listing 10.10 **stack.h**

```
// stack.h -- class definition for the st
#ifndef STACK_H_
#define STACK_H_

typedef unsigned long Item;

class Stack
{
private:
    enum {MAX = 10};    // constant speci
    Item items[MAX];    // holds stack it
    int top;            // index for top
public:
    Stack();
    bool isempty() const;
    bool isfull() const;
    // push() returns false if stack alre
    bool push(const Item & item);   // ad
```

method for isolating from the class design the

Next, you need to implement the class me

Listing 10.11 **stack.cpp**

```
// stack.cpp -- Stack member functions
#include "stack.h"
Stack::Stack()    // create an empty stack
{
    top = 0;
}

bool Stack::isempty() const
{
    return top == 0;
}

bool Stack::isfull() const
{
    return top == MAX;
}

bool Stack::push(const Item & item)
{
```

like this are one of the things that make OO
separate array to represent the stack and an in
the top. In that case, it is your responsibility t
new stack. Without the protection that privat
making some program blunder that alters dat

Let's test this stack. Listing 10.12 models t
orders from the top of his in-basket, using th

Listing 10.12 **stacker.cpp**

```
// stacker.cpp -- testing the Stack class
#include <iostream>
#include <cctype>  // or ctype.h
#include "stack.h"
int main()
{
    using namespace std;
    Stack st; // create an empty stack
    char ch;
    unsigned long po;
    cout << "Please enter A to add a purc
        << "P to process a PO, or Q to qu
    while (cin >> ch && toupper(ch) != 'Q
    {
```

```
                    }
                    break;
        }
        cout << "Please enter A to add a p
             << "P to process a PO, or Q t
    }
    cout << "Bye\n";
    return 0;
}
```

The little `while` loop in Listing 10.12 that
absolutely necessary at this point, but it will c
gram in Chapter 14. Here's a sample run:

```
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
A
Enter a PO number to add: 17885
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
P
PO #17885 popped
Please enter A to add a purchase order,
```

Summary

OOP emphasizes how a program represents
gramming problem by using the OOP appro
interface with the program, specifying how t
class that implements the interface. Typically,
whereas public member functions, also calle
data. The class combines data and methods in
plishes data hiding.

Usually, you separate a class declaration in
The class declaration proper goes into a head
function prototypes. The source code that de
methods file. This approach separates the desc
the implementation. In principle, you need to
the class. Of course, you can look at the impl
in compiled form only), but your program sh
tion, such as knowing that a particular value
a class communicate only through methods o
either part separately without worrying abou

the class methods provide an implementation

Chapter Review

1. What is a class?

2. How does a class accomplish abstractio

3. What is the relationship between an ob

4. In what way, aside from being function class data members?

5. Define a class to represent a bank accou depositor's name, the account number (tions should allow the following:

   - Creating an object and initializin
   - Displaying the depositor's name, a
   - Depositing an amount of money
   - Withdrawing an amount of mon

   Just show the class declaration, not the Exercise 1 provides you with an opport

```
public:
    Person() {lname = ""; fname[0] 
    Person(const string & ln, const
// the following methods display ln
    void Show() const;         // fi
    void FormalShow() const;  // la
};
```

(It uses both a string object and a cha
the two forms are used.) Write a progr
providing code for the undefined meth
should also use the three possible const
and two arguments) and the two displa
constructors and methods:

```
Person one;                        /
Person two("Smythecraft");         /
Person three("Dimwiddy", "Sam");  /
one.Show();
cout << endl;
one.FormalShow();
// etc. for two and three
```

3. Do Programming Exercise 1 from Cha
an appropriate golf class declaration. R

typedef declaration so that Item is typ

6. Here's a class declaration:

```
class Move
{
private:
    double x;
    double y;
public:
    Move(double a = 0, double b = 0)
    showmove() const;
    Move add(const Move & m) const;
// this function adds x of m to x of
// adds y of m to y of invoking obje
// move object initialized to new x,
    reset(double a = 0, double b = 0
};
```

Create member function definitions and a pro

- You can add items to the list.
- You can determine whether the
- You can determine whether the
- You can visit each item in the lis

As you can see, this list really is simple; it doe

Design a `List` class to represent this ab header file with the class declaration an implementations. You should also creat

The main reason for keeping the list sp gramming exercise. You can implement the data type, as a linked list. But the p choice. That is, the public interface sho and so on. It should be expressed in th

- Class conversion functions

C++ classes are feature-rich, complex, an
Classes," you began a journey toward object-
and use a simple class. You saw how a class d
data to be used to represent an object and by
the operations that can be performed with t
member functions, the constructor and the d
ing objects made to a class specification. This
exploration of class properties, concentrating
general principles. You'll probably find some
and some a bit more subtle. To best understa
examples and experiment with them: What
instead of a reference argument for this func
out of a destructor? Don't be afraid to make
unraveling an error than by doing something
assume that a maelstrom of mistakes inevitab
you'll be rewarded with a fuller understandin
can do for you.

This chapter starts with operator overload
tors such as = and + with class objects. Then
letting nonmember functions access private

Operator Overloading

Let's look at a technique for giving object op
an example of C++ polymorphism. In Chapt
how C++ enables you to define several funct
they have different signatures (argument lists).
functional polymorphism. Its purpose is to let yo
basic operation, even though you apply the o
how awkward English would be if you had to
type of object—for example, lift_lft your left
loading extends the overloading concept to o
ings to C++ operators. Actually, many C++ (
example, the * operator, when applied to an a
address. But applying * to two numbers yields
number and type of operands to decide which

C++ lets you extend operator overloading
to use the + symbol to add two objects. Again
operands to determine which definition of ad
often make code look more natural. For exam
two arrays. Usually, this winds up looking like

```
for (int i = 0; i < 20; i++)
      evening[i] = sam[i] + janet[i];  //
```

The compiler, recognizing the operands a
replaces the operator with the corresponding

```
district2 = sid.operator+(sara);
```

The function then uses the sid object im
the sara object explicitly (because it's passed
it then returns. Of course, the nice part is tha
instead of the clunky function notation.

C++ imposes some restrictions on operat
stand after you've seen how overloading wor
the process and then discuss the limitations.

Time on Our Hands: Deve
Overloading Example

If you worked on the Priggs account for 2 h
40 minutes in the afternoon, how long did y
an example where the concept of addition m
(a mixture of hours and minutes) don't match
C++'s Programming Modules," handles a sin
ture and a sum() function for adding such st
class, using a method to handle addition. Let'

```
#endif
```

The Time class provides methods for adjus
values, and for adding two times. Listing 11.2
the AddMin() and Sum() methods use integer
the minutes and hours values when the total
because the only iostream feature used is co
economical to use std::cout rather than use

Listing 11.2 **mytime0.cpp**

```
// mytime0.cpp  -- implementing Time metho
#include <iostream>
#include "mytime0.h"

Time::Time()
{
    hours = minutes = 0;
}

Time::Time(int h, int m )
{
    hours = h;
```

```
{
    Time sum;
    sum.minutes = minutes + t.minutes;
    sum.hours = hours + t.hours + sum.min
    sum.minutes %= 60;
    return sum;
}

void Time::Show() const
{
    std::cout << hours << " hours, " << m
}
```

Consider the code for the `Sum()` function
that the return type is not a reference. The re
efficiency. The code would produce the same
value, but it's usually faster and more memor

However, the return value cannot be a ref
ates a new `Time` object (`sum`) that represents t
Returning the object, as this code does, creat
tion can use. If the return type were `Time &`,
object. But the `sum` object is a local variable a
nates, so the reference would be a reference t

```
    Time planning;
    Time coding(2, 40);
    Time fixing(5, 55);
    Time total;

    cout << "planning time = ";
    planning.Show();
    cout << endl;

    cout << "coding time = ";
    coding.Show();
    cout << endl;

    cout << "fixing time = ";
    fixing.Show();
    cout << endl;

    total = coding.Sum(fixing);
    cout << "coding.Sum(fixing) = ";
    total.Show();
    cout << endl;

    return 0;
}
```

```
{
private:
    int hours;
    int minutes;
public:
    Time();
    Time(int h, int m = 0);
    void AddMin(int m);
    void AddHr(int h);
    void Reset(int h = 0, int m = 0);
    Time operator+(const Time & t) const;
    void Show() const;
};
#endif
```

Listing 11.5 **mytime1.cpp**

```
// mytime1.cpp  -- implementing Time meth
#include <iostream>
#include "mytime1.h"

Time::Time()
{
```

```
    hours = h;
    minutes = m;
}

Time Time::operator+(const Time & t) const
{
    Time sum;
    sum.minutes = minutes + t.minutes;
    sum.hours = hours + t.hours + sum.minu
    sum.minutes %= 60;
    return sum;
}

void Time::Show() const
{
    std::cout << hours << " hours, " << mi
}
```

Like `Sum()`, `operator+()` is invoked by a `T` argument, and returns a `Time` object. Thus, yo using the same syntax that `Sum()` uses:

```
total = coding.operator+(fixing);    // fu
```

```
cout << "planning time = ";
planning.Show();
cout << endl;

cout << "coding time = ";
coding.Show();
cout << endl;

cout << "fixing time = ";
fixing.Show();
cout << endl;

total = coding + fixing;
// operator notation
cout << "coding + fixing = ";
total.Show();
cout << endl;

Time morefixing(3, 28);
cout << "more fixing time = ";
morefixing.Show();
cout << endl;
total = morefixing.operator+(total);
```

```
C = A + B;      // use addition as defined
```

Can you add more than two objects? For e
objects, can you do the following?

```
t4 = t1 + t2 + t3;                  //
```

The way to answer this is to consider how
calls. Because addition is a left-to-right operat

```
t4 = t1.operator+(t2 + t3);         //
```

Then the function argument is itself transla

```
t4 = t1.operator+(t2.operator+(t3));   //
```

Is this valid? Yes, it is. The function call `t2.`
represents the sum of `t2` and `t3`. This object t
`t1.operator+()` function call, and that call re
that represents the sum of `t2` and `t3`. In short,
and `t3`, just as desired.

addition.

- You can't create new operator symbols
  operator**() function to denote exp
- You cannot overload the following ope

Operator	Descrip
sizeof	The siz
.	The mer
.*	The poi
::	The sco
?:	The con
typeid	An RTTI
const_cast	A type c
dynamic_cast	A type c
reinterpret_cast	A type c
static_cast	A type c

This still leaves all the operators

<table>
<tr><td>></td><td>+=</td><td>-=</td><td>*=</td></tr>
<tr><td>^=</td><td>&=</td><td>|=</td><td><<</td></tr>
<tr><td><<=</td><td>==</td><td>!=</td><td><=</td></tr>
<tr><td>||</td><td>++</td><td>--</td><td>,</td></tr>
<tr><td>()</td><td>[]</td><td>new</td><td>del</td></tr>
</table>

In addition to these formal restrictions, yo
ing operators. For example, you shouldn't ove
data members of two `Time` objects. Nothing i
operator did, so it would be better to define a
such as `Swap()`.

More Overloaded Operators

Some other operations make sense for the `Ti`
subtract one time from another or multiply a
the subtraction and multiplication operators. T
operator: you create `operator-()` and `operat`
ing prototypes to the class declaration:

```
Time operator-(const Time & t) const;
Time operator*(double n) const;
```

```
};
#endif
```

Then you add definitions for the new met
Listing 11.8.

Listing 11.8 **mytime2.cpp**

```
// mytime2.cpp  -- implementing Time meth
#include <iostream>
#include "mytime2.h"

Time::Time()
{
    hours = minutes = 0;
}

Time::Time(int h, int m )
{
    hours = h;
    minutes = m;
}

void Time::AddMin(int m)
{
```

```
Time Time::operator-(const Time & t) const
{
    Time diff;
    int tot1, tot2;
    tot1 = t.minutes + 60 * t.hours;
    tot2 = minutes + 60 * hours;
    diff.minutes = (tot2 - tot1) % 60;
    diff.hours = (tot2 - tot1) / 60;
    return diff;
}

Time Time::operator*(double mult) const
{
    Time result;
    long totalminutes = hours * mult * 60
    result.hours = totalminutes / 60;
    result.minutes = totalminutes % 60;
    return result;
}

void Time::Show() const
{
    std::cout << hours << " hours, " << mi
}
```

```
    cout << "waxing time = ";
    waxing.Show();
    cout << endl;

    cout << "total work time = ";
    total = weeding + waxing;   // use op
    total.Show();
    cout << endl;

    diff = weeding - waxing;    // use op
    cout << "weeding time - waxing time =
    diff.Show();
    cout << endl;

    adjusted = total * 1.5;      // use o
    cout << "adjusted work time = ";
    adjusted.Show();
    cout << endl;

    return 0;
}
```

Before seeing how to make friends, let's lo
overloading a binary operator (that is, an oper
ates a need for friends. Multiplying a `Time` obj
uation, so let's study that case.

In the previous `Time` class example, the ove
from the other two overloaded operators in th
the addition and subtraction operators each co
tion operator combines a `Time` value with a d
can be used. Remember, the left operand is th

```
A = B * 2.75;
```

translates to the following member functio

```
A = B.operator*(2.75);
```

But what about the following statement?

```
A = 2.75 * B;     // cannot correspond to
```

Conceptually, `2.75 * B` should be the sam
not correspond to a member function becaus
the left operand is the invoking object, but `2.`
replace the expression with a member functio

lack access. But there is a special category of
access private members of a class.

Creating Friends

The first step toward creating a friend functio
tion and prefix the declaration with the keyw

```
friend Time operator*(double m, const Tim
```

This prototype has two implications:

- Although the `operator*()` function is
  member function. So it isn't invoked by
- Although the `operator*()` function is
  access rights as a member function.

The second step is to write the function d
tion, you don't use the `Time::` qualifier. Also
definition. The definition should look like thi

```
Time operator*(double m, const Time & t)
{
    Time result;
    long totalminutes = t.hours * mult *
    result.hours = totalminutes / 60;
```

keep in mind that only a class declaration can
class declaration still controls which functions
and friends are simply two different mechanis

Actually, you can write this particular frien
definition so that it switches which value com

```
Time operator*(double m, const Time & t)
{
    return t * m;     // use t.operator*(m
}
```

The original version accessed `t.minutes` a
friend. This version only uses the `Time` object
handle the private values, so this version doesn
are reasons to make this version a friend, too.
part of the official class interface. Second, if yo
private data directly, you only have to change
prototype.

Tip

If you want to overload an operator for a class
class term as the first operand, you can use a

The First Version of Overloading <<

To teach the `Time` class to use `cout`, you can
ment like the following uses two objects, wit

```
cout << trip;
```

If you use a `Time` member function to ove
as it did when you overloaded the `*` operator
would have to use the `<<` operator this way:

```
trip << cout;   // if operator<<() were a
```

This would be confusing. But by using a f
tor this way:

```
void operator<<(ostream & os, const Time
{
    os << t.hours << " hours, " << t.minu
}
```

This lets you use

```
cout << trip;
```

to print data in the following format:

```
4 hours, 23 minutes
```

output to a file. Through the magic of inherita
ofstream objects can use ostream methods
tion to write Time data to files as well as to t
ofstream object instead of cout as the first

The call `cout << trip` should use the cou
passes the object as a reference instead of by v
causes os to be an alias for cout, and the expr
for cerr. The Time object can be passed by va
makes the object values available to the functi
memory and time than passing by value.

The Second Version of Overloading <<

The implementation just presented has a prob

```
cout << trip;
```

But the implementation doesn't allow you
the ones cout normally uses:

```
cout << "Trip time: " << trip << " (Tuesda
```

To understand why this doesn't work and v
need to know a bit more about how cout ope

```
}
```

Note that the return type is `ostream &`. R
returns a reference to an `ostream` object. Bec
the function to begin with, the net effect is th
object passed to it. That is, the statement

```
cout << trip;
```

becomes the following function call:

```
operator<<(cout, trip);
```

And that call returns the `cout` object. So n

```
cout << "Trip time: " << trip << " (Tuesd
```

Let's break this into separate steps to see h
particular `ostream` definition of `<<` that displ

```
cout << "Trip time: "
```

So the expression `cout << "Trip time:`
its return value, `cout`. This reduces the origin

```
cout << trip << " (Tuesday)\n";
```

friend function with a definition in this form:

```
ostream & operator<<(ostream & os, cons
{
    os << ... ;  // display object cont
    return os;
}
```

Listing 11.10 shows the class definition as r `operator*()` and `operator<<()`. It implemen because the code is so short. (When the defin use the `friend` prefix.)

Caution

You use the `friend` keyword only in the proto use it in the function definition unless the de

Listing 11.10 **mytime3.h**

```
// mytime3.h -- Time class with friends
#ifndef MYTIME3_H_
#define MYTIME3_H_
#include <iostream>

class Time
{
```

mytime3.h in mytime3.cpp provides support

Listing 11.11 **mytime3.cpp**

```
// mytime3.cpp  -- implementing Time meth
#include "mytime3.h"

Time::Time()
{
    hours = minutes = 0;
}

Time::Time(int h, int m )
{
    hours = h;
    minutes = m;
}

void Time::AddMin(int m)
{
    minutes += m;
    hours += minutes / 60;
    minutes %= 60;
}
```

```
    int tot1, tot2;
    tot1 = t.minutes + 60 * t.hours;
    tot2 = minutes + 60 * hours;
    diff.minutes = (tot2 - tot1) % 60;
    diff.hours = (tot2 - tot1) / 60;
    return diff;
}

Time Time::operator*(double mult) const
{
    Time result;
    long totalminutes = hours * mult * 60
    result.hours = totalminutes / 60;
    result.minutes = totalminutes % 60;
    return result;
}

std::ostream & operator<<(std::ostream & o
{
    os << t.hours << " hours, " << t.minut
    return os;
}
```

```
    temp = aida + tosca;     // operator+(
    cout << "Aida + Tosca: " << temp << en
    temp = aida* 1.17;  // member operator
    cout << "Aida * 1.17: " << temp << end
    cout << "10.0 * Tosca: " << 10.0 * tos

    return 0;
}
```

Here is the output of the program in Listi

```
Aida and Tosca:
3 hours, 35 minutes; 2 hours, 48 minutes
Aida + Tosca: 6 hours, 23 minutes
Aida * 1.17: 4 hours, 11 minutes
10.0 * Tosca: 28 hours, 0 minutes
```

Overloaded Operators: M Nonmember Functions

For many operators, you have a choice betwe functions to implement operator overloading

```
T1 = T2.operator+(T3);    // member functi
T1 = operator+(T2, T3);   // nonmember fun
```

Keep in mind that you must choose one o
operator, but not both. Because both forms m
forms is an ambiguity error, leading to a comp

Which form, then, is it best to use? For so
ber function is the only valid choice. Otherwi
Sometimes, depending on the class design, the
tage, particularly if you have defined type con
sions and Friends," near the end of this chapte

More Overloading: A Vect

Let's look at another class design that uses ope
senting vectors. This class also illustrates furthe
ing two different ways of describing the same
for vectors, you can use many of the new tech
vector, as the term is used in engineering and p
tude (size) and a direction. For example, if you
how hard you push (the magnitude) and in w
direction can save a tottering vase, whereas a p
to doom. To fully describe the motion of your

position is called a *displacement vector*.

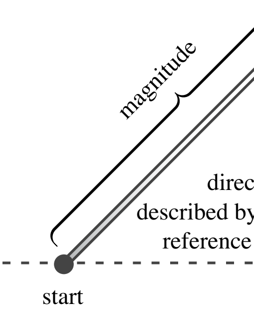


Figure 11.1
displacement

Now say you're Lhanappa, the great mammo
kilometers to the northwest. But because of
from the southeast. So you go 10 kilometers
ing the herd from the south. You know these
same location as the single 14.1-kilometer v
mammoth hunter, also knows how to add vec

Vectors are a natural choice for operator o
tor with a single number, so it makes sense to
vectors have analogs to ordinary arithmetic o
This parallel suggests overloading the correspo
vectors.

To keep things simple, in this section we'll
as a screen displacement, instead of a three-di
movement of a helicopter or a gymnast. You r
dimensional vector, but you have a choice of

- You can describe a vector by its magnit
- You can represent a vector by its x and

The components are a horizontal vector (t
y component), which add up to the final vect
as moving a point 30 units to the right and 40
puts the point at the same spot as moving 50
tal. Therefore, a vector with a magnitude of 50
vector having a horizontal component of 30 a
counts with displacement vectors is where yo

presents a class declaration. To refresh your m
the class declaration inside the VECTOR names
couple constants (RECT and POL) for identifyi
technique in Chapter 10, so we may as well u

Listing 11.13 **vect.h**

```
// vect.h -- Vector class with <<, mode s
#ifndef VECTOR_H_
#define VECTOR_H_
#include <iostream>
namespace VECTOR
{
    class Vector
    {
    public:
        enum Mode {RECT, POL};
    // RECT for rectangular, POL for Pola
    private:
        double x;          // horizontal
        double y;          // vertical va
        double mag;        // length of v
        double ang;        // direction o
        Mode mode;         // RECT or POL
```

```
        friend std::ostream &
              operator<<(std::ostream & o
    };

}   // end namespace VECTOR
#endif
```

Notice that the four functions in Listing 1
defined in the class declaration. This automati
that these functions are so short makes them
them should alter object data, so they are decl
recall from Chapter 10, this is the syntax for d
object it implicitly accesses.

Listing 11.14 shows all the methods and fr
The listing uses the open nature of namespace
`VECTOR` namespace. Note how the constructo
set both the rectangular and the polar represe
ues is available immediately without further c
mentioned in Chapter 4, "Compound Types,"
tions use angles in radians, so the functions bu
methods. The `Vector` class implementation hi
coordinates to rectangular coordinates or con

```
void Vector::set_mag()
{
    mag = sqrt(x * x + y * y);
}

void Vector::set_ang()
{
    if (x == 0.0 && y == 0.0)
        ang = 0.0;
    else
        ang = atan2(y, x);
}

// set x from polar coordinate
void Vector::set_x()
{
    x = mag * cos(ang);
}

// set y from polar coordinate
void Vector::set_y()
{
    y = mag * sin(ang);
}
```

```
            set_x();
            set_y();
        }
        else
        {
            cout << "Incorrect 3rd argume
            cout << "vector set to 0\n";
            x = y = mag = ang = 0.0;
            mode = RECT;
        }
    }

    // reset vector from rectangular coord
    // RECT (the default) or else from pol
    // form is POL
    void Vector:: reset(double n1, double
    {
        mode = form;
        if (form == RECT)
         {
            x = n1;
            y = n2;
            set_mag();
```

```
{
    mode = POL;
}

void Vector::rect_mode()     // set t
{
    mode = RECT;
}

// operator overloading
// add two Vectors
Vector Vector::operator+(const Vector
{
    return Vector(x + b.x, y + b.y);
}

// subtract Vector b from a
Vector Vector::operator-(const Vector
{
    return Vector(x - b.x, y - b.y);
}

// reverse sign of Vector
```

```
            os << "(x,y) = (" << v.x << "
        else if (v.mode == Vector::POL)
        {
            os << "(m,a) = (" << v.mag <<
                << v.ang * Rad_to_deg <<
        }
        else
            os << "Vector object mode is
        return os;
    }

}  // end namespace VECTOR
```

You could design the class differently. For e
gular coordinates and not the polar coordinat
coordinates could be moved to the `magval()`
which conversions are seldom used, this coul
`reset()` method isn't really needed. Suppose
the following code:

```
shove.reset(100,300);
```

means, look at the code for the constructor:

```
Vector::Vector(double n1, double n2, Mode
{
    mode = form;
    if (form == RECT)
    {
         x = n1;
         y = n2;
         set_mag();
         set_ang();
    }
    else if (form == POL)
    {
         mag = n1;
         ang = n2 / Rad_to_deg;
         set_x();
         set_y();
    }
    else
    {
         cout << "Incorrect 3rd argument
         cout << "vector set to 0\n";
```

enum type:

```
Vector rector(20.0, 30.0, 2);  // type mis
```

Still, the resourceful and curious user coul
what happens:

```
Vector rector(20.0, 30.0, VECTOR::Vector::
```

In this case, he gets admonished.

Next, the `operator<<()` function uses the

```
// display rectangular coordinates if mod
// else display polar coordinates if mode
std::ostream & operator<<(std::ostream & o
{
    if (v.mode == Vector::RECT)
         os << "(x,y) = (" << v.x << ", "
    else if (v.mode == Vector::POL)
    {
         os << "(m,a) = (" << v.mag << ",
             << v.ang * Rad_to_deg << ")"
    }
    else
         os << "Vector object mode is inva
    return os;
}
```

Overloading Arithmetic Operators

Adding two vectors is very simple when you components to get the x component of the get the y component of the answer. From th this code:

```
Vector Vector::operator+(const Vector & b
{
    Vector sum;
    sum.x = x + b.x;
    sum.y = y + b.y;
    return sum;          // incomplete ve
}
```

And this would be fine if the object stored nately, this version of the code fails to set the adding more code:

```
Vector Vector::operator+(const Vector & b
{
    Vector sum;
    sum.x = x + b.x;
    sum.y = y + b.y;
    sum.set_ang(sum.x, sum.y);
```

In visual terms, multiplying a vector by a num
that factor. So multiplying a vector by 3 produ
still pointed in the same direction. It's easy to
class represents a vector. In polar terms, you m
alone. In rectangular terms, you multiply a vec
components separately by the number. That is
multiplying by 3 makes the components 15 ar
tiplication operator does:

```
Vector Vector::operator*(double n) const
{
    return Vector(n * x, n * y);
}
```

As with overloaded addition, the code lets
object from the new x and y components. Thi
`double` value. Just as in the `Time` example, you
`double` times `Vector`:

```
Vector operator*(double n, const Vector &
{
    return a * n;   // convert double time
}
```

implicit vector argument, so you should use x

Next, consider the unary minus operator, operator to a regular number, as in -x, change operator to a vector reverses the sign of each should return a new vector that is the reverse leaves the magnitude unchanged but reverses no mathematical training nonetheless have ar are the prototype and definition for overload

```
Vector operator-() const;
Vector Vector::operator-() const
{
    return Vector (-x, -y);
}
```

Note that now there are two separate defi the two definitions have different signatures. Y sions of the - operator because C++ provide operator to begin with. An operator that has only be overloaded as a binary operator.

look-up is faster. If an application often needs
the implementation used in this example wou
only infrequently, the other implementation w
implementation in one program and the secor
same user interface for both.

Taking the `Vector` Class on a Ra

Listing 11.15 provides a short program that us
famous Drunkard's Walk problem. Actually, no
a serious health problem rather than as a sourc
dom Walk problem. The idea is that you place
walking, but the direction of each step varies
ing step. One way of phrasing the problem is
dom walker to travel, say, 50 feet away from th
adding a bunch of randomly oriented vectors

Listing 11.15 lets you select the target dista
wanderer's step. It maintains a running total th
(represented as a vector) and reports the numb
tance, along with the walker's location (in bot
progress is quite inefficient. A journey of 1,00
walker only 50 feet from the starting point. Th
traveled (50 feet, in this case) by the number c

```
double dstep;
cout << "Enter target distance (q to
while (cin >> target)
{
    cout << "Enter step length: ";
    if (!(cin >> dstep))
        break;

    while (result.magval() < target)
    {
        direction = rand() % 360;
        step.reset(dstep, direction,
        result = result + step;
        steps++;
    }
    cout << "After " << steps << " st
        "has the following location:\
    cout << result << endl;
    result.polar_mode();
    cout << " or\n" << result << endl
    cout << "Average outward distance
        << result.magval()/steps << e
    steps = 0;
    result.reset(0.0, 0.0);
```

```
(x,y) = (-21.9577, 45.3019)
 or
(m,a) = (50.3429, 115.8593)
Average outward distance per step = 0.0529
Enter target distance (q to quit): 50
Enter step length: 1
After 1716 steps, the subject has the foll
(x,y) = (40.0164, 31.1244)
 or
(m,a) = (50.6956, 37.8755)
Average outward distance per step = 0.0295
Enter target distance (q to quit): q
Bye!
```

The random nature of the process produce
even if the initial conditions are the same. On
quadruples the number of steps needed to cov
gests that, on average, the number of steps (*N*)
of *D* is given by the following equation:

$$N = (D/s)^2$$

This is just an average, but there will be co
example, 1,000 trials of attempting to travel 5
636 steps (close to the theoretical value of 625

the default seed value and initiate a different
uses the return value of `time(0)` to set the se
calendar time, often implemented as the num
(More generally, `time()` takes the address of a
that variable and also returns it. Using `0` for th
otherwise unneeded `time_t` variable.) Thus, t
each time you run the program, making the r

```
srand(time(0));
```

The `cstdlib` header file (formerly `stdlib`
`rand()`, whereas `ctime` (formerly `time.h`) co
vides more extensive random number suppor
header file.)

The program uses the `result` vector to ke
cycle of the inner loop, the program sets the
the current `result` vector. When the magnitu
loop terminates.

By setting the vector mode, the program d
and in polar terms.

Incidentally, the following statement has th
mode, regardless of the initial modes of `resul`

```
result = result + step;
```

Automatic Conversions a
Classes

The next topic on the class menu is type conv
conversions to and from user-defined types. To
handles conversions for its built-in types. Whe
of one standard type to a variable of another s
the value to the same type as the receiving var
patible. For example, the following statements

```
long count = 8;      // int value 8 conver
double time = 11;    // int value 11 conve
int side = 3.33;     // double value 3.33
```

These assignments work because C++ reco
represent the same basic thing—a number—a
for making the conversions. Recall from Chap
you can lose some precision in these conversio
variable `side` results in `side` getting the value

The C++ language does not automatically
example, the following statement fails because
right side is a number:

```
int * p = 10;  // type clash
```

```
class Stonewt
{
private:
    enum {Lbs_per_stn = 14};      // poun
    int stone;                    // whol
    double pds_left;              // frac
    double pounds;                // enti
public:
    Stonewt(double lbs);          // cons
    Stonewt(int stn, double lbs); // cons
    Stonewt();                    // defa
    ~Stonewt();
    void show_lbs() const;        // show
    void show_stn() const;        // show
};
#endif
```

As mentioned in Chapter 10, `enum` provid
constants, provided that they are integers. Or

```
static const int Lbs_per_stn = 14;
```

```
// construct Stonewt object from double va
Stonewt::Stonewt(double lbs)
{
    stone = int (lbs) / Lbs_per_stn;    //
    pds_left = int (lbs) % Lbs_per_stn + l
    pounds = lbs;
}

// construct Stonewt object from stone, do
Stonewt::Stonewt(int stn, double lbs)
{
    stone = stn;
    pds_left = lbs;
    pounds =  stn * Lbs_per_stn +lbs;
}

Stonewt::Stonewt()          // default con
{
    stone = pounds = pds_left = 0;
}

Stonewt::~Stonewt()         // destructor
{
}
```

The program uses the `Stonewt(double)` c
`Stonewt` object, using `19.6` as the initializatio
copies the contents of the temporary object i
conversion because it happens automatically, w

Only a constructor that can be used with j
tion. The following constructor has two argum

```
Stonewt(int stn, double lbs);  // not a c
```

However, it would act as a guide to `int` co
second parameter:

```
Stonewt(int stn, double lbs = 0);  // int
```

Having a constructor work as an automati
feature. As programmers acquired more exper
found that the automatic aspect isn't always d
conversions. So C++ added a new keyword,
That is, you can declare the constructor this v

```
explicit Stonewt(double lbs);   // no imp
```

This turns off implicit conversions such as
explicit conversions—that is, conversions usin

Let's look at the last point in more detail. T
function prototyping lets the Stonewt(double
numerical types. That is, both of the following
double and then using the Stonewt(double)

```
Stonewt Jumbo(7000);   // uses Stonewt(dou
Jumbo = 7300;          // uses Stonewt(dou
```

However, this two-step conversion process
choice. That is, if the class also defined a Stone
reject these statements, probably pointing out
long or a double, so the call is ambiguous.

Listing 11.18 uses the class constructors to
dle type conversions. Be sure to compile Listi

Listing 11.18 **stone.cpp**

```
// stone.cpp -- user-defined conversions
// compile with stonewt.cpp
#include <iostream>
using std::cout;
#include "stonewt.h"
void display(const Stonewt & st, int n);
int main()
{
```

```
void display(const Stonewt & st, int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << "Wow! ";
        st.show_stn();
    }
}
```

Here is the output of the program in Listi

```
The celebrity weighed 19 stone, 9 pounds
The detective weighed 20 stone, 5.7 pound
The President weighed 302 pounds
After dinner, the celebrity weighed 19 st
After dinner, the President weighed 325 p
Wow! 23 stone, 3 pounds
Wow! 23 stone, 3 pounds
The wrestler weighed even more.
Wow! 30 stone, 2 pounds
Wow! 30 stone, 2 pounds
No stone left unearned
```

double and then uses Stonewt(double) to se

Finally, note the following function call:

```
display(422, 2);    // convert 422 to doub
```

The prototype for display() indicates tha object. (Either a Stonewt or a Stonewt & forr ment.) Confronted with an int argument, the structor to convert the int to the desired Sto the compiler looks for a constructor with son be converted. The Stonewt(double) construc int to double and then uses Stonewt(double

Conversion Functions

Listing 11.18 converts a number to a Stonewt you convert a Stonewt object to a double val

```
Stonewt wolfe(285.7);
double host = wolfe;  // ?? possible ??
```

The answer is that you can do this—but no provide for converting another type *to* the clas special form of a C++ operator function calle

- The conversion function must not spec
- The conversion function must have no

For example, a function to convert to type

```
operator double();
```

The *typeName* part (in this case *typeName*
which to convert, so no return type is needed
means it has to be invoked by a particular cla
value to convert. Thus, the function doesn't n

To add functions that convert `stone_wt` o
requires adding the following prototypes to th

```
operator int();
operator double();
```

Listing 11.19 shows the modified class dec

Listing 11.19 **stonewt1.h**

```
// stonewt1.h -- revised definition for t
#ifndef STONEWT1_H_
#define STONEWT1_H_
class Stonewt
{
```

0.5 is 114.9, and int (114.9) is 114. But if
and int (115.1) is 115.

Listing 11.20 **stonewt1.cpp**

```
// stonewt1.cpp -- Stonewt class methods +
#include <iostream>
using std::cout;
#include "stonewt1.h"

// construct Stonewt object from double va
Stonewt::Stonewt(double lbs)
{
    stone = int (lbs) / Lbs_per_stn;    //
    pds_left = int (lbs) % Lbs_per_stn + l
    pounds = lbs;
}

// construct Stonewt object from stone, do
Stonewt::Stonewt(int stn, double lbs)
{
    stone = stn;
    pds_left = lbs;
    pounds =  stn * Lbs_per_stn +lbs;
}
```

```
{
    return int (pounds + 0.5);
}

Stonewt::operator double()const
{
    return pounds;
}
```

Listing 11.21 tests the new conversion fun
gram uses an implicit conversion, whereas th
cast. Be sure to compile Listing 11.20 along w

Listing 11.21 **stone1.cpp**

```
// stone1.cpp -- user-defined conversion
// compile with stonewt1.cpp
#include <iostream>
#include "stonewt1.h"

int main()
{
```

```
double p_wt = poppins;
```

The answer is no. In the `p_wt` example, the
converted to type `double`. But in the `cout` ex
version should be to `int` or to `double`. Facing
would complain that you were using an ambi
indicates what type to use.

Interestingly, if the class defined only the d
would accept the statement. That's because wi
ambiguity.

You can have a similar situation with assign
the compiler rejects the following statement a

```
long gone = poppins;   // ambiguous
```

In C++, you can assign both `int` and `doub`
legitimately can use either conversion functio
bility of choosing which. But if you eliminate
compiler accepts the statement. For example, s
Then the compiler will use the `int` conversio
Then it converts the `int` value to type `long` v

When the class defines two or more conver
indicate which conversion function to use. You

implicit conversions. In C++98, the keyword
functions, but C++11 removes that limitation
sion operator as explicit:

```
class Stonewt
{
...
// conversion functions
    explicit operator int() const;
    explicit operator double() const;
};
```

With these declarations in place, you woul

Another approach is to replace a conversio
that does the same task—but only if called ex

```
Stonewt::operator int() { return int (pou
```

with

```
int Stonewt::Stone_to_Int() { return int
```

This disallows the following:

```
int plb = poppins;
```

Conversions and Friends

Let's bring addition to the `Stonewt` class. As m
class, you can use either a member function o
simplify matters, assume that no conversion fu
are defined.) You can implement addition wit

```
Stonewt Stonewt::operator+(const Stonewt &
{
    double pds = pounds + st.pounds;
    Stonewt sum(pds);
    return sum;
}
```

Or you can implement addition as a friend

```
Stonewt operator+(const Stonewt & st1, con
{
    double pds = st1.pounds + st2.pounds;
    Stonewt sum(pds);
    return sum;
}
```

Remember, you can provide the method o
both. Either form lets you do the following:

or else

```
total = operator+(jennySt, bennySt);  //
```

In either case, the actual argument types n
ber function is invoked, as required, by a Sto
Next,

```
total = jennySt + kennyD;
```

becomes

```
total = jennySt.operator+(kennyD);   // m
```

or else

```
total = operator+(jennySt, kennyD);  // f
```

Again, the member function is invoked, as
each case, one argument (`kennyD`) is type dou
constructor to convert the argument to a Sto

By the way, having an `operator double(`
confusion at this point because that would cr
Instead of converting `kennyD` to `double` and
could convert `jennySt` to `double` and perfor
version functions creates ambiguities.

choices. The first, as you just saw, is to define t
the Stonewt(double) constructor handle con
Stonewt arguments:

```
operator+(const Stonewt &, const Stonewt &
```

The second choice is to further overload th
explicitly use one type double argument:

```
Stonewt operator+(double x);  // member fu
friend Stonewt operator+(double x, Stonewt
```

That way, the following statement exactly m
ber function:

```
total = jennySt + kennyD; // Stonewt + dou
```

And the following statement exactly match
s) friend function:

```
total = pennyD + jennySt; // double + Ston
```

Earlier, we did something similar for Vecto

Each choice has advantages. The first choic
a shorter program because you define fewer fu
and fewer chances to mess up. The disadvanta
ory needed to invoke the conversion construc

operator *op* has this form:

```
operatorop(argument-list)
```

argument-list represents operands for th
member function, then the first operand is th
argument-list. For example, this chapter ov
`operator+()` member function for the `Vect`
vectors, you can use either of the following s

```
result = up.operator+(right);
result = up + right;
```

For the second version, the fact that the o
C++ to use the `Vector` definition of additio

When an operator function is a member f
invoking the function. In the preceding state
invoking object. If you want to define an ope
a class object, you must use a friend function
function definition in whichever order you v

One of the most common tasks for opera
that it can be used in conjunction with the c
allow an `ostream` object to be the first opera

```
bean = String("pinto");   // converts type
```

To convert from a class to another type, y
provide instruction about how to make the c
a member function. If it is to convert to type
prototype:

```
operator typeName();
```

Note that it must have no declared return
(despite having no declared return type) retur
tion to convert type `Vector` to type `double` v

```
Vector::operator double()
{
    ...
    return a_double_value;
}
```

Experience has shown that often it is bette
functions.

As you might have noticed, classes require
do simple C-style structures. In return, they d

into a file. Label each position with the
the initial conditions (target distance an
the file. The file contents might look li

```
Target Distance: 100, Step Size: 20
0: (x,y) = (0, 0)
1: (x,y) = (-11.4715, 16.383)
2: (x,y) = (-8.68807, -3.42232)
...
26: (x,y) = (42.2919, -78.2594)
27: (x,y) = (58.6749, -89.7309)
After 27 steps, the subject has the
(x,y) = (58.6749, -89.7309)
 or
(m,a) = (107.212, -56.8194)
Average outward distance per step =
```

2. Modify the `Vector` class header and in
so that the magnitude and angle are no
they should be calculated on demand w
are called. You should leave the public i
ods with the same arguments) but alter
private method and the method implem

or equal to 11 stone. (The simplest appr
ized to 11 stone and to compare the ot

7. A complex number has two parts: a rea
write an imaginary number is this: (3.0
the imaginary part. Suppose a = (A,Bi)
operations:

   - Addition: a + c = (A + C, (B + I
   - Subtraction: a - c = (A - C, (B - I
   - Multiplication: a   c = (A   C - I
   - Multiplication: (x a real number):
   - Conjugation: ~a = (A, - Bi)

   Define a complex class so that the follo
   results:

```
#include <iostream>
using namespace std;
#include "complex0.h"  // to avoid c
int main()
{
    complex a(3.0, 4.0);   // initia
```

```
Enter a complex number (q to quit):
real: 10
imaginary: 12
c is (10,12i)
complex conjugate is (10,-12i)
a is (3,4i)
a + c is (13,16i)
a - c is (-7,-8i)
a * c is (-18,76i)
2 * c is (20,24i)
Enter a complex number (q to quit):
real: q
Done!
```

Note that `cin >> c`, through overload

parts.

- What you must do if you use new in a
- Using `static` class members
- Using placement `new` with objects
- Using pointers to objects
- Implementing a queue abstract data ty

This chapter looks at how to use `new` and some of the subtle problems that using dyna short list of topics, but these topics affect con tor overloading.

Let's look at a specific example of how C you want to create a class with a member tha plest, most primitive way is to use a characte has some drawbacks. You might use a 14-cha Smeadsbury-Crafthovingham. Or to be safer you then create an array of 2,000 such objec ter arrays that are only partly filled. (At that p ory load.) There is an alternative.

Often it is much better to decide many m when a program runs rather than when it's c a name in an object is to use the `new` operato amount of memory while the program is ru

overload an assignment operator to get a prog
these matters now.

A Review Example and Static Cla

We haven't used new and delete for a while,
While we're at it, let's look at a new storage c
be a StringBad class, later to be superseded b
(You've already seen the standard C++ strin
Chapter 16, "The string Class and the Stand
ble StringBad and String classes in this chap
lies such a class. A lot of programming techni
interface.)

StringBad and String class objects will h
senting the string length. We'll use the Strin
inside look at how new, delete, and static c
constructors and destructors will display mess
action. Also we'll omit several useful member
and >> operators and a conversion function, i
rejoice! The review questions for this chapter
ful support functions.) Listing 12.1 shows the

under development. It's the first stage of deve
allocation, and it does the obvious things cor
correctly in the constructors and destructor.
omits doing some additional good things tha
the problems the class has should help you u
changes you will make later, when you conve

You should note two points about this de
instead of a `char` array to represent a name. T
allocate storage space for the string itself. Ins
cate space for the string. This arrangement av
a predefined limit to the string size.

Second, the definition declares the `num_s`
storage class. A *static class member* has a special
of a static class variable, regardless of the num
ber is shared among all objects of that class, n
among all members of a family. If, say, you cr
10 `str` members and 10 `len` members, but ju
Figure 12.1). This is convenient for data that
have the same value for all class objects. The
intended to keep track of the number of obj

Figure 12.1 A st

By the way, Listing 12.1 uses the num_stri
trating static data members and as a device to
In general, a string class doesn't need such a n

Take a look at the implementation of the cl
handles using a pointer and a static member.

Listing 12.2 **strngbad.cpp**

```
// strngbad.cpp -- StringBad class methods
#include <cstring>                    // s
#include "strngbad.h"
using std::cout;

// initializing static class member
int StringBad::num_strings = 0;

// class methods
```

```
    cout << "\"" << str << "\" object del
    --num_strings;                    //
    cout << num_strings << " left\n"; //
    delete [] str;                    //
}

std::ostream & operator<<(std::ostream &
{
    os << st.str;
    return os;
}
```

First, notice the following statement from

```
int StringBad::num_strings = 0;
```

This statement initializes the static `num_st` initialize a static member variable inside the tion is a description of how memory is to be allocate and initialize memory by creating an static class member, you initialize the static m ment outside the class declaration. That's bec rately rather than as part of an object. Note t and uses the scope operator, but it doesn't us with a regular C string:

```
StringBad::StringBad(const char * s)
{
    len = std::strlen(s);           // set
    str = new char[len + 1];        // allo
    std::strcpy(str, s);            // init
    num_strings++;                  // set
    cout << num_strings << ": \"" << str
         << "\" object created\n"; // For
}
```

Recall that the class `str` member is just a p
memory for holding a string. You can pass a st
initialize an object:

```
String boston("Boston");
```

The constructor must then allocate enough
must copy the string to that location. Let's go

First, the function initializes the `len` memb
the length of the string. Next, it uses `new` to al
then it assigns the address of the new memory
returns the length of a string, not counting th
structor adds one to `len` to allow space for th

}

The destructor begins by announcing wh
informative but not essential. However, the d
member points to memory allocated with ne
pointer expires. But the memory `str` pointed
to free it. Deleting an object frees the memor
not automatically free memory pointed to by
that, you must use the destructor. By placing
ensure that the memory that a constructor al
expires.

Warning

Whenever you use `new` in a constructor to al
corresponding destructor to free that memor
should use `delete []` (with brackets).

Listing 12.3, which is taken from a progra
illustrates when and how the `StringBad` con
places the object declarations within an inne
execution exits the block in which an object
destructors would be called after execution e
some environments from seeing the destructo
closes. Be sure to compile Listing 12.2 along

```
        cout << "headline1: " << headline1
        callme2(headline2);
        cout << "headline2: " << headline2
        cout << "Initialize one object to
        StringBad sailor = sports;
        cout << "sailor: " << sailor << en
        cout << "Assign one object to anot
        StringBad knot;
        knot = headline1;
        cout << "knot: " << knot << endl;
        cout << "Exiting the block.\n";
    }
    cout << "End of main()\n";

    return 0;
}

void callme1(StringBad & rsb)
{
    cout << "String passed by reference:\n
    cout << "    \"" << rsb << "\"\n";
}

void callme2(StringBad sb)
```

```
headline1: Celery Stalks at Midnight
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 2 left
headline2: Dû°
Initialize one object to another:
sailor: Spinach Leaves Bowl for Dollars
Assign one object to another:
3: "C++" default object created
knot: Celery Stalks at Midnight
Exiting the block.
"Celery Stalks at Midnight" object delete
"Spinach Leaves Bowl for Dollars" object
"Spinach Leaves Bowl for Doll8" object de
"@g" object deleted, -1 left
"-|" object deleted, -2 left
End of main()
```

The various nonstandard characters that a
system; they are one of the signs that `String`
the negative object count. Newer compiler/o
the program just before displaying the line ab
report a General Protection Fault (GPF). A G
memory location forbidden to it; this is anot

This section of code seems to work fine, to
But then the program executes the followi

```
callme2(headline2);
cout << "headline2: " << headline2 << endl
```

Here, `callme2()` passes `headline2` by valu
cates a serious problem:

```
String passed by value:
    "Lettuce Prey"
"Lettuce Prey" object deleted, 2 left
headline2: Dû°
```

First, passing `headline2` as a function argu
called. Second, although passing by value is su
from change, the function messes up the origi
nonstandard characters get displayed. (The exa
pens to sitting in memory.)

Even worse, look at the end of the output
cally for each of the objects created earlier:

```
Exiting the block.
"Celery Stalks at Midnight" object deleted
"Spinach Leaves Bowl for Dollars" object d
"Spinach Leaves Bowl for Doll8" object del
```

tax for the following:

```
StringBad sailor = StringBad(sports); //c
```

Because `sports` is type `StringBad`, a matc

```
StringBad(const StringBad &);
```

And it turns out that the compiler auton *copy constructor* because it makes a copy of ar another. The automatic version would not k variable, so it would mess up the counting s by this example stem from member functior so let's look at that topic now.

Special Member Functions

The problems with the `StringBad` class stem member functions that are defined automatic of these member functions is inappropriate t C++ automatically provides the following m

- A default constructor if you define no
- A default destructor if you don't define
- A copy constructor if you don't define

compiler supplies the following default:

```
Klunk::Klunk() { }  // implicit default co
```

That is, it supplies a constructor (the *defau*
ments and that does nothing. It's needed beca
constructor:

```
Klunk lunk;  // invokes default constructo
```

The default constructor makes `lunk` like a
value at initialization is unknown.

After you define any constructor, C++ do
If you want to create objects that aren't initial
default constructor explicitly. It's a constructo
set particular values:

```
Klunk::Klunk()  // explicit default constr
{
    klunk_ct = 0;
    ...
}
```

A constructor with arguments still can be a
default values. For example, the `Klunk` class co

```
Klunk(int n = 0) { klunk_ct = n; }
```

You must know two things about a copy

When a Copy Constructor Is Used

A copy constructor is invoked whenever a ne
existing object of the same kind. This happen
ation is when you explicitly initialize a new o
given that `motto` is a `StringBad` object, the f
copy constructor:

```
StringBad ditto(motto);  // calls StringB
StringBad metoo = motto; // calls StringB
StringBad also = StringBad(motto);
                         // calls StringB
StringBad * pStringBad = new StringBad(mo
                         // calls StringB
```

Depending on the implementation, the m
structor directly to create `metoo` and `also`, or
temporary objects whose contents are then a
initializes an anonymous object to `motto` and
`pstring` pointer.

Less obviously, a compiler uses a copy con
copies of an object. In particular, it's used wh
`callme2()` does in Listing 12.3) or when a fu

private members is not allowed):

```
StringBad sailor;
sailor.str = sports.str;
sailor.len = sports.len;
```

If a member is itself a class object, the copy
one member object to another. Static member
because they belong to the class as a whole in
illustrates the action of an implicit copy const

Back to Stringbad: Where the Co

You are now in a position to understand the t
assume that the output is the one shown after
program output indicates two more objects d
that the program does create two additional o
The copy constructor is used to initialize the
function is called, and it is used to initialize th
constructor doesn't vocalize its activities, so it
increment the `num_strings` counter. Howeve
it's invoked upon the demise of all objects, reg
weirdness is a problem because it means the p
count. The solution is to provide an explicit c

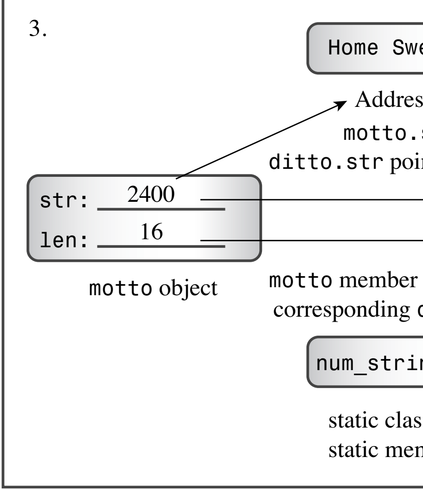


Figure 12.2 An inside lo

Tip

If your class has a static data member whos
you should provide an explicit copy construct

the destructor for sailor, and this results in u
case of Listing 12.3, the program produces ma
memory mismanagement.

Another disturbing symptom is that attem
cause the program to abort. Microsoft Visual C
plays an error message window saying "Debug
reports "double free or corruption" and abort
messages or even no message, but the same ev

Fixing the Problem by Defining an Explic

The cure for the problems in the class design
just copy the address of the string, the copy co
assign the address of the duplicate to the str
string rather than referring to another object's
a different string rather than making duplicate
how you can code the String copy construct

```
StringBad::StringBad(const StringBad & st)
{
    num_strings++;              // handle s
    len = st.len;               // same len
    str = new char [len + 1];   // allot sp
    std::strcpy(str, st.str);   // copy str
```

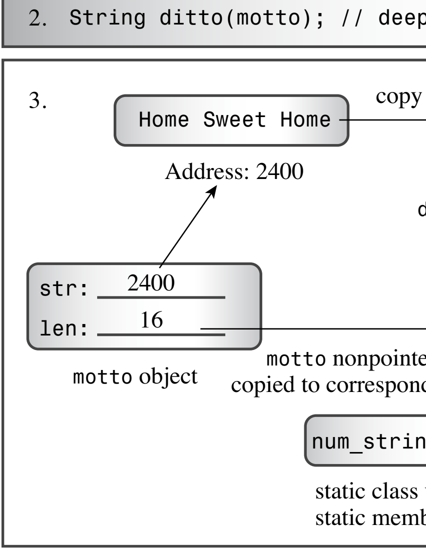


Figure 12.3 An insic

When an Assignment Operator Is Used a

An overloaded assignment operator is used wh
ing object:

```
StringBad headline1("Celery Stalks at Midn
...
StringBad knot;
knot = headline1;   // assignment operator
```

An assignment operator is not necessarily u

```
StringBad metoo = knot; // use copy constr
```

Here `metoo` is a newly created object being
constructor is used. However, as mentioned be
handling this statement in two steps: using the
object and then using assignment to copy the
tion always invokes a copy constructor, and fo
assignment operator.

Like a copy constructor, an implicit implem
forms a member-to-member copy. If a memb
gram uses the assignment operator defined fo
particular member. Static data members are u

frivolous or destructive behavior.

Fixing Assignment

The solution for the problems created by an i
to provide your own assignment operator def
implementation is similar to that of the copy

- Because the target object may already r
should use `delete []` to free former o
- The function should protect against ass
ing of memory described previously co
are reassigned.
- The function returns a reference to the

By returning an object, the function can e
built-in types can be chained. That is, if `S0`, `S1`
write the following:

```
S0 = S1 = S2;
```

In function notation, this becomes the foll

```
S0.operator=(S1.operator=(S2));
```

don't first apply the delete operator, the prev
the program no longer has a pointer to the ol

Next, the program proceeds like a copy co
new string and then copying the string from t

When it is finished, the program returns *t

Assignment does not create a new object, s
static data member num_strings.

Adding the copy constructor and the assig
StringBad class clears up all the problems. He
put after these changes have been made:

```
End of main()
"Celery Stalks at Midnight" object deleted
"Spinach Leaves Bowl for Dollars" object d
"Spinach Leaves Bowl for Dollars" object d
"Lettuce Prey" object deleted, 1 left
"Celery Stalks at Midnight" object deleted
```

The object counting is correct now, and no

```
static int HowMany();
```

The first new method returns the length o functions allow you to compare strings. The c capabilities. The two `operator[]()` function characters in a string. The static class method member `num_strings`. Let's look at some det

The Revised Default Constructor

The new default constructor merits notice. It

```
String::String()
{
    len = 0;
    str = new char[1];
    str[0] = '\0';              // defau
}
```

You might wonder why the code uses

```
str = new char[1];
```

and not this:

```
str = new char;
```

C++11 Null Pointer

In C++98, the literal `0` has two meanings—it
null pointer—thus making it difficult for the re
two. Sometimes programmers use `(void *)`
pointer itself may have a nonzero internal rep
C macro defined to represent the null pointer.
tion. C++11 provides a better solution by intr
the null pointer. You still can use `0` as before
code would be invalidated—but henceforth th

```
str = nullptr;  // C++11 null pointer n
```

Comparison Members

Three of the methods in the `String` class perf
tion returns `true` if the first string comes befo
precisely, in the machine collating sequence). T
comparison functions is to use the standard `st`
value if its first argument precedes the second
and a positive value if the first follows the seco
like this:

```
bool operator<(const String &st1, const St
{
    if (std::strcmp(st1.str, st2.str) < 0)
```

good choice for an inline function.

Making the comparison functions friends objects and regular C strings. For example, su you have the following code:

```
if ("love" == answer)
```

This gets translated to the following:

```
if (operator==("love", answer))
```

The compiler then uses one of the constru

```
if (operator==(String("love"), answer))
```

And this matches the prototype.

Accessing Characters by Using B

With a standard C-style string, you can use b

```
char city[40] = "Amsterdam";
cout << city[0] << endl; // display the l
```

In C++ the two bracket symbols constitu you can overload this operator by using a me C++ operator (one with two operands) puts `2 + 5`. But the bracket operator places one o

```
cout << opera[4];
```

becomes this:

```
cout << opera.operator[4];
```

The return value is `opera.str[4]`, or the c
access to private data.

Declaring the return type as type `char &` a
ment. For example, you can use the following

```
String means("might");
means[0] = 'r';
```

The second statement is converted to an o

```
means.operator[][0] = 'r';
```

This assigns `'r'` to the method's return val
`means.str[0]`, making the code equivalent to

```
means.str[0] = 'r';
```

This last line of code violates private access
method, it is allowed to alter the array conten
becomes `"right"`.

```
cout << text[1];    // ok, uses non-const
cout << answer[1];  // ok, uses const ver
cin >> text[1];     // ok, uses non-const
cin >> answer[1];   // compile-time error
```

Static Class Member Functions

It's possible to declare a member function as
appear in the function declaration but not in
rate.) This has two important consequences.

First, a static member function doesn't hav
doesn't even get a `this` pointer to play with.
the public section, it can be invoked using th
tor. For instance, you can give the `String` cla
with the following prototype/definition in th

```
static int HowMany() { return num_strings
```

It could be invoked like this:

```
int count = String::HowMany();  // invoki
```

The second consequence is that because a
with a particular object, the only data membe

serves as a conversion function.

2. In Listing 12.6, later in this chapter, the
ator=(const String &) function to c
to the name object.

3. The program calls the ~String() destru

The simplest way to make the process mor
operator so that it works directly with ordinar
creating and destroying a temporary object. H

```
String & String::operator=(const char * s)
{
    delete [] str;
    len = std::strlen(s);
    str = new char[len + 1];
    std::strcpy(str, s);
    return *this;
}
```

As usual, you must deallocate memory form
memory for the new string.

Listing 12.4 shows the revised class declara
mentioned, it defines the constant CINLIM, wh

```
    int length () const { return len; }
// overloaded operator methods
    String & operator=(const String &);
    String & operator=(const char *);
    char & operator[](int i);
    const char & operator[](int i) const;
// overloaded operator friends
    friend bool operator<(const String &s
    friend bool operator>(const String &s
    friend bool operator==(const String &
    friend ostream & operator<<(ostream &
    friend istream & operator>>(istream &
// static function
    static int HowMany();
};
#endif
```

Listing 12.5 presents the revised method d

Listing 12.5 **string1.cpp**

```
// string1.cpp -- String class methods
#include <cstring>                    // str
#include "string1.h"                  // inc
```

```
{
    len = 4;
    str = new char[1];
    str[0] = '\0';                    // defa
    num_strings++;
}

String::String(const String & st)
{
    num_strings++;              // handle s
    len = st.len;               // same len
    str = new char [len + 1];   // allot sp
    std::strcpy(str, st.str);   // copy str
}

String::~String()                     // n
{
    --num_strings;                    // r
    delete [] str;                    // r
}

// overloaded operator methods

    // assign a String to a String
```

```
{
    return str[i];
}

    // read-only char access for const St
const char & String::operator[](int i) co
{
    return str[i];
}

// overloaded operator friends

bool operator<(const String &st1, const S
{
    return (std::strcmp(st1.str, st2.str)
}

bool operator>(const String &st1, const S
{
    return st2 < st1;
}

bool operator==(const String &st1, const
{
```

The overloaded >> operator provides a sim
a String object. It assumes an input line of St
cards any characters beyond that limit. Keep in
in an if condition evaluates to false if input
an end-of-file condition, or in the case of get

Listing 12.6 exercises the String class with
strings. The program has the user enter saying
plays them, and reports which string is the sho

Listing 12.6 **sayings1.cpp**

```
// sayings1.cpp -- using expanded String c
// compile with string1.cpp
#include <iostream>
#include "string1.h"
const int ArSize = 10;
const int MaxLen =81;
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;
    String name;
```

```
        cout << "Here are your sayings:\n
        for (i = 0; i < total; i++)
            cout << sayings[i][0] << ": "

        int shortest = 0;
        int first = 0;
        for (i = 1; i < total; i++)
        {
            if (sayings[i].length() < say
                shortest = i;
            if (sayings[i] < sayings[firs
                first = i;
        }
        cout << "Shortest saying:\n" << s
        cout << "First alphabetically:\n"
        cout << "This program used "<< St
             << " String objects. Bye.\n"
    }
    else
        cout << "No input! Bye.\n";
     return 0;
}
```

```
1: a fool and his money are soon parted
2: penny wise, pound foolish
3: the love of money is the root of much e
4: out of sight, out of mind
5: absence makes the heart grow fonder
6: absinthe makes the hart grow fonder
7:
Here are your sayings:
a: a fool and his money are soon parted
p: penny wise, pound foolish
t: the love of money is the root of much e
o: out of sight, out of mind
a: absence makes the heart grow fonder
a: absinthe makes the hart grow fonder
Shortest saying:
penny wise, pound foolish
First alphabetically:
a fool and his money are soon parted
This program used 11 String objects. Bye.
```

character as a visual reminder that this valu
favored a simple 0 instead of the equivalent
the nullptr keyword as a better alternative

- You should define a copy constructor t deep copying. Typically, the constructor

```
String::String(const String & st)
{
    num_strings++;             // ha
    len = st.len;              // sa
    str = new char [len + 1]; // al
    std::strcpy(str, st.str); // co
}
```

In particular, the copy constructor shou it should copy the data, not just the add static class members whose value would

- You should define an assignment opera doing deep copying. Typically, the class example:

```
}

String::String(const char * s)
{
   len = std::strlen(s);
   str = new char;              // oops, no
   std::strcpy(str, s);         // oops, no
}

String::String(const String & st)
{
       len = st.len;
       str = new char[len + 1];       // go
       std::strcpy(str, st.str);      // go
}
```

The first constructor fails to use `new` to ini default object, applies `delete` to `str`. The resu tialized by `new` is undefined, but it is probably

```
String::String()
{
    len = 0;
    str = new char[1];  // uses new with [
    str[0] = '\0';
}
```

```
{
    delete str;      // oops, should be d
}
```

The destructor uses `delete` incorrectly. B
characters, the destructor should delete an ar

Memberwise Copying for Classes

Suppose you use the `String` class, or, for that
for class members:

```
class Magazine
{
private:
    String title;
    string publisher;
...
};
```

`String` and `string` both use dynamic me
to write a copy constructor and assignment o
not in itself. The default memberwise copyin
smarts. If you copy or assign one `Magazine` o
the copy constructors and assignment operat

in Chapter 11. The function would be used in

```
Vector force1(50,60);
Vector force2(10,70);
Vector max;
max = Max(force1, force2);
```

Either of the following two implementatio

```
// version 1
Vector Max(const Vector & v1, const Vector
{
    if (v1.magval() > v2.magval())
        return v1;
    else
        return v2;
}

// version 2
const Vector & Max(const Vector & v1, cons
{
    if (v1.magval() > v2.magval())
        return v1;
    else
        return v2;
}
```

`operator=()` method returns a reference to s

The return value of `operator<<()` is used

```
String s1("Good stuff");
cout << s1 << "is coming!";
```

Here, the return value of `operator<<(cou` the string `"is coming!"`. Here, the return ty `ostream`. Using an `ostream` return type wou structor, and, as it turns out, the `ostream` clas Fortunately, returning a reference to `cout` po scope in the calling function.

Returning an Object

If the object being returned is local to the ca by reference because the local object has its d nates. Thus, when control returns to the callin the reference can refer. In these circumstance ence. Typically, overloaded arithmetic operato ple, which uses the `Vector` class again:

```
Vector force1(50,60);
Vector force2(10,70);
Vector net;
net = force1 + force2;
```

```
net = force1 + force2;
```

However, the definition also allows you to

```
force1 + force2 = net;
cout << (force1 + force2 = net).magval() <
```

Three questions immediately arise. Why w
they possible? What do they do?

First, there is no sensible reason for writing
sensible reasons. People, even programmers, m
`operator==()` were defined for the `Vector` cl

```
if (force1 + force2 = net)
```

instead of this:

```
if (force1 + force2 == net)
```

Also programmers tend to be ingenious, an
mistakes.

Second, this code is possible because the co
object to represent the return value. So in the
`force2` stands for that temporary object. In St
to `net`. In Statements 2 and 3, `net` is assigned

cally. Another approach is to use pointers to p
gories. Listing 12.7 implements this approach
`shortest` pointer points to the first object in
object with a shorter string, it resets `shortes`
pointer tracks the alphabetically earliest string
new objects; they merely point to existing ob
allocate additional memory.

For variety, the program in Listing 12.7 us
object:

```
String * favorite = new String(sayings[ch
```

Here the pointer `favorite` provides the c
`new`. This particular syntax means to initialize
`sayings[choice]`. That invokes the copy cor
copy constructor (`const String &`) matches
The program uses `srand()`, `rand()`, and `tim`

Listing 12.7 **sayings2.cpp**

```
// sayings2.cpp -- using pointers to obje
// compile with string1.cpp
#include <iostream>
#include <cstdlib>      // (or stdlib.h)
#include <ctime>        // (or time.h) fo
```

```
        else
            sayings[i] = temp;        // ov
    }
    int total = i;                    // to

    if (total > 0)
    {
        cout << "Here are your sayings:\n"
        for (i = 0; i < total; i++)
            cout << sayings[i] << "\n";

    // use pointers to keep track of short
        String * shortest = &sayings[0]; /
        String * first = &sayings[0];
        for (i = 1; i < total; i++)
        {
            if (sayings[i].length() < shor
                shortest = &sayings[i];
            if (sayings[i] < *first)     /
                first = &sayings[i];     /
        }
        cout << "Shortest saying:\n" << *
        cout << "First alphabetically:\n"
        srand(time(0));
```

Also the usual conversions invoked by protot
takes place as long as there is no ambiguity.
the default constructor:

```
Class_name * ptr = new Class_name;
```

Here's a sample run of the program in List

```
Hi, what's your name?
>> Kirt Rood
Kirt Rood, please enter up to 10 short sa
1: a friend in need is a friend indeed
2: neither a borrower nor a lender be
3: a stitch in time saves nine
4: a niche in time saves stine
5: it takes a crook to catch a crook
6: cold hands, warm heart
7:
Here are your sayings:
a friend in need is a friend indeed
neither a borrower nor a lender be
a stitch in time saves nine
a niche in time saves stine
it takes a crook to catch a crook
cold hands, warm heart
```

This allocates space not for the string to be
`str` pointer that holds the address of the strin
cate space for the `num_strings` member beca
rately from the objects.) Creating the object, i
space for storing the string and assigns the stri
`delete` to delete this object when it is finishe
program uses `delete` without brackets. Again,
`str` pointer and the `len` member. It doesn't fr
points to, but the destructor takes care of that

Again, destructors are called in the followi

- If an object is an automatic variable, the
gram exits the block in which the objec
destructor is called for `headlines[0]` a
`main()`, and the destructor for `grub` is c
- If an object is a static variable (external,
its destructor is called when the progra
`sports` object in Listing 12.3.
- If an object is created by `new`, its destruc
`delete` on the object.

- You declare a pointer to an object by u

```
String * glamour;
```

- You can initialize a pointer to point to

```
String * first = &sayings[0];
```

- You can initialize a pointer by using ne

```
String * favorite = new String(sayin
```

  Also see Figure 12.6 for a more detaile with new.

- Using new with a class invokes the appr newly created object:

```
// invokes default constructor
String * gleep = new String;

// invokes the String(const char *)
String * glop = new String("my my my
```

Initializing a pointer using `new` and the `String(const char*)` class constructor:	`String * g`
Initializing a pointer using `new` and the `String(const String &)` class constructor:	`String * f`
Using the -> operator to access a class method via a pointer:	$\underbrace{\texttt{(saying}}_{\text{objec}}$ preceded by `if`
Using the * deferencing operator to obtain an object from a pointer:	$\underbrace{\texttt{(saying}}_{\text{objec}}$ preceded by `if`

Figure 12.5 Poi

Looking Again at Placement new

Recall that placement new allows you to spec
memory. Chapter 9, "Memory Models and N
context of built-in types. Using placement ne
12.8 uses placement new along with regular n
a class with a chatty constructor and destruct
objects.

Listing 12.8 **placenew1.cpp**

```
// placenew1.cpp  -- new, placement new,
#include <iostream>
#include <string>
#include <new>
using namespace std;
const int BUF = 512;
```

```
    cout << pc1 << ": ";
    pc1->Show();
    cout << pc2 << ": ";
    pc2->Show();

    JustTesting *pc3, *pc4;
    pc3 = new (buffer) JustTesting("Bad Id
    pc4 = new JustTesting("Heap2", 10);

    cout << "Memory contents:\n";
    cout << pc3 << ": ";
    pc3->Show();
    cout << pc4 << ": ";
    pc4->Show();

    delete pc2;                         /
    delete pc4;                         /
    delete [] buffer;                   /
    cout << "Done\n";
    return 0;
}
```

There are a couple problems with placem
creating a second object, placement new simp
first object with a new object. Not only is th
called for the first object. This, of course, wou
dynamic memory allocation for its members.

Second, using delete with pc2 and pc4 a
two objects that pc2 and pc4 point to. But us
the destructors for the objects created with p

One lesson to be learned here is the same
you to manage the memory locations in a bu
different locations, you provide two different
the locations don't overlap. You can, for exam

```
pc1 = new (buffer) JustTesting;
pc3 = new (buffer + sizeof (JustTesting))
```

Here the pointer pc3 is offset from pc1 by

The second lesson to be learned here is th
you need to arrange for their destructors to b
the heap, you can use this:

```
delete pc2;   // delete object pointed to
```

of the rare cases that require an explicit call. A
fying the object to be destroyed. Because ther
these pointers:

```
pc3->~JustTesting();  // destroy object pc
pc1->~JustTesting();  // destroy object pc
```

Listing 12.9 fixes Listing 12.8 by managing
and by adding appropriate uses of `delete` and
fact is the proper order of deletion. The objec
destroyed in order opposite that in which they
principle, a later object might have dependenc
to hold the objects should be freed only after

Listing 12.9 **placenew2.cpp**

```
// placenew2.cpp  -- new, placement new, n
#include <iostream>
#include <string>
#include <new>
using namespace std;
const int BUF = 512;

class JustTesting
{
```

```
    cout << pc2 << ": ";
    pc2->Show();

    JustTesting *pc3, *pc4;
// fix placement new location
    pc3 = new (buffer + sizeof (JustTesti
                JustTesting("Better Idea"
    pc4 = new JustTesting("Heap2", 10);

    cout << "Memory contents:\n";
    cout << pc3 << ": ";
    pc3->Show();
    cout << pc4 << ": ";
    pc4->Show();

    delete pc2;             // free Heap1
    delete pc4;             // free Heap2
// explicitly destroy placement new objec
    pc3->~JustTesting();  // destroy obje
    pc1->~JustTesting();  // destroy obje
    delete [] buffer;     // free buffer
    cout << "Done\n";
    return 0;
}
```

Reviewing Techniques

By now, you've encountered several programm
class-related problems, and you may be having
following sections summarize several techniqu

Overloading the << Operator

To redefine the << operator so that you use it
you define a friend operator function that has

```
ostream & operator<<(ostream & os, const c
{
    os << ... ;  // display object content
    return os;
}
```

Here *c_name* represents the name of the cl
return the required contents, you can use thos
pense with the friend status.

does not know them and thus won't catch yo

- Any class member that points to memo operator applied to it in the class destru
- If a destructor frees memory by applyin every constructor for that class should by setting the pointer to the null point
- Constructors should settle on using eit both. The destructor should use delete should use delete if the constructors u
- You should define a copy constructor t ing a pointer to existing memory. This object to another. The constructor shou

  ```
  className(const className &)
  ```

- You should define a class member func and that has a function definition with is a member of the *c_name* class and ha lowing example assumes that the const using new []:

A rather natural way of representing the pr
queue is an abstract data type (ADT) that hol
are added to the rear of the queue, and items
bit like a stack, except that a stack has additio
a stack a LIFO (last in, first out) structure, wh
structure. Conceptually, a queue is like a line
suited to the task. So one part of the project i
you'll read about the Standard Template Libra
developing your own than by just reading abo

The items in the queue will be customers.
that, on average, a third of the customers will
take two minutes, and a third will take three
random intervals, but the average number of
more parts of your project will be to design a
together a program that simulates the interact
Figure 12.7).

A Queue Class

The first order of business is to design a `Queu` the kind of queue you'll need:

- A queue holds an ordered sequence of
- A queue has a limit on the number of
- You should be able to create an empty
- You should be able to check whether a
- You should be able to check whether a
- You should be able to add an item to t
- You should be able to remove an item
- You should be able to determine the n

As usual when designing a class, you need implementation.

The Queue Class Interface

The queue attributes listed in the preceding s for a queue class:

The Queue Class Implementation

After you determine the interface, you can im
represent the queue data. One approach is to
the required number of elements. However, ar
tions. For example, removing an item from th
shifting every remaining element one unit clo
something more elaborate, such as treat the ar
a reasonable fit to the requirements of a queue
nodes. Each *node* contains the information to
node in the list. For the queue in this example
you can use a structure to represent a node:

```
struct Node
{
    Item item;              // data stored
    struct Node * next;     // pointer to
};
```

Figure 12.8 illustrates a linked list.

The example shown in Figure 12.8 is calle
single link, or pointer, to another node. If you
follow the pointers to each subsequent node i
node in the list is set to NULL (or, equivalently,
nodes. With C++11, you should use the new

a queue always adds a new item to the end o
member point to the last node, too (see Figu
bers to keep track of the maximum number
rent number of items. Thus, the private part o

```
class Queue
{
private:
// class scope definitions
    // Node is a nested structure definit
    struct Node { Item item; struct Node
    enum {Q_SIZE = 10};
// private class members
    Node * front;       // pointer to fro
    Node * rear;        // pointer to rea
    int items;          // current number
    const int qsize;    // maximum number
    ...
public:
//...
};
```

The declaration uses the C++ ability to ne
By placing the `Node` declaration inside the `Que`
a type that you can use to declare class membe

type is restricted to the class. That way, you don
`Node` conflicting with some global declaration o
class. Some obsolescent compilers do not suppo
n't, then you have to define a `Node` structure glo

Nested Structures and Classes

A structure, a class, or an enumeration declar
nested in the class. It has class scope. Such
Rather, it specifies a type that can be used in
made in the private section of the class, then
class. If the declaration is made in the public
used out of the class, through use of the sco
were declared in the public section of the `Que`
`Queue::Node` outside the `Queue` class.

After you settle on a data representation, t

The Class Methods

A class constructor should provide values for t
example begins in an empty state, you should

```
Queue::Queue(int qs) : qsize(qs)   // ini
{
    front = rear = NULL;
    items = 0;
}
```

In general, the initial value can involve co
argument list. The technique is not limited to
`Queue` constructor like this:

```
Queue::Queue(int qs) : qsize(qs), front(N
{
}
```

Only constructors can use this initializer-l
this syntax for `const` class members. You also
declared as references:

```
class Agency {...};
class Agent
{
private:
    Agency & belong;    // must use initi
    ...
};
Agent::Agent(Agency & a) : belong(a) {...
```

Data members are initialized in the order in w
in the order in which initializers are listed.

Caution

You can't use the member initializer list synta

The parenthesized form used in the memb
tializations, too. That is, if you like, you can re

```
int games = 162;
double talk = 2.71828;
```

with

```
int games(162);
double talk(2.71828);
```

This allows initializing built-in types to loo

C++11 Member In-Class Initialization

C++11 allows you to do what would seem to

```
class Classy
{
    int mem1 = 10;       // in-class in
    const int mem2 = 20; // in-class in
```

```
    if (isfull())
        return false;
    Node * add = new Node;  // create nod
// on failure, new throws std::bad_alloc
    add->item = item;       // set node p
    add->next = NULL;       // or nullptr
    items++;
    if (front == NULL)      // if queue i
        front = add;        // place item
    else
        rear->next = add;   // else place
    rear = add;             // have rear
    return true;
}
```

In brief, the method goes through the fol

1. Terminate if the queue is already full. (
   is selected by the user via the construct

2. Create a new node. If `new` can't do so, i
   up in Chapter 15, "Friends, Exceptions
   unless one provides additional program
   terminates.

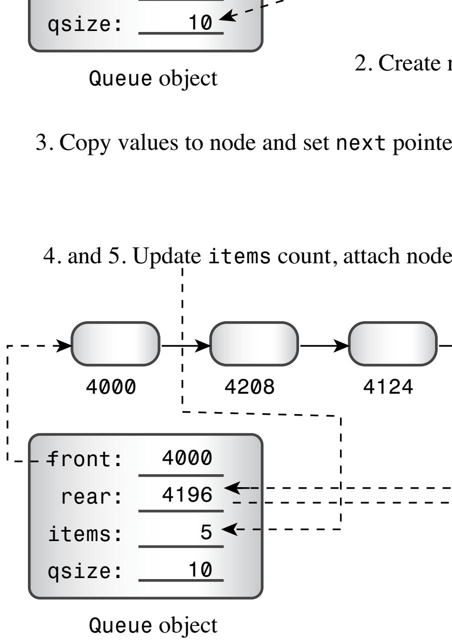


Figure 12.10 E

3. Decrease the item count (`items`) by on

4. Save the location of the front node for

5. Take the node off the queue. This is acc pointer `front` to point to the next nod

6. To conserve memory, delete the forme

7. If the list is now empty, set `rear` to `NUL` in this case, after setting `front->next`.) with C++11, you can use `nullptr`.

Step 4 is necessary because step 5 erases th node is.

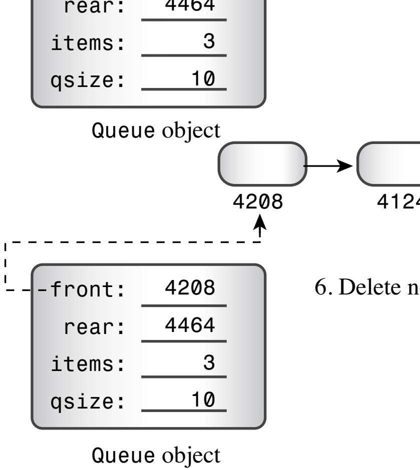


Figure 12.11 D

answer is, "Does the default memberwise cop
Memberwise copying of a `Queue` object wou
front and rear of the same linked list as the or
`Queue` object changes the shared linked list. T
the copy's rear pointer gets updated, essential
the original object. Clearly, then, cloning or c
constructor and an assignment constructor th

Of course, that raises the question of why
haps you would want to save snapshots of a q
Or you would like to provide identical input
be useful to have operations that split a queue
opening an additional checkout stand. Simila
into one or truncate a queue.

But suppose you don't want to do any of
ply ignore those concerns and use the metho
However, at some time in the future, you mig
ing. And you might forget that you failed to
your programs will compile and run, but they
So it would seem that it's best to provide a co
even though you don't need them now.

Chapter 18 returns to this topic.

Are there any other effects to note? Yes. Re
objects are passed (or returned) by value. How
preferred practice of passing objects as referen
other temporary objects. But the `Queue` defini
objects, such as overloading the addition oper

The `Customer` Class

At this point, we need to design a customer cl
properties, such as a name, account numbers, a
properties you need for the simulation are wh
required for the customer's transaction. When
the program should create a new customer ob
arrival and a randomly generated value for the
reaches the front of the queue, the program sh
joining time to get the customer's waiting tim
ment the `Customer` class:

```
class Customer
{
private:
    long arrive;        // arrival time fo
    int processtime;    // processing time
```

```
// This queue will contain Customer items
class Customer
{
private:
    long arrive;        // arrival time f
    int processtime;    // processing tim
public:
    Customer() { arrive = processtime = 0
    void set(long when);
    long when() const { return arrive; }
    int ptime() const { return processtim
};

typedef Customer Item;

class Queue
{
private:
// class scope definitions
    // Node is a nested structure definit
    struct Node { Item item; struct Node
    enum {Q_SIZE = 10};
// private class members
    Node * front;       // pointer to fro
```

```
// Queue methods
Queue::Queue(int qs) : qsize(qs)
{
    front = rear = NULL;    // or nullptr
    items = 0;
}

Queue::~Queue()
{
    Node * temp;
    while (front != NULL)   // while queue
    {
        temp = front;       // save addres
        front = front->next;// reset point
        delete temp;        // delete form
    }
}

bool Queue::isempty() const
{
    return items == 0;
}
```

```
        rear->next = add;   // else place
    rear = add;             // have rear
    return true;
}

// Place front item into item variable an
bool Queue::dequeue(Item & item)
{
    if (front == NULL)
        return false;
    item = front->item;     // set item t
    items--;
    Node * temp = front;    // save locat
    front = front->next;    // reset fron
    delete temp;            // delete for
    if (items == 0)
        rear = NULL;
    return true;
}

// customer method

// when is the time at which the customer
// the arrival time is set to when and th
```

3. If a customer is being processed, decren

4. Track various quantities, such as the nu
customers turned away, cumulative time
queue length.

When the simulation cycle is finished, the
findings.

An interesting matter is how the program
arrived. Suppose that on average, 10 customer
tomer every 6 minutes. The program compute
`min_per_cust`. However, having a customer s
tic. What you really want (at least most of the
to a customer every 6 minutes. The program
customer shows up during a cycle:

```
bool newcustomer(double x)
{
    return (std::rand() * x / RAND_MAX < 1
}
```

Here's how it works. The value `RAND_MAX` i
`stdlib.h`) and represents the largest value the
value). Suppose that `x`, the average time betwe
`* x / RAND_MAX` will be somewhere between

```
int main()
{
    using std::cin;
    using std::cout;
    using std::endl;
    using std::ios_base;
// setting things up
    std::srand(std::time(0));    //  rand

    cout << "Case Study: Bank of Heather
    cout << "Enter maximum size of queue:
    int qs;
    cin >> qs;
    Queue line(qs);          // line queue

    cout << "Enter the number of simulati
    int hours;               //  hours of
    cin >> hours;
    // simulation will run 1 cycle per mi
    long cyclelimit = MIN_PER_HR * hours;

    cout << "Enter the average number of
    double perhour;          //  average #
```

```
                temp.set(cycle);    // cyc
                line.enqueue(temp); // add
            }
        }
        if (wait_time <= 0 && !line.isempt
        {
            line.dequeue (temp);       // a
            wait_time = temp.ptime(); // f
            line_wait += cycle - temp.when
            served++;
        }
        if (wait_time > 0)
            wait_time--;
        sum_line += line.queuecount();
    }

// reporting results
    if (customers > 0)
    {
        cout << "customers accepted: " <<
        cout << "  customers served: " <<
        cout << "         turnaways: " <<
        cout << "average queue size: ";
        cout.precision(2);
        cout.setf(ios_base::fixed, ios_bas
```

RAND_MAX yourself.

Here are a few sample runs of the progran

```
Case Study: Bank of Heather Automatic Tel
Enter maximum size of queue: 10
Enter the number of simulation hours: 100
Enter the average number of customers per
customers accepted: 1485
  customers served: 1485
         turnaways: 0
average queue size: 0.15
 average wait time: 0.63 minutes
Done!

Case Study: Bank of Heather Automatic Tel
Enter maximum size of queue: 10
Enter the number of simulation hours: 100
Enter the average number of customers per
customers accepted: 2896
  customers served: 2888
         turnaways: 101
average queue size: 4.64
 average wait time: 9.63 minutes
Done!
```

```
customers accepted: 114
  customers served: 110
         turnaways: 0
average queue size: 2.15
 average wait time: 4.52 minutes
Done!

Case Study: Bank of Heather Automatic Tel
Enter maximum size of queue: 10
Enter the number of simulation hours: 4
Enter the average number of customers per
customers accepted: 121
  customers served: 116
         turnaways: 5
average queue size: 5.28
 average wait time: 10.72 minutes
Done!

Case Study: Bank of Heather Automatic Tel
Enter maximum size of queue: 10
Enter the number of simulation hours: 4
Enter the average number of customers per
customers accepted: 112
  customers served: 109
```

demise of an object automatically triggers th

Objects that have members pointing to m
with initializing one object to another or assi
C++ uses memberwise initialization and assi
the assigned-to object winds up with exact c
original member points to a block of data, th
When the program eventually deletes the tw
delete the same block of memory twice, whi
cial copy constructor that redefines initializat
In each case, the new definition should creat
the new object point to the copies. That way
separate but identical data, with no overlap. T
assignment operator. In each case, the goal is
real data and not just pointers to the data.

When an object has automatic storage or
object is called automatically when the objec
object by using `new` and assign its address to a
called automatically when you apply `delete`
age for class objects by using placement `new`
responsibility of calling the destructor for tha
method with a pointer to the object. C++ al
meration definitions inside a class. Such neste

```
    int items = 0;
    const int qsize = Q_SIZE;
...
};
```

This is equivalent to using a member initia
a membership initialization list will override t

As you might have noticed, classes require
do simple C-style structures. In return, they d

Chapter Review

1. Suppose a `String` class has the followi

```
class String
{
private:
    char * str;    // points to stri
    int len;       // holds length o
//...
};
```

4. Identify and correct the errors in the f

```
class nifty
{
// data
    char personality[];
    int talents;
// methods
    nifty();
    nifty(char * s);
    ostream & operator<<(ostream & o
}

nifty:nifty()
{
    personality = NULL;
    talents = 0;
}

nifty:nifty(char * s)
{
    personality = new char [strlen(s
    personality = s;
    talents = 0;
}
```

```
Golfer lulu("Little Lulu");
Golfer roy("Roy Hobbs", 12);
Golfer * par = new Golfer;
Golfer next = lulu;
Golfer hazzard = "Weed Thwacke
*par = nancy;
nancy = "Nancy Putter";
```

b. Clearly, the class requires several
tional method does it require to

Programming Exercises

1. Consider the following class declaration

```
class Cow {
    char name[20];
    char * hobby;
    double weight;
public:
    Cow();
    Cow(const char * nm, const char
    Cow(const Cow c&);
```

```
// pe12_2.cpp
#include <iostream>
using namespace std;
#include "string2.h"
int main()
{
    String s1(" and I am a C++ stud
    String s2 = "Please enter your 
    String s3;
    cout << s2;                 // 
    cin >> s3;                  // 
    s2 = "My name is " + s3;    // 
    cout << s2 << ".\n";
    s2 = s2 + s1;
    s2.stringup();              // 
    cout << "The string\n" << s2 <<
         << " 'A' characters in it.
    s1 = "red";     // String(const
                    // then String 
    String rgb[3] = { String(s1), S
    cout << "Enter the name of a pr
    String ans;
    bool success = false;
    while (cin >> ans)
```

```
My name is Fretta Farbo.
The string
MY NAME IS FRETTA FARBO AND I AM A C
contains 6 'A' characters in it.
Enter the name of a primary color fo
Try again!
BLUE
That's right!
Bye
```

3. Rewrite the `Stock` class, as described in
that it uses dynamically allocated memo
objects to hold the stock names. Also re
overloaded `operator<<()` definition. Te
10.9.

4. Consider the following variation of the

```
// stack.h -- class declaration for
typedef unsigned long Item;

class Stack
{
```

5. The Bank of Heather has performed a
wait more than one minute in line. Us
value for number of customers per hou
minute. (Use at least a 100-hour trial p

6. The Bank of Heather would like to kn
ATM. Modify the simulation in this ch
a customer will join the first queue if i
queue and that the customer will join
value for number of customers per hou
minute. (Note: This is a nonlinear prob
doesn't double the number of custome
minute wait maximum.)

- Virtual member functions
- Early (static) binding and late (dynami
- Abstract base classes
- Pure virtual functions
- When and how to use public inheritan

One of the main goals of object-oriented
When you develop a new project, particularl
reuse proven code rather than to reinvent it.
it has already been used and tested, can help
gram. Also the less you have to concern your
trate on overall program strategy.

Traditional C function libraries provide re
functions, such as `strlen()` and `rand()`, tha
dors furnish specialized C libraries that provi
library. For example, you can purchase librari
screen control functions. However, function
supplies the source code for its library functi
modify the functions to meet your particular
gram to meet the workings of the library. Ev
you run the risk of unintentionally modifyin
the relationships among library functions as

Of course, you could accomplish the same
and modifying it, but the inheritance mechan
the new features. You don't even need access t
you purchase a class library that provides only
class methods, you can still derive new classes
can distribute your own classes to others, keep
still giving your clients the option of adding f

Inheritance is a splendid concept, and its b
managing inheritance so that it works proper
This chapter looks at both the simple and the

Beginning with a Simple

When one class inherits from another, the ori
inheriting class is called a *derived class*. So to il
class. The Webtown Social Club has decided t
tennis. As head programmer for the club, you
`TableTennisPlayer` class defined in Listings

Listing 13.1 **tabtenn0.h**

```
// tabtenn0.h -- a table-tennis base class
#ifndef TABTENN0_H_
#define TABTENN0_H_
#include <string>
```

```
TableTennisPlayer::TableTennisPlayer (con
    const string & ln, bool ht) : firstna
        lastname(ln), hasTable(ht) {}

void TableTennisPlayer::Name() const
{
    std::cout << lastname << ", " << firs
}
```

All the TableTennisPlayer class does is k
they have tables. There are a couple of points
string class to hold the names. This is more
character array. And it is rather more professi
"Classes and Dynamic Memory Allocation."
initializer list syntax introduced in Chapter 1

```
TableTennisPlayer::TableTennisPlayer (con
                  const string & ln, boo
{
    firstname = fn;
    lastname = ln;
    hasTable = ht;
}
```

```
    if (player2.HasTable())
        cout << ": has a table";
    else
        cout << ": hasn't a table.\n";

    return 0;
}
```

And here's the output of the program in L

```
Blizzard, Chuck: has a table.
Boomdea, Tara: hasn't a table.
```

Note that the program uses constructors w

```
TableTennisPlayer player1("Chuck", "Blizza
TableTennisPlayer player2("Tara", "Boomdea
```

But the formal parameters for the construc
This is a type mismatch, but the `string` class,
has a constructor with a `const char *` param
cally to create a `string` object initialized by t
either a `string` object or a C-style string as a
constructor. The first invokes a `string` constr
and the second invokes a `string` constructor

- An object of the derived type has store
  type. (The derived class inherits the ba
- An object of the derived type can use
  class inherits the base-class interface.)

Thus, a `RatedPlayer` object can store th
and whether the player has a table. Also a `Ra`
`HasTable()`, and `ResetTable()` methods fr
Figure 13.1 for another example).

What needs to be added to these inherite

- A derived class needs its own construc
- A derived class can add additional data

In this particular case, the class needs one
value. It should also have a method for retrie
the rating. So the class declaration could loo

```
// simple derived class
class RatedPlayer : public TableTennisPla
{
private:
    unsigned int rating;     // add a dat
```

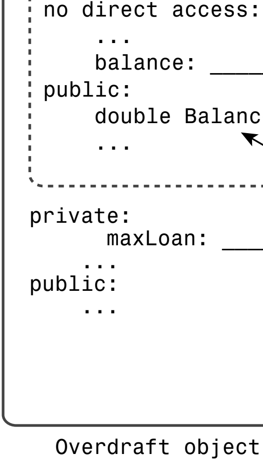


Figure 13.1 Base-class a

The constructors have to provide data for
ited members. The first `RatedPlayer` constru
each member, and the second `RatedPlayer` c
parameter, which bundles three items (`first`
gle unit.

example, a program has the following declara

```
RatedPlayer rplayer1(1140, "Mallory", "Du
```

The `RatedPlayer` constructor assigns the `true` to the formal parameters `fn`, `ln`, and `ht` arguments to the `TableTennisPlayer` constr embedded `TableTennisPlayer` object and s in it. Then the program enters the body of th construction of the `RatedPlayer` object, and `1140`) to the `rating` member (see Figure 13.

derived-class constructor

```
Overdraft::Overdraft(const char * s,

         double ml, double r) : Band
{
   ...
}
```

base-class co

Figure 13.2 Passing arguments t

```
    rating = r;
}
```

Again, the `TableTennisPlayer` informatio
constructor:

```
TableTennisPlayer(tp)
```

Because `tp` is type `const TableTennisPl`
constructor. The base class didn't define a cop
that the compiler automatically generates a co
haven't defined one already. In this case, the in
berwise copying, is fine because the class does
(The `string` members do use dynamic mem
ing will use the `string` class copy constructo

You may, if you like, also use member initi
class. In this case, you use the member name i
second constructor can also be written in this

```
// alternative version
RatedPlayer::RatedPlayer(unsigned int r, c
    : TableTennisPlayer(tp), rating(r)
{
}
```

When an object of a derived class expires, th
tor and then calls the base-class destructor.

Member Initializer Lists

A constructor for a derived class can use the
to a base-class constructor. Consider this ex

```
derived::derived(type1 x, type2 y) : b
{
    ...
}
```

Here, `derived` is the derived class, `base` is
by the base-class constructor. If, say, the der
`10` and `12`, this mechanism then passes `10`
defined as taking arguments of these types.
(see Chapter 14, "Reusing Code in C++"), a
ate base class. However, that class can use
to its immediate base class, and so on. If yo
member initializer list, the program uses the
tializer list can be used *only* in constructors.

```
                       const string & ln =
    void Name() const;
    bool HasTable() const { return hasTabl
    void ResetTable(bool v) { hasTable = v
};

// simple derived class
class RatedPlayer : public TableTennisPlay
{
private:
    unsigned int rating;
public:
    RatedPlayer (unsigned int r = 0, const
                 const string & ln = "none
    RatedPlayer(unsigned int r, const Tabl
    unsigned int Rating() const { return r
    void ResetRating (unsigned int r) {rat
};

#endif
```

Listing 13.5 provides the method definitio
arate files, but it's simpler to keep the definitic

```
    : TableTennisPlayer(tp), rating(r)
{
}
```

Listing 13.6 creates objects of both the Ta
class. Notice that objects of both classes can u
HasTable() methods.

Listing 13.6 **usett1.cpp**

```
// usett1.cpp -- using base class and der
#include <iostream>
#include "tabtenn1.h"

int main ( void )
{
    using std::cout;
    using std::endl;
    TableTennisPlayer player1("Tara", "Bo
    RatedPlayer rplayer1(1140, "Mallory",
    rplayer1.Name();            // derived
    if (rplayer1.HasTable())
        cout << ": has a table.\n";
    else
```

```
Name: Boomdea, Tara; Rating: 1212
```

Special Relationships Between De

A derived class has some special relationships
seen, is that a derived-class object can use base
are not private:

```
RatedPlayer rplayer1(1140, "Mallory", "Duc
rplayer1.Name();  // derived object uses b
```

Two other important relationships are that
class object without an explicit type cast and t
derived-class object without an explicit type c

```
RatedPlayer rplayer1(1140, "Mallory", "Duc
TableTennisPlayer & rt = rplayer;
TableTennisPlayer * pt = &rplayer;
rt.Name();   // invoke Name() with referen
pt->Name();  // invoke Name() with pointer
```

However, a base-class pointer or reference
couldn't use `rt` or `pt` to invoke, say, the derive

```
{
    using std::cout;
    cout << "Name: ";
    rt.Name();
    cout << "\nTable: ";
    if (rt.HasTable())
        cout << "yes\n";
    else
        cout << "no\n";
}
```

The formal parameter rt is a reference to
object or to a derived-class object. Thus, you
argument or a RatedPlayer argument:

```
TableTennisPlayer player1("Tara", "Boomde
RatedPlayer rplayer1(1140, "Mallory", "Du
Show(player1);   // works with TableTenni
Show(rplayer1);  // works with RatedPlaye
```

A similar relationship would hold for a fun
parameter; it could be used with either the ac
a derived-class object as an actual argument:

`TableTennisPlayer` object embedded in the

Similarly, you can assign a derived-class obj

```
RatedPlayer olaf1(1840, "Olaf", "Loaf", tr
TableTennisPlayer winner;
winner = olaf1; // assign derived to base
```

In this case, the program uses the implicit o

```
TableTennisPlayer & operator=(const TableT
```

Again, a base-class reference refers to a deri
tion of `olaf1` is copied to `winner`.

Inheritance: An *Is-a* Relat

The special relationship between a derived cla
model for C++ inheritance. Actually, C++ ha
tected, and private. Public inheritance is the m
relationship. This is shorthand for saying that a
object of the base class. Anything you do with
with a derived-class object. Suppose, for exam
store, say, the weight and caloric content of a f
fruit, you could derive a `Banana` class from the
the data members of the original class, so a `Ba`
ing the weight and caloric content of a banan

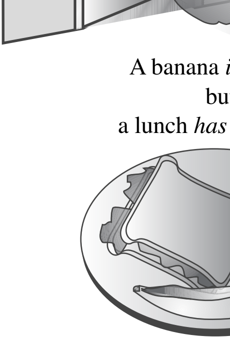


Figure 13.3 Is-a ar

Public inheritance doesn't model an *is-like*
It's often pointed out that lawyers are like sha
a shark. For example, sharks can live underwa
class from a `Shark` class. Inheritance can add
properties from a base class. In some cases, sh
ing a class encompassing those characteristics
has-a relationship, to define the related classes

you can have multiple behaviors for a method
key mechanisms for implementing polymorph

- Redefining base-class methods in a deri
- Using virtual methods

It's time for another example. You have lev
Social Club to become head programmer for
the bank asks you to do is develop two classes
plan, the Brass Account, and the second class
account, which adds an overdraft protection f
(but not too much larger) than his or her bala
the user for the excess payment and adding a
accounts in terms of data to be stored and op
First, here is the information for a Brass Ac

- Client name
- Account number
- Current balance

And here are the operations to be represen

- Creating an account
- Depositing money into the account

BrassPlus class meet the *is-a* test? Sure. Ever
true for a BrassPlus object. Both store a clie
With both, you can make deposits and withd
that the *is-a* relationship is not, in general, sy
similarly, a Brass object won't have all the ca

Developing the Brass and Bras

The Brass Account class information is pretty
you enough details about how the overdraft
for further information, the friendly Pontoo
following:

- A Brass Plus Account limits how much
drafts. The default value is $500, but so
- The bank may change a customer's ove
- A Brass Plus Account charges interest o
some customers may start with a differ
- The bank may change a customer's int
- The account keeps track of how much
plus interest). The user cannot pay off t

```
{
private:
    std::string fullName;
    long acctNum;
    double balance;
public:
    Brass(const std::string & s = "Nullbod
                double bal = 0.0);
    void Deposit(double amt);
    virtual void Withdraw(double amt);
    double Balance() const;
    virtual void ViewAcct() const;
    virtual ~Brass() {}
};

//Brass Plus Account Class
class BrassPlus : public Brass
{
private:
    double maxLoan;
    double rate;
    double owesBank;
public:
    BrassPlus(const std::string & s = "Nul
            double bal = 0.0, double ml =
```

- The Brass class also declares a virtual d
nothing.

The first point in the list is nothing new. T
when it added a new data member and two n

The second point in the list is how the de
differently for the derived class. The two Vie
be two separate method definitions. The qual
Brass::ViewAcct(), and the qualified name
BrassPlus::ViewAcct(). A program will us
sion to use:

```
Brass dom("Dominic Banker", 11224, 4183.4
BrassPlus dot("Dorothy Banker", 12118, 25
dom.ViewAcct();      // use Brass::ViewAc
dot.ViewAcct();      // use BrassPlus::Vi
```

Similarly, there will be two versions of Wit
and one that's used by BrassPlus objects. M
such as Deposit() and Balance(), are declar

The third point (the use of virtual) is m
determines which method is used if the meth
instead of by an object. If you don't use the k
method based on the reference type or point

In this case, both references are type Brass
BrassPlus::ViewAcct() is used for it. Using
results in similar behavior.

It turns out, as you'll see in a bit, that this b
Therefore, it is the common practice to declar
that might be redefined in a derived class. Wh
class, it is automatically virtual in the derived
which functions are virtual by using the keyw
tions, too.

The fourth point is that the base class decla
that the correct sequence of destructors is call
discuss this point in more detail later in this c

Note

If you redefine a base-class method in a deriv
base-class method as virtual. This makes the
object type instead of the type of a reference
declare a virtual destructor for the base class

```
Brass::Brass(const string & s, long an, d
{
    fullName = s;
    acctNum = an;
    balance = bal;
}

void Brass::Deposit(double amt)
{
    if (amt < 0)
        cout << "Negative deposit not all
             << "deposit is cancelled.\n"
    else
        balance += amt;
}

void Brass::Withdraw(double amt)
{
    // set up ###.## format
    format initialState = setFormat();
    precis prec = cout.precision(2);

    if (amt < 0)
        cout << "Withdrawal amount must b
```

```
}

// BrassPlus Methods
BrassPlus::BrassPlus(const string & s, lon
          double ml, double r) : Brass(s,
{
    maxLoan = ml;
    owesBank = 0.0;
    rate = r;
}

BrassPlus::BrassPlus(const Brass & ba, dou
          : Brass(ba)   // uses implicit
{
    maxLoan = ml;
    owesBank = 0.0;
    rate = r;
}

// redefine how ViewAcct() works
void BrassPlus::ViewAcct() const
{
    // set up ###.## format
    format initialState = setFormat();
```

```
        owesBank += advance * (1.0 + rate
        cout << "Bank advance: $" << adva
        cout << "Finance charge: $" << ad
        Deposit(advance);
        Brass::Withdraw(amt);
    }
    else
        cout << "Credit limit exceeded. T
    restore(initialState, prec);
}

format setFormat()
{
    // set up ###.## format
    return cout.setf(std::ios_base::fixed
            std::ios_base::floatfield
}

void restore(format f, precis p)
{
    cout.setf(f, std::ios_base::floatfiel
    cout.precision(p);
}
```

```
    owesBank = 0.0;
    rate = r;
}
```

Each of these constructors uses the membe
information to a base-class constructor and th
new data items added by the BrassPlus class.

Non-constructors can't use the member in
method can call a public base-class method. Fo
the core of the BrassPlus version of ViewAcc

```
// redefine how ViewAcct() works
void BrassPlus::ViewAcct() const
{
...
    Brass::ViewAcct();   // display base p
    cout << "Maximum loan: $" << maxLoan <
    cout << "Owed to bank: $" << owesBank
    cout << "Loan Rate: " << 100 * rate <<
...
}
```

In other words, BrassPlus::ViewAcct() c
and calls on the base-class method Brass::Vi
members. Using the scope-resolution operato
class method is a standard technique.

```
{
...
    double bal = Balance();
    if (amt <= bal)
        Brass::Withdraw(amt);
    else if ( amt <= bal + maxLoan - owes
    {
        double advance = amt - bal;
        owesBank += advance * (1.0 + rate
        cout << "Bank advance: $" << adva
        cout << "Finance charge: $" << ad
        Deposit(advance);
        Brass::Withdraw(amt);
    }
    else
        cout << "Credit limit exceeded. T
...
}
```

Note that the method uses the base-class
nal balance. The code doesn't have to use the
because this method has not been redefined i

The `ViewAcct()` and the `Withdraw()` me
formatting methods to set the output mode f

```
    cout.setf(f, std::ios_base::floatfield
    cout.precision(p);
}
```

You can find more details about formatting

Using the `Brass` and `BrassPlus` Class

Listing 13.9 shows the class definitions with a

Listing 13.9 **usebrass1.cpp**

```
// usebrass1.cpp -- testing bank account c
// compile with brass.cpp
#include <iostream>
#include "brass.h"

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

    Brass Piggy("Porcelot Pigg", 381299, 4
    BrassPlus Hoggy("Horatio Hogg", 382288
    Piggy.ViewAcct();
    cout << endl;
```

```
Account Number: 382288
Balance: $3000.00
Maximum loan: $500.00
Owed to bank: $0.00
Loan Rate: 11.125%

Depositing $1000 into the Hogg Account:
New balance: $4000
Withdrawing $4200 from the Pigg Account:
Withdrawal amount of $4200.00 exceeds you
Withdrawal canceled.
Pigg account balance: $4000
Withdrawing $4200 from the Hogg Account:
Bank advance: $200.00
Finance charge: $22.25
Client: Horatio Hogg
Account Number: 382288
Balance: $0.00
Maximum loan: $500.00
Owed to bank: $222.25
Loan Rate: 11.125%
```

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

   Brass * p_clients[CLIENTS];
   std::string temp;
   long tempnum;
   double tempbal;
   char kind;

   for (int i = 0; i < CLIENTS; i++)
   {
       cout << "Enter client's name: ";
       getline(cin,temp);
       cout << "Enter client's account num
       cin >> tempnum;
       cout << "Enter opening balance: $";
       cin >> tempbal;
       cout << "Enter 1 for Brass Account
            << "2 for BrassPlus Account: "
       while (cin >> kind && (kind != '1'
           cout <<"Enter either 1 or 2: ";
```

```
    for (int i = 0; i < CLIENTS; i++)
    {
        delete p_clients[i];  // free memo
    }
    cout << "Done.\n";
    return 0;
}
```

The program in Listing 13.10 lets user inp added and then uses `new` to create and initiali recall that `getline(cin,temp)` reads a line o object `temp`.

Here is a sample run of the program in Li

```
Enter client's name: Harry Fishsong
Enter client's account number: 112233
Enter opening balance: $1500
Enter 1 for Brass Account or 2 for BrassP
Enter client's name: Dinah Otternoe
Enter client's account number: 121213
Enter opening balance: $1800
Enter 1 for Brass Account or 2 for BrassP
Enter the overdraft limit: $350
Enter the interest rate as a decimal frac
```

```
Account Number: 212118
Balance: $5200.00
Maximum loan: $800.00
Owed to bank: $0.00
Loan Rate: 10.00%

Client: Tim Turtletop
Account Number: 233255
Balance: $688.00

Done.
```

The polymorphic aspect is provided by the

```
for (i = 0; i < CLIENTS; i++)
{
    p_clients[i]->ViewAcct();
    cout << endl;
}
```

If the array member points to a `Brass` obje
array member points to a `BrassPlus` object, B
`Brass::ViewAcct()` were been declared as vi
in all cases.

With C++, the task is more complex becaus
to look at the function arguments as well as t
tion to use. Nonetheless, this kind of binding
form during the compiling process; binding t
static binding (or *early binding*). However, virtu
As shown in Listing 13.10, the decision of w
pile time because the compiler doesn't know
choose to make. Therefore, the compiler has
tual method to be selected as the program ru
binding).

Now that you've seen virtual methods at
depth, beginning with how C++ handles po

Pointer and Reference Type Com

Dynamic binding in C++ is associated with
and this is governed, in part, by the inheritan
els the *is-a* relationship is in how it handles p
C++ does not allow you to assign an address
Nor does it let a reference to one type refer t

```
double x = 2.5;
int * pi = &x;   // invalid assignment, m
long & rl = x;   // invalid assignment, m
```

these data members wouldn't apply to the base
Singer class from an Employee class, adding a
range and a member function, called range(),
It wouldn't make sense to apply the range() r
implicit downcasting were allowed, you could
to the address of an Employee object and use
(see Figure 13.4).

Upcasting also takes place for function calls
parameters. Consider the following code fragm
the virtual method ViewAcct():

```
void fr(Brass & rb); // uses rb.ViewAcct()
void fp(Brass * pb); // uses pb->ViewAcct(
void fv(Brass b);    // uses b.ViewAcct()
int main()
{
    Brass b("Billy Bee", 123432, 10000.0);
    BrassPlus bp("Betty Beep", 232313, 123
    fr(b);  // uses Brass::ViewAcct()
    fr(bp); // uses BrassPlus::ViewAcct()
    fp(b);  // uses Brass::ViewAcct()
    fp(bp); // uses BrassPlus::ViewAcct()
```

>ViewAcct() goes by the pointer type (Bras
pointer type is known at compile time, so the
Brass::ViewAcct() at compile time. In shor
virtual methods.

But if ViewAcct() is declared as virtual in
object type (BrassPlus) and invokes BrassP
see that the object type is BrassPlus, but, in
might only be determined when the program
ates code that binds ViewAcct() to Brass::V
depending on the object type, while the prog
dynamic binding for virtual methods.

In most cases, dynamic binding is a good t
the method designed for a particular type. Gi
the following:

- Why have two kinds of binding?
- If dynamic binding is so good, why isn
- How does it work?

We'll look at answers to these questions n ods you expect to be redefined.

> **Tip**
>
> If a method in a base class will be redefined
> the method should not be redefined, you shou

Of course, when you design a class, it's not
method falls. Like many aspects of real life, cla

How Virtual Functions Work

C++ specifies how virtual functions should b
the compiler writer. You don't need to know
functions, but seeing how it is done may help
take a look.

The usual way compilers handle virtual fu
object. The hidden member holds a pointer to
array is usually termed a *virtual function table* (*v*
tual functions declared for objects of that class
tains a pointer to a table of addresses of all the
a derived class contains a pointer to a separate
vides a new definition of a virtual function, th

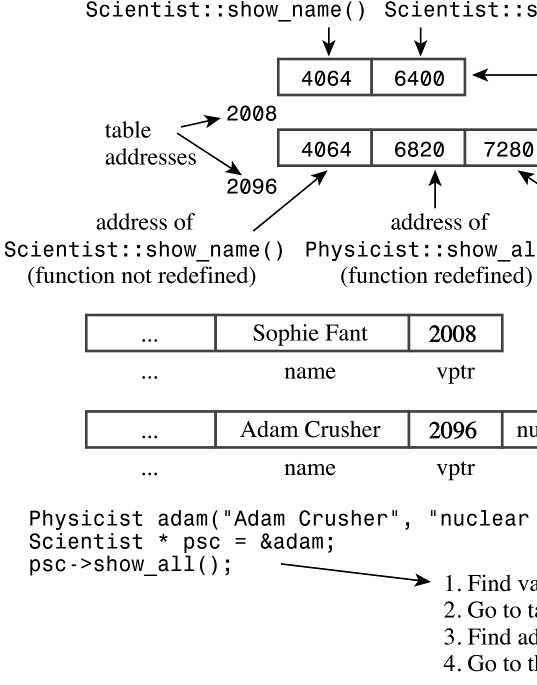


Figure 13.5 A virtua

class, including classes derived from the

- If a virtual method is invoked by using a pointer to an object, the program uses t than the method defined for the referen *late*, *binding*. This behavior is important b pointer or reference to refer to an objec
- If you're defining a class that will be use declare as virtual functions the class met derived classes.

There are several other things you may nee which have been mentioned in passing already

Constructors

Constructors can't be virtual. Creating a deriv tor, not a base-class constructor. The derived-c structor, but this sequence is distinct from the doesn't inherit the base-class constructors, so u them virtual, anyway.

Destructors

Destructors should be virtual unless a class isn suppose `Employee` is a base class and `Singer` i

Normally, you should provide a base class wi
need a destructor.

Friends

Friends can't be virtual functions because frie
bers can be virtual functions. If this poses a p
sidestep it by having the friend function use v

No Redefinition

If a derived class fails to redefine a function (
version of the function. If a derived class is pa
the most recently defined version of the func
hidden, as described next.

Redefinition Hides Methods

Suppose you create something like the follow

```
class Dwelling
{
public:
      virtual void showperks(int a) const
...
};
class Hovel : public Dwelling
```

new exception to this rule is that a return ty
class can be replaced by a reference or pointe
covariance of return type because the return typ
class type:

```
class Dwelling
{
public:
// a base method
      virtual Dwelling & build(int n);
      ...
};
class Hovel : public Dwelling
{
public:
// a derived method with a covariant retur
      virtual Hovel & build(int n);  // sa
      ...
};
```

Note that this exception applies only to re

Second, if the base class declaration is overl
class versions in the derived class:

Access Control: protec

So far the class examples in this book have us control access to class members. There is one word `protected`. The `protected` keyword is access class members in a `protected` section ference between `private` and `protected` co from the base class. Members of a derived cla class directly, but they cannot directly access p bers in the protected category behave like pri concerned but behave like public members a

For example, suppose the `Brass` class decl

```
class Brass
{
protected:
    double balance;
...
};
```

In this case, the `BrassPlus` class could acc methods. For example, the core of `BrassPlu` you could write code like this:

```
void BrassPlus::Reset(double amt)
{
    balance = amt;
}
```

The `Brass` class was designed so that the D
vides the only means for altering `balance`. Bu
`balance` a public variable as far as `BrassPlus`
ple, the safeguards found in `Withdraw()`.

> **Caution**
>
> You should prefer private to protected access
> should use base-class methods to provide de

However, protected access control can be c
derived classes access to internal functions tha

Abstract Base Classes

So far you've seen simple inheritance and the
The next step in increasing sophistication is th
some programming situations that provide the

```
public:
    ...
    void Move(int nx, ny) { x = nx; y = n
    virtual double Area() const { return
    virtual void Rotate(double nang) { an
    virtual void Scale(double sa, double
    ...
};
```

Now suppose you derive a `Circle` class fr

```
class Circle : public Ellipse
{
    ...
};
```

Although a circle is an ellipse, this derivati only a single value, its radius, to describe its s axis (`a`) and semiminor axis (`b`). The `Circle` the same value to both the `a` and `b` members of the same information. The `angle` paramet make sense for a circle, and the `Scale()` met non-circle by scaling the two axes differently putting a redefined `Rotate()` method in the

Circle and Ellipse classes from the ABC. Th
base-class pointers to manage a mixture of El
use a polymorphic approach. In this case, wha
coordinates of the center of the shape; a Move
an Area() method, which works differently fo
method can't even be implemented for the AB
data members. C++ has a way to provide an u
tual function. A pure virtual function has = 0 a
Area() method:

```
class BaseEllipse  // abstract base class
{
private:
    double x;   // x-coordinate of center
    double y;   // y-coordinate of center
    ...
public:
    BaseEllipse(double x0 = 0, double y0 =
    virtual ~BaseEllipse() {}
    void Move(int nx, ny) { x = nx; y = ny
    virtual double Area() const = 0; // a
   ...
}
```

A program using these classes would be a
objects but no `BaseEllipse` objects. Becaus
base class, a collection of such objects can be
pointers. Classes such as `Circle` and `Ellips`
indicate that you can create objects of those

In short, an ABC describes an interface th
classes derived from an ABC use regular virt
terms of the properties of the particular deri

Applying the ABC Concept

You'd probably like to see a complete examp
representing the `Brass` and `BrassPlus` accou
This class should contain all methods and da
`Brass` and the `BrassPlus` classes. The metho
`BrassPlus` class than they do for the `Brass`
At least one virtual function should be a pur
`AcctABC` class abstract.

Listing 13.11 is a header file that declares
and `BrassPlus` classes (both concrete classes
data, `AcctABC` provides some protected meth
ods that derived-class methods can call but t
derived-class objects. `AcctABC` also provides

```
    };
    const std::string & FullName() const
    long AcctNum() const {return acctNum
    Formatting SetFormat() const;
    void Restore(Formatting & f) const;
public:
    AcctABC(const std::string & s = "Null
                double bal = 0.0);
    void Deposit(double amt) ;
    virtual void Withdraw(double amt) =
    double Balance() const {return balan
    virtual void ViewAcct() const = 0;
    virtual ~AcctABC() {}
};

// Brass Account Class
class Brass :public AcctABC
{
public:
    Brass(const std::string & s = "Nullb
           double bal = 0.0) : AcctABC(s,
    virtual void Withdraw(double amt);
    virtual void ViewAcct() const;
    virtual ~Brass() {}
};
```

Listing 13.12 **acctabc.cpp**

```
// acctabc.cpp -- bank account class meth
#include <iostream>
#include "acctabc.h"
using std::cout;
using std::ios_base;
using std::endl;
using std::string;

// Abstract Base Class
AcctABC::AcctABC(const string & s, long a
{
    fullName = s;
    acctNum = an;
    balance = bal;
}

void AcctABC::Deposit(double amt)
{
    if (amt < 0)
        cout << "Negative deposit not all
             << "deposit is cancelled.\n"
```

```
    cout.precision(f.pr);
}

// Brass methods
void Brass::Withdraw(double amt)
{
    if (amt < 0)
        cout << "Withdrawal amount must be
             << "withdrawal canceled.\n";
    else if (amt <= Balance())
        AcctABC::Withdraw(amt);
    else
        cout << "Withdrawal amount of $" <
             << " exceeds your balance.\n"
             << "Withdrawal canceled.\n";
}

void Brass::ViewAcct() const
{

    Formatting f = SetFormat();
    cout << "Brass Client: " << FullName()
    cout << "Account Number: " << AcctNum
```

```
{
    Formatting f = SetFormat();

    cout << "BrassPlus Client: " << FullN
    cout << "Account Number: " << AcctNum
    cout << "Balance: $" << Balance() <<
    cout << "Maximum loan: $" << maxLoan
    cout << "Owed to bank: $" << owesBank
    cout.precision(3);
    cout << "Loan Rate: " << 100 * rate <
    Restore(f);
}

void BrassPlus::Withdraw(double amt)
{
    Formatting f = SetFormat();

    double bal = Balance();
    if (amt <= bal)
        AcctABC::Withdraw(amt);
    else if ( amt <= bal + maxLoan - owes
    {
        double advance = amt - bal;
        owesBank += advance * (1.0 + rate
```

structure to set and restore formats with just t

```
struct Formatting
{
     std::ios_base::fmtfglas flag;
     std::streamsize pr;
};
...
Formatting f = SetFormat();
...
Restore(f);
```

It's a neater look.

A problem with the older version was that
were standalone functions, so those function n
functions of the same name. There are several
declare both functions `static`, making them
cessor, `acctabc.cpp`. A second is to place both
definition into a namespace. But one of the to
this example places the structure definition an
class definition. This makes them available to t
hiding them from the outside world.

```
const int CLIENTS = 4;

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

   AcctABC * p_clients[CLIENTS];
   std::string temp;
   long tempnum;
   double tempbal;
   char kind;

   for (int i = 0; i < CLIENTS; i++)
   {
       cout << "Enter client's name: "
       getline(cin,temp);
       cout << "Enter client's account
       cin >> tempnum;
       cout << "Enter opening balance:
       cin >> tempbal;
       cout << "Enter 1 for Brass Acco
            << "2 for BrassPlus Accoun
```

```
    }

    for (int i = 0; i < CLIENTS; i++)
    {
        delete p_clients[i];  // free memo
    }
    cout << "Done.\n";

    return 0;
}
```

The program itself behaves the same as the
same input as for Listing 13.10, the output wo

ABC Philosophy

The ABC methodology is a much more syste
tance than the more ad hoc, spur-of-the-mon
example. Before designing an ABC, you first h
needed to represent a programming problem
school of thought holds that if you design an
concrete classes should be those that never se
produce cleaner designs with fewer complicat

```
//  Base Class Using DMA
class baseDMA
{
private:
    char * label;
    int rating;

public:
    baseDMA(const char * l = "null", int
    baseDMA(const baseDMA & rs);
    virtual ~baseDMA();
    baseDMA & operator=(const baseDMA & r
...
};
```

The declaration contains the special meth
`new`: a destructor, a copy constructor, and an

Now suppose you derive a `lackDMA` class
`new` or have other unusual design features tha

```
// derived class without DMA
class lacksDMA :public baseDMA
{
private:
    char color[40];
```

a class automatically uses the base-class assign
So it, too, is fine.

These properties of inherited objects also
selves objects. For example, Chapter 10, "Obj
by using a `string` object to represent the co
our `String` example, uses dynamic memory a
create problems. The default `Stock` copy cons
structor to copy the `company` member of an o
tor would use the `string` assignment operato
object, and the `Stock` destructor (default or o
`string` destructor.

Case 2: Derived Class Does Use n

Suppose that the derived class uses `new`:

```
// derived class with DMA
class hasDMA :public baseDMA
{
private:
    char * style;  // use new in construct
public:
...
};
```

```
{
    label = new char[std::strlen(rs.label
    std::strcpy(label, rs.label);
    rating = rs.rating;
}
```

The hasDMA copy constructor only has ac
baseDMA copy constructor to handle the bas

```
hasDMA::hasDMA(const hasDMA & hs)
         : baseDMA(hs)
{
    style = new char[std::strlen(hs.style
    std::strcpy(style, hs.style);
}
```

The point to note is that the member init
baseDMA constructor. There is no baseDMA co
parameter, but none is needed. That's because
baseDMA reference parameter, and a base class
the baseDMA copy constructor uses the baseI
struct the baseDMA portion of the new objec

Next, consider assignment operators. The
usual pattern:

```
    style = new char[std::strlen(hs.style)
    std::strcpy(style, hs.style);
    return *this;
}
```

The following statement might look a littl

```
baseDMA::operator=(hs);  // copy base port
```

But using function notation instead of ope
tion operator. In effect, the statement means t

```
*this = hs;  // use baseDMA::operator=()
```

But, of course, the compiler ignores comm
compiler would use `hasDMA::operator=()` in
function notation gets the correct assignment

In summary, when both the base class and
cation, the derived-class destructor, copy cons
use their base-class counterparts to handle the
requirement is accomplished three different w
cally. For a constructor, it is accomplished by
the member initialization list, or else the defa
the assignment operator, it is accomplished by
explicit call of the base-class assignment opera

```
    baseDMA(const char * l = "null", int
    baseDMA(const baseDMA & rs);
    virtual ~baseDMA();
    baseDMA & operator=(const baseDMA & r
    friend std::ostream & operator<<(std:
                                     cons
};

// derived class without DMA
// no destructor needed
// uses implicit copy constructor
// uses implicit assignment operator
class lacksDMA :public baseDMA
{
private:
    enum { COL_LEN = 40};
    char color[COL_LEN];
public:
    lacksDMA(const char * c = "blank", co
             int r = 0);
    lacksDMA(const char * c, const baseDM
    friend std::ostream & operator<<(std:
                                     cons
};
```

```
// dma.cpp --dma class methods

#include "dma.h"
#include <cstring>

// baseDMA methods
baseDMA::baseDMA(const char * l, int r)
{
    label = new char[std::strlen(l) + 1];
    std::strcpy(label, l);
    rating = r;
}

baseDMA::baseDMA(const baseDMA & rs)
{
    label = new char[std::strlen(rs.label)
    std::strcpy(label, rs.label);
    rating = rs.rating;
}

baseDMA::~baseDMA()
{
    delete [] label;
}
```

```
    color[39] = '\0';
}

lacksDMA::lacksDMA(const char * c, const
    : baseDMA(rs)
{
    std::strncpy(color, c, COL_LEN - 1);
    color[COL_LEN - 1] = '\0';
}

std::ostream & operator<<(std::ostream &
{
    os << (const baseDMA &) ls;
    os << "Color: " << ls.color << std::e
    return os;
}

// hasDMA methods
hasDMA::hasDMA(const char * s, const char
        : baseDMA(l, r)
{
    style = new char[std::strlen(s) + 1];
    std::strcpy(style, s);
}
```

```
    baseDMA::operator=(hs);  // copy base
    delete [] style;         // prepare fo
    style = new char[std::strlen(hs.style)
    std::strcpy(style, hs.style);
    return *this;
}

std::ostream & operator<<(std::ostream & o
{
    os << (const baseDMA &) hs;
    os << "Style: " << hs.style << std::en
    return os;
}
```

The new feature to note in Listings 13.14 a
of a friend to a base class. Consider, for exampl

```
friend std::ostream & operator<<(std::ostr
                                 const has
```

Being a friend to the `hasDMA` class gives th
there's a problem: This function is not a frienc
the `label` and `rating` members? The solution

```
    using std::cout;
    using std::endl;

    baseDMA shirt("Portabelly", 8);
    lacksDMA balloon("red", "Blimpo", 4);
    hasDMA map("Mercator", "Buffalo Keys"
    cout << "Displaying baseDMA object:\n
    cout << shirt << endl;
    cout << "Displaying lacksDMA object:\
    cout << balloon << endl;
    cout << "Displaying hasDMA object:\n"
    cout << map << endl;
    lacksDMA balloon2(balloon);
    cout << "Result of lacksDMA copy:\n";
    cout << balloon2 << endl;
    hasDMA map2;
    map2 = map;
    cout << "Result of hasDMA assignment:
    cout << map2 << endl;
    return 0;
}
```

```
Rating: 5
Style: Mercator
```

Class Design Review

C++ can be applied to a wide variety of prog
class design to some paint-by-numbers routin
often apply, and this is as good a time as any t
earlier discussions.

Member Functions That the Comp

As first discussed in Chapter 12, the compiler
member functions, termed *special member funct*
these special member functions are particularl
them now.

Default Constructors

A default constructor is one that has no argum
ments have default arguments. If you don't de
default constructor for you. Its existence allow
pose `Star` is a class. You need a default constru

A class copy constructor is used in the fol

- When a new object is initialized to an
- When an object is passed to a function
- When a function returns an object by
- When the compiler generates a tempo

If a program doesn't use a copy construct
vides a prototype but not a function definitio
constructor that performs memberwise initia
object is initialized to the value of the corres
member is itself a class object, then memberv
defined for that particular class.

In some cases, memberwise initialization i
initialized with `new` generally require that yo
class example. Or a class may have a static var
cases, you need to define your own copy con

Assignment Operators

A default assignment operator handles assign
class. Don't confuse assignment with initializ
using initialization, and if a statement alters th

conversion function approach can lead to con

Chapter 18, "Visiting with the New C++ ods added by C++11: the move constructor a

Other Class Method Consideratio

There are several other points to keep in min tions list some of them.

Constructor Considerations

Constructors are different from other class me whereas other methods are invoked by existin aren't inherited. Inheritance means a derived o case of constructors, the object doesn't exist u

Destructor Considerations

You need to remember to define an explicit d by `new` in the class constructors and takes care destroying a class object requires. If the class is vide a virtual destructor even if the class does

```
...
public:
    explicit Star(const char *);
...
};
...
Star north;
north = "polaris";        // not allowed
north = Star("polaris");  // allowed
```

To convert from a class object to some ot
(see Chapter 11, "Working with Classes"). A
tion with no arguments or declared return ty
verted to. Despite having no declared return
conversion value. Here are some examples:

```
Star::Star double() {...}           // con
Star::Star const char * () {...}    // con
```

You should be judicious with such functi
sense. Also with some class designs, having co
of writing ambiguous code. For example, sup
`Vector` type of Chapter 11, and suppose you

```
Vector ius(6.0, 0.0);
Vector lux = ius + 20.2;        // ambigu
```

Some class methods return objects. You've pro
objects directly whereas others return referen
object, but if it isn't necessary, you should use
closely.

First, the only coding difference between r
reference is in the function prototype and hea

```
Star nova1(const Star &);     // returns a
Star & nova2(const Star &);   // returns a
```

Next, the reason you should return a refere
an object involves generating a temporary cop
made available to the calling program. Thus, r
calling a copy constructor to generate the cop
to get rid of the copy. Returning a reference s
object directly is analogous to passing an obje
rary copies. Similarly, returning a reference is
Both the calling and the called function opera

However, it's not always possible to return
reference to a temporary object created in the
invalid when the function terminates and the
should return an object in order to generate a
program.

method doesn't modify an argument:

```
Star::Star(const char * s) {...} // won't
```

You can use `const` to guarantee that a me

```
void Star::show() const {...} // won't ch
```

Here `const` means `const Star * this`,

Normally, a function that returns a referen
statement, which really means you can assign
can use `const` to ensure that a reference or p
data in an object:

```
const Stock & Stock::topval(const Stock &
{
   if (s.total_val > total_val)
      return s;          // argument ob
   else
      return *this;      // invoking ob
}
```

Here the method returns a reference eithe
both declared `const`, the function is not allo
returned reference also must be declared con

explicit type cast. Depending on the class dec
cast) may or may not make sense. (You might

What's Not Inherited

Constructors are not inherited. That is, creatin
derived-class constructor. However, derived-c
initializer list syntax to call on base-class cons
of a derived object. If the derived-class constr
structor by using the member-initializer list sy
structor. In an inheritance chain, each class ca
information to its immediate base class. C++
iting of constructors. However, the default be
inherited.

Destructors are not inherited either. Howe
gram first calls the derived destructor and the
base class destructor, the compiler generates a
speaking, if a class serves as a base class, its des

Assignment operators are not inherited. Th
the same function signature in a derived class
assignment operator has a function signature
a formal parameter that is the class type. Assig
properties, which we'll look at next.

Figure 13.4 Upcast

Assignment Operator Considerations

If the compiler detects that a program assigns
automatically supplies that class with an assig
sion of this operator uses memberwise assign
being assigned the value of the correspondin
the object belongs to a derived class, the com
to handle assignment for the base-class portic
explicitly provided an assignment operator fo
larly, if a class contains a member that is an o
tor for that class is used for that member.

As you've seen several times, you need to
class constructors use `new` to initialize pointe
ment operator for the base part of derived ol
ment operator for a derived class *unless* it add
example, the `baseDMA` class defines assignmer
uses the implicit assignment operator generat

lated into a method that is invoked by the left

```
blips.operator=(snips);
```

Here the left-hand object is a `Brass` objec `Brass::operator=(const Brass &)` functio erence to refer to a derived-class object, such deals with base-class members, so the `maxLoa` `snips` are ignored in the assignment. In short object, and only the base-class members are in

What about the reverse? Can you assign a look at this example:

```
Brass gp("Griff Hexbait", 21234, 1200);
BrassPlus temp;
temp = gp;   // possible?
```

Here the assignment statement would be t

```
temp.operator=(gp);
```

The left-hand object is a `BrassPlus` objec `BrassPlus::operator=(const BrassPlus` ence cannot automatically refer to a base-class is *also* a conversion constructor:

```
BrassPlus(const Brass &);
```

Private Versus Protected Members

Remember that protected members act like
concerned, but they act like private member
access protected members of a base class dire
via base-class member functions. Thus, makin
security, whereas making them protected sim
trup, in his book *The Design and Evolution of*
data members than protected data members

Virtual Method Considerations

When you design a base class, you have to de
If you want a derived class to be able to rede
tual in the base class. This enables late, or dyn
to be redefined, you don't make it virtual. Th
the method, but it should be interpreted as n

Note that inappropriate code can circumv
the following two functions:

```
void show(const Brass & rba)
{
    rba.ViewAcct();
    cout << endl;
}
```

a derived object via a base-class pointer or ref
derived-class destructor followed by the base-
base-class destructor.

Friend Considerations

Because a friend function is not actually a clas
might still want a friend to a derived class to u
accomplish this is to type cast a derived-class
alent and to then use the type cast reference o

```
ostream & operator<<(ostream & os, const h
{
//  type cast to match operator<<(ostream
    os << (const baseDMA &) hs;
    os << "Style: " << hs.style << endl;
    return os;
}
```

You can also use the `dynamic_cast<>` ope
Exceptions, and More," for the type cast:

```
os << dynamic_cast<const baseDMA &> (hs);
```

For reasons discussed in Chapter 15, this w

ence Manual, summarizes these properties. In
operators of the form `+=`, `*=`, and so on. Not
no different from those of the "other operato
rately is to point out that these operators do

Table 13.1 **Member Function Properties**

Function	Inherited	Member or Friend
Constructor	No	Member
Destructor	No	Member
`=`	No	Member
`&`	Yes	Either
Conversion	Yes	Member
`()`	Yes	Member
`[]`	Yes	Member
`->`	Yes	Member
`op=`	Yes	Either

public and protected base-class methods. You
ods to the class, and you can use the derived c
Each derived class requires its own constructo
object, it first calls a base-class constructor and
program deletes an object, it first calls the der
destructor.

If a class is meant to be a base class, you ma
of private members so that derived classes can
using private members, in general, reduces the
that a derived class can redefine a base-class m
tion by declaring it with the keyword `virtua`
or references to be handled on the basis of the
reference type or pointer type. In particular, th
mally be virtual.

You might want to define an ABC that de
implementation matters. For example, you co
which particular shape classes, such as `Circle`
include at least one pure virtual method. You
`0` before the closing semicolon of the declarat

```
virtual double area() const = 0;
```

8. Can you assign the address of an object
class? Can you assign the address of an
derived class?

9. Can you assign an object of a derived
assign an object of a base class to an ob

10. Suppose you define a function that tak
argument. Why can this function also u

11. Suppose you define a function that tak
the function passes a base-class object
derived-class object as an argument?

12. Why is it usually better to pass objects

13. Suppose `Corporation` is a base class an
suppose that each class defines a `head(`
the `Corporation` type, and that `ph` is a
object. How is `ph->head()` interprete

    a. Regular nonvirtual method

    b. Virtual method

```
// base class
class Cd {  // represents a CD disk
private:
    char performers[50];
    char label[20];
    int selections;   // number of s
    double playtime;  // playing tim
public:
    Cd(char * s1, char * s2, int n,
    Cd(const Cd & d);
    Cd();
    ~Cd();
    void Report() const;  // reports
    Cd & operator=(const Cd & d);
};
```

Derive a `Classic` class that adds an arr
identifying the primary work on the C
tions be virtual, modify the base-class d
method is not needed, remove it from
following program:

```
#include <iostream>
using namespace std;
#include "classic.h"     // which wi
```

```
    copy = c2;
    copy.Report()

    return 0;
}

void Bravo(const Cd & disk)
{
    disk.Report();
}
```

2. Do Programming Exercise 1 but use d
size arrays for the various strings tracke

3. Revise the `baseDMA-lacksDMA-hasDMA`
derived from an ABC. Test the result w
13.10. That is, it should feature an arra
to make runtime decisions as to what t
`View()` methods to the class definition

4. The Benevolent Order of Programme
describe it, the BOP Portmaster has de

```
Brand: Gallo
Kind: tawny
Bottles: 20
```

The operator<<() function presents in
newline character at the end):

```
Gallo, tawny, 20
```

The Portmaster completed the method
derived the VintagePort class as follow
accidentally routing a bottle of '45 Coc
tal barbecue sauce:

```
class VintagePort : public Port // s
{
private:
    char * nickname;           // i.
    int year;                  // vi
public:
    VintagePort();
    VintagePort(const char * br, int
    VintagePort(const VintagePort &
    ~VintagePort() { delete [] nickn
    VintagePort & operator=(const Vi
```

- Virtual base classes
- Creating class templates
- Using class templates
- Template specializations

One of the main goals of C++ is to facili
one mechanism for achieving this goal, but it
other choices. One technique is to use class m
another class. This is referred to as *containmen*
to use private or protected inheritance. Cont
inheritance are typically used to implement *h*
which the new class has an object of another
might have a `BluRayPlayer` object. Multiple
inherit from two or more base classes, combi

Chapter 10, "Objects and Classes," introdu
look at class templates, which provide anothe
you define a class in generic terms. Then you
classes defined for specific types. For example
and then use the template to create one class
another class that represents a stack of `doubl`
represents a stack of stacks.

your project.)

Representing the quiz scores presents simi
which places a size limitation. You could use c
large body of supporting code. You could use
memory allocation to represent an array. You c
that is capable of representing the data.

Developing your own class is not out of th
that difficult because an array of `double` share
you could base the design of an array-of-`doub`
fact, that is what earlier editions of this book c

But, of course, it is even easier if the library
does: the `valarray` class.

The `valarray` Class: A Quick Lo

The `valarray` class is supported by the `vala`
class is targeted to deal with numeric values (
supports operations such as summing the con
values in an array. So that it can handle differe
plate class. Later, this chapter goes into how t
know now is how to use one.

the values from an ordinary array. With C++

```
valarray<int> v5 = {20, 32, 17, 9};  // C
```

Next, here are a few of the methods:

- The `operator[]()` method provides a
- The `size()` method returns the numb
- The `sum()` method returns the sum of
- The `max()` method returns the largest
- The `min()` method returns the smalles

There are many more methods, some of v you've already seen more than enough to pro

The `Student` Class Design

At this point, the design plan for the `Student` the name and a `valarray<double>` object to be done? You might be tempted to publicly d That would be an example of multiple public would be inappropriate here. The reason is th classes doesn't fit the *is-a* model. A student is

holding the name. But a Student object does
string operator+=() function for appendin

Interfaces and Implementations

With public inheritance, a class inherits an int
virtual functions in a base class can provide a
ing the interface is part of the *is-a* relationshi
acquires the implementation without the inter
has-a relationship.

The fact that a class object doesn't automat
object is a good thing for a *has-a* relationship.
tor to allow concatenating two strings, but, co
nate two Student objects. That's one reason n
the other hand, parts of the interface for the c
class. For example, you might want to use the
interface to sort Student objects by name. You
Student::operator<() member function tha
function. Let's move on to some details.

The `Student` Class Example

At this point you need to provide the `Studer`
include constructors and at least a few functi
class. Listing 14.1 does this, defining all the c
friends for input and output.

Listing 14.1 **studentc.h**

```
// studentc.h -- defining a Student class
#ifndef STUDENTC_H_
#define STUDENTC_H_

#include <iostream>
#include <string>
#include <valarray>
class Student
{
private:
    typedef std::valarray<double> ArrayDb
    std::string name;       // contained
    ArrayDb scores;         // contained
    // private method for scores output
```

```
    // output
    friend std::ostream & operator<<(std::
                                     const
};

#endif
```

In order to simplify notation, the `Student`

```
typedef std::valarray<double> ArrayDb;
```

This enables the remaining code to use the of `std::valarray<double>`. Thus, methods a Placing this `typedef` in the private portion of internally in the `Student` implementation but

Note the use of the keyword `explicit`:

```
explicit Student(const std::string & s)
    : name(s), scores() {}
explicit Student(int n) : name("Nully"), s
```

Recall that a constructor that can be called conversion function from the argument type idea. In the second constructor, for instance, t elements in an array rather than a value for th

```
Queue::Queue(int qs) : qsize(qs)  {...} /
```

This code uses the name of the data mem
constructors from previous examples, such as
initialize the base-class portion of a derived c

```
hasDMA::hasDMA(const hasDMA & hs) : baseD
```

For *inherited* objects, constructors use the
invoke a specific base-class constructor. For *m*
name. For example, look at the last construct

```
Student(const char * str, const double *
      : name(str), scores(pd, n) {}
```

Because it initializes member objects, not
member names, not the class names, in the in
list invokes the matching constructor. That is,
`char *)` constructor, and `scores(pd, n)` in
constructor, which, because of the `typedef`,
`double *, int)` constructor.

What happens if you don't use the initializ
nents, C++ requires that all member objects
constructed. So if you omit the initialization
defined for the member objects' classes.

```
    else
        return 0;
}
```

This defines a method that can be invoked
valarray `size()` and `sum()` methods. That's be
invoke the member functions of the `valarray`
a `Student` method, and the `Student` method
invoke `valarray` methods.

Similarly, you can define a friend function
operator:

```
// use string version of operator<<()
ostream & operator<<(ostream & os, const S
{
    os << "Scores for " << stu.name << ":\
    ...
}
```

Because `stu.name` is a `string` object, it in
`string &)` function, which is provided as par
`operator<<(ostream & os, const Student`
`Student` class so that it can access the `name` m
the public `Name()` method instead of the priv

```
}
```

Using a helper like this gathers the messy
coding of the friend function neater:

```
// use string version of operator<<()
ostream & operator<<(ostream & os, const
{
    os << "Scores for " << stu.name << ":
    stu.arr_out(os);  // use private meth
    return os;
}
```

The helper function could also act as a bu
tions, should you choose to provide them.

Listing 14.2 shows the class methods file f
allow you to use the `[]` operator to access in

Listing 14.2 **studentc.cpp**

```
// studentc.cpp -- Student class using co
#include "studentc.h"
using std::ostream;
using std::endl;
using std::istream;
using std::string;
```

```
// private method
ostream & Student::arr_out(ostream & os) c
{
    int i;
    int lim = scores.size();
    if (lim > 0)
    {
        for (i = 0; i < lim; i++)
        {
            os << scores[i] << " ";
            if (i % 5 == 4)
                os << endl;
        }
        if (i % 5 != 0)
            os << endl;
    }
    else
        os << " empty array ";
    return os;
}

// friends
```

Aside from the private helper method, Lis
Using containment allows you to take advant
already written.

Using the New `Student` Class

Let's put together a small program to test the
should use an array of just three `Student` obj
should use an unsophisticated input cycle tha
cut the input process short. Listing 14.3 prese
along with `studentc.cpp`.

Listing 14.3 **`use_stuc.cpp`**

```
// use_stuc.cpp -- using a composite clas
// compile with studentc.cpp
#include <iostream>
#include "studentc.h"
using std::cin;
using std::cout;
using std::endl;

void set(Student & sa, int n);
```

```
void set(Student & sa, int n)
{
    cout << "Please enter the student's na
    getline(cin, sa);
    cout << "Please enter " << n << " quiz
    for (int i = 0; i < n; i++)
        cin >> sa[i];
    while (cin.get() != '\n')
        continue;
}
```

Here is a sample run of the program in Lis

```
Please enter the student's name: Gil Bayts
Please enter 5 quiz scores:
92 94 96 93 95
Please enter the student's name: Pat Roone
Please enter 5 quiz scores:
83 89 72 78 95
Please enter the student's name: Fleur O'D
Please enter 5 quiz scores:
92 89 96 74 64
```

private inheritance, public and protected memb
of the derived class. This means the methods
public interface of the derived object. They c
functions of the derived class.

Let's look at the interface topic more clos
ods of the base class become public methods
inherits the base-class interface. This is part o
tance, the public methods of the base class be
short, the derived class does not inherit the b
objects, this lack of inheritance is part of the

With private inheritance, a class does inhe
base a `Student` class on a `string` class, the `St`
`string` class component that can be used to
methods can use the `string` methods intern

Containment adds an object to a class as a
inheritance adds an object to a class as an un
term *subobject* to denote an object added by i

Private inheritance, then, provides the sam
implementation, don't acquire the interface. T
has-a relationship. In fact, you can produce a
and has the same public interface as the cont
between the two approaches affect the imple
you can use private inheritance to redesign th

Initializing Base-Class Components

Having implicitly inherited components inste
this example because you can no longer use n
Instead, you have to go back to the technique
ple, consider constructors. Containment uses t

```
Student(const char * str, const double * p
   : name(str), scores(pd, n) {}      // u
```

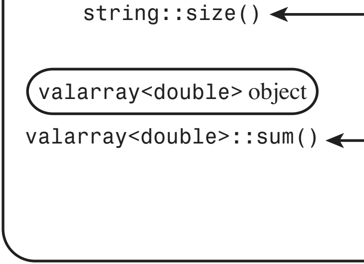
The new version should use the member i
which uses the *class* name instead of a *member*

```
Student(const char * str, const double * p
   : std::string(str), ArrayDb(pd, n) {}
```

Here, as in the preceding example, `ArrayDb`
Be sure to note that the member initializer lis
instead of `name(str)`. This is the second main

Listing 14.4 shows the new class declaratio
explicit object names and the use of class nam
constructors.

```
        : std::string(s), ArrayDb(a)
    Student(const char * str, const doubl
        : std::string(str), ArrayDb(p
    ~Student() {}
    double Average() const;
    double & operator[](int i);
    double operator[](int i) const;
    const std::string & Name() const;
// friends
    // input
    friend std::istream & operator>>(std:
                                   Stud
    friend std::istream & getline(std::is
                                 Student
    // output
    friend std::ostream & operator<<(std:
                                   cons
};

#endif
```




```
class Student:private str
private valarray<double>
{
   …
};
```

Figure 14.2 Objects withi

Here, however, inheritance lets you use the
tor to invoke base-class methods:

```
double Student::Average() const
{
    if (ArrayDb::size() > 0)
        return ArrayDb::sum()/ArrayDb::siz
```

}

This code returns a reference to the inher
Student object.

Accessing Base-Class Friends

The technique of explicitly qualifying a func
for friend functions because a friend functior
use an explicit type cast to the base class to in
the same technique used to access a base-clas
you have a name for the Student object, so t
example, consider the following friend functi

```
ostream & operator<<(ostream & os, const
{
     os << "Scores for " << (const Strin
...

}
```

If plato is a Student object, then the foll
stu being a reference to plato and os being

```
cout << plato;
```

```
using std::ostream;
using std::endl;
using std::istream;
using std::string;

// public methods
double Student::Average() const
{
    if (ArrayDb::size() > 0)
        return ArrayDb::sum()/ArrayDb::siz
    else
        return 0;
}

const string & Student::Name() const
{
    return (const string &) *this;
}

double & Student::operator[](int i)
{
    return ArrayDb::operator[](i);
}
```

```
    return os;
}

// friends
// use String version of operator>>()
istream & operator>>(istream & is, Studen
{
    is >> (string &)stu;
    return is;
}

// use string friend getline(ostream &, c
istream & getline(istream & is, Student &
{
    getline(is, (string &)stu);
    return is;
}

// use string version of operator<<()
ostream & operator<<(ostream & os, const
{
    os << "Scores for " << (const string
    stu.arr_out(os);  // use private meth
    return os;
}
```

```
const int pupils = 3;
const int quizzes = 5;

int main()
{
    Student ada[pupils] =
        {Student(quizzes), Student(quizzes

    int i;
    for (i = 0; i < pupils; i++)
        set(ada[i], quizzes);
    cout << "\nStudent List:\n";
    for (i = 0; i < pupils; ++i)
        cout << ada[i].Name() << endl;
    cout << "\nResults:";
    for (i = 0; i < pupils; i++)
    {
        cout << endl << ada[i];
        cout << "average: " << ada[i].Aver
    }
    cout << "Done.\n";
    return 0;
}
```

```
Student List:
Gil Bayts
Pat Roone
Fleur O'Day

Results:
Scores for Gil Bayts:
92 94 96 93 95
average: 94

Scores for Pat Roone:
83 89 72 78 95
average: 83.4

Scores for Fleur O'Day:
92 89 96 74 64
average: 83
Done.
```

The same input as before leads to the sar
produces.

Another situation that calls for using priva
tual functions. Again, this is a privilege accord
class. With private inheritance, the redefined f
class, not publicly.

> **Tip**
>
> In general, you should use containment to mo
> vate inheritance if the new class needs to acc
> if it needs to redefine virtual functions.

Protected Inheritance

Protected inheritance is a variation on private
when listing a base class:

```
class Student : protected std::string,
                protected std::valarray<do
{...};
```

With protected inheritance, public and pro
tected members of the derived class. As with p
class is available to the derived class but not to
between private and protected inheritance oc
derived class. With private inheritance, this thi
use of the base-class interface. That's because t

Redefining Access with `using`

Public members of a base class become prote
private derivation. Suppose you want to mak
publicly in the derived class. One option is to
base-class method. For example, suppose you
`valarray sum()` method. You can declare a s
then define the method this way:

```
double Student::sum() const    // public
{
    return std::valarray<double>::sum();
}
```

Then a `Student` object can invoke `Studen`
`valarray<double>::sum()` method to the e
`typedef` is in scope, you can use `ArrayDb` ins

There is an alternative to wrapping one fu
tion (such as those used with namespaces) to
ber can be used by the derived class, even tho
suppose you want to be able to use the `vala`

remove the existing prototypes and definition
declaration approach works only for inheritan

There is an older way to redeclare base-cla
place the method name in the public section
do that:

```
class Student : private std::string, priva
{
public:
    std::valarray<double>::operator[];  //
    ...

};
```

This looks like a `using` declaration withou
deprecated, meaning that the intention is to pha
`using` declaration, you can use it to make a m
the derived class.

Multiple Inheritance

MI describes a class that has more than one in
tance, public MI should express an *is-a* relatio
class and a `Singer` class, you could derive a `Si`

which is the circumstance that causes the mo
tions for the `Worker`, `Waiter`, and `Singer` cla

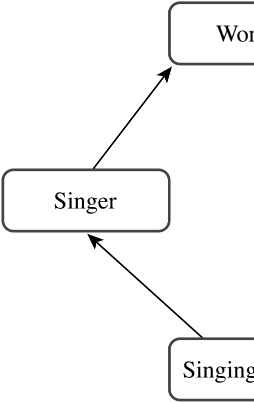


Figure 14.3 MI wi

```
{
private:
    int panache;
public:
    Waiter() : Worker(), panache(0) {}
    Waiter(const std::string & s, long n,
            : Worker(s, n), panache(p) {}
    Waiter(const Worker & wk, int p = 0)
            : Worker(wk), panache(p) {}
    void Set();
    void Show() const;
};

class Singer : public Worker
{
protected:
    enum {other, alto, contralto, soprano,
                    bass, baritone, tenor}
    enum {Vtypes = 7};
private:
    static char *pv[Vtypes];    // string
    int voice;
public:
    Singer() : Worker(), voice(other) {}
```

```
using std::endl;
// Worker methods

// must implement virtual destructor, eve
Worker::~Worker() {}

void Worker::Set()
{
    cout << "Enter worker's name: ";
    getline(cin, fullname);
    cout << "Enter worker's ID: ";
    cin >> id;
    while (cin.get() != '\n')
        continue;
}

void Worker::Show() const
{
    cout << "Name: " << fullname << "\n";
    cout << "Employee ID: " << id << "\n"
}

// Waiter methods
void Waiter::Set()
```

```
    cout << "Enter number for singer's voc
    int i;
    for (i = 0; i < Vtypes; i++)
    {
        cout << i << ": " << pv[i] << "
        if ( i % 4 == 3)
            cout << endl;
    }
    if (i % 4 != 0)
        cout << endl;
    while (cin >>  voice && (voice < 0 ||
        cout << "Please enter a value >= 0

    while (cin.get() != '\n')
        continue;
}

void Singer::Show() const
{
    cout << "Category: singer\n";
    Worker::Show();
    cout << "Vocal range: " << pv[voice] <
}
```

```
        std::cout << std::endl;
    }

    return 0;
}
```

Here is the output of the program in Listi

```
Enter waiter's name: Waldo Dropmaster
Enter worker's ID: 442
Enter waiter's panache rating: 3
Enter singer's name: Sylvie Sirenne
Enter worker's ID: 555
Enter number for singer's vocal range:
0: other   1: alto   2: contralto   3: so
4: bass   5: baritone   6: tenor
3
Category: waiter
Name: Bob Apple
Employee ID: 314
Panache rating: 5

Category: singer
Name: Beverly Hills
```

How Many Workers?

Suppose you begin by publicly deriving Sing

```
class SingingWaiter: public Singer, public
```

Because both `Singer` and `Waiter` inherit a up with two `Worker` components (see Figure

As you might expect, this raises problems. address of a derived-class object to a base-clas

```
SingingWaiter ed;
Worker * pw = &ed;   // ambiguous
```

Normally, such an assignment sets a base-c object within the derived object. But `ed` cont addresses from which to choose. You could sp

```
Worker * pw1 = (Waiter *) &ed;   // the Wo
Worker * pw2 = (Singer *) &ed;   // the Wo
```

This certainly complicates the technique o refer to a variety of objects (polymorphism).

Having two copies of a `Worker` object caus issue is why should you have two copies of a any other worker, should have just one name of tricks, it added a virtual base class to make

Figure 14.4 Inheriting

Virtual Base Classes

Virtual base classes allow an object derived fr
common base to inherit just one object of th
would make Worker a virtual base class to Si
virtual in the class declarations (virtual ar

```
class Singer : virtual public Worker {...
class Waiter : public virtual Worker {...
```

Then you would define SingingWaiter a

```
class SingingWaiter: public Singer, publi
```

Now a SingingWaiter object will contai
essence, the inherited Singer and Waiter ob
of each bringing in its own copy (see Figure
one Worker subobject, you can use polymor

Let's look at some questions you might hav

- Why the term *virtual*?
- Why don't we dispense with declaring b the norm for MI?
- Are there any catches?

First, why the term virtual? After all, there between the concepts of virtual functions and from the C++ community to resist the introd ward, for example, if a new keyword correspo tion or variable in a major program. So C++ the new facility—a bit of keyword overloadin

Next, why don't we dispense with declarin behavior the norm for MI? First, there are cas of a base. Second, making a base class virtual r accounting, and you shouldn't have to pay for are the disadvantages presented in the next pa

Finally, are there catches? Yes. Making virtu C++ rules, and you have to code some things may involve changing existing code. For exam

```
class C : public B
{
    int c;
public:
    C(int q = 0, int m = 0, int n = 0) :
    ...
};
```

A `C` constructor can invoke only construct
invoke only constructors from the `A` class. He
passes the values of `m` and `n` back to the `B` con
`m` and passes the value of `n` back to the `A` cons

This automatic passing of information do
example, consider the following possible cons

```
SingingWaiter(const Worker & wk, int p =
                  : Waiter(wk,p), Singer(
```

The problem is that automatic passing of i
object via two separate paths (`Waiter` and `Si`
disables the automatic passing of information
if the base class is virtual. Thus, the previous c
`voice` members, but the information in the `w`
ject. However, the compiler must construct a
derived objects; in this case, it will use the de

object to invoke an inherited Show() method

```
SingingWaiter newhire("Elise Hawks", 2005,
newhire.Show();  // ambiguous
```

With single inheritance, failing to redefine
ancestral definition. In this case, each direct an
this call ambiguous.

> **Caution**
>
> Multiple Inheritance can result in ambiguous
> could inherit two quite different Draw() meth
> PokerPlayer class.

You can use the scope-resolution operator

```
SingingWaiter newhire("Elise Hawks", 2005,
newhire.Singer::Show();  // use Singer ver
```

However, a better approach is to redefine S
specify which Show() to use. For example, if y
Singer version, you could use this:

```
    Waiter::Show();
    cout << "Presence rating: " << presen
}
```

This incremental approach fails for the `Si` method fails because it ignores the `Waiter` c

```
void SingingWaiter::Show()
{
    Singer::Show();
}
```

You can remedy that by calling the `Waite`

```
void SingingWaiter::Show()
{
     Singer::Show();
     Waiter::Show();
}
```

However, this displays a person's name and with `Waiter::Show()` both call `Worker::Sh`

How can you fix this? One way is to use tal approach. That is, you can provide a meth another method that displays only `Waiter` co components), and another that displays only

```
}

void SingingWaiter::Show() const
{
    cout << "Category: singing waiter\n";
    Worker::Data();
    Data();
}
```

Similarly, the other `Show()` methods would
components.

With this approach, objects would still use
methods, on the other hand, should be intern
methods used to facilitate the public interface
vate would prevent, say, `Waiter` code from usi
of situation for which the protected access cla
tected, they can by used internally by all the c
den from the outside world.

Another approach would be to make all th
vate, but using protected methods instead of p
allowable access to the data.

The `Set()` methods, which solicit data for
lem. For example, `SingingWaiter::Set()` sho

```
    virtual void Get();
public:
    Worker() : fullname("no one"), id(0L)
    Worker(const std::string & s, long n)
            : fullname(s), id(n) {}
    virtual ~Worker() = 0; // pure virtua
    virtual void Set() = 0;
    virtual void Show() const = 0;
};

class Waiter : virtual public Worker
{
private:
    int panache;
protected:
    void Data() const;
    void Get();
public:
    Waiter() : Worker(), panache(0) {}
    Waiter(const std::string & s, long n,
            : Worker(s, n), panache(p) {}
    Waiter(const Worker & wk, int p = 0)
            : Worker(wk), panache(p) {}
```

```
};

// multiple inheritance
class SingingWaiter : public Singer, publi
{
protected:
    void Data() const;
    void Get();
public:
    SingingWaiter()  {}
    SingingWaiter(const std::string & s, l
                            int v = other)
            : Worker(s,n), Waiter(s, n, p)
    SingingWaiter(const Worker & wk, int p
            : Worker(wk), Waiter(wk,p), Si
    SingingWaiter(const Waiter & wt, int v
            : Worker(wt),Waiter(wt), Singe
    SingingWaiter(const Singer & wt, int p
            : Worker(wt),Waiter(wt,p), Sin
    void Set();
    void Show() const;
};

#endif
```

```
    while (cin.get() != '\n')
        continue;
}

// Waiter methods
void Waiter::Set()
{
    cout << "Enter waiter's name: ";
    Worker::Get();
    Get();
}

void Waiter::Show() const
{
    cout << "Category: waiter\n";
    Worker::Data();
    Data();
}

// protected methods
void Waiter::Data() const
{
    cout << "Panache rating: " << panache
}
```

```
    Worker::Data();
    Data();
}

// protected methods
void Singer::Data() const
{
    cout << "Vocal range: " << pv[voice] <
}

void Singer::Get()
{
    cout << "Enter number for singer's voc
    int i;
    for (i = 0; i < Vtypes; i++)
    {
        cout << i << ": " << pv[i] << "
        if ( i % 4 == 3)
            cout << endl;
    }
    if (i % 4 != 0)
        cout << '\n';
    cin >>  voice;
```

```
}

void SingingWaiter::Show() const
{
    cout << "Category: singing waiter\n";
    Worker::Data();
    Data();
}
```

Of course, curiosity demands that you tes
code to do so. Note that the program makes
addresses of various kinds of classes to base-c
C-style `string` library function `strchr()` in

```
while (strchr("wstq", choice) == NULL)
```

This function returns the address of the fi
in the string `"wstq"`; the function returns the
This test is simpler to write than an `if` staten
individually.

Be sure to compile Listing 14.12 along wi

```
        << "w: waiter  s: singer  "
        << "t: singing waiter  q: quit
    cin >> choice;
    while (strchr("wstq", choice) == N
    {
        cout << "Please enter a w, s,
        cin >> choice;
    }
    if (choice == 'q')
        break;
    switch(choice)
    {
        case 'w':   lolas[ct] = new Wa
                    break;
        case 's':   lolas[ct] = new Si
                    break;
        case 't':   lolas[ct] = new Si
                    break;
    }
    cin.get();
    lolas[ct]->Set();
}

cout << "\nHere is your staff:\n";
```

```
Enter singer's name: Sinclair Parma
Enter worker's ID: 1044
Enter number for singer's vocal range:
0: other   1: alto   2: contralto   3: sc
4: bass   5: baritone   6: tenor
5
Enter the employee category:
w: waiter  s: singer  t: singing waiter
t
Enter singing waiter's name: Natasha Garg
Enter worker's ID: 1021
Enter waiter's panache rating: 6
Enter number for singer's vocal range:
0: other   1: alto   2: contralto   3: sc
4: bass   5: baritone   6: tenor
3
Enter the employee category:
w: waiter  s: singer  t: singing waiter
q

Here is your staff:

Category: waiter
Name: Wally Slipshod
```

X, and Y. In this case, class M contains one class
ancestors (that is, classes C and D) and a separa
ancestor (that is, classes X and Y). So, all told, it
When a class inherits a particular base class th
virtual paths, the class has one base-class subo
separate base-class subobject to represent each

Virtual Base Classes and Dominance

Using virtual base classes alters how C++ res
classes, the rules are simple. If a class inherits t
with the same name from different classes, usi
class name is ambiguous. If virtual base classes
may not be ambiguous. In this case, if one nar
biguously without a qualifier.

So how does one member name dominate
nates the same name in any ancestor class, wh
sider the following definitions:

```
class B
{
public:
      short q();
      ...
```

```
{
    ...
};
```

Here the definition of `q()` in class `C` domi
derived from `B`. Thus, methods in `F` can use `q`
neither definition of `omg()` dominates the ot
the other. Therefore, an attempt by `F` to use a

The virtual ambiguity rules pay no attenti
`E::omg()` is private and hence not directly ac
ous. Similarly, even if `C::q()` were private, it
could call `B::q()` in class `F`, but an unqualifie
ble `C::q()`.

Multiple Inheritance Synopsis

First, let's review MI without virtual base cla
However, if a class inherits two members wit
you need to use class qualifiers in the derived
bers. That is, methods in the `BadDude` class, d
would use `Gunslinger::draw()` and `PokerP`
`draw()` methods inherited from the two class
about ambiguous usage.

Class Templates

Inheritance (public, private, or protected) and
when you want to reuse code. Consider, for e
and the `Queue` class (see Chapter 12). These ar
classes designed to hold other objects or data
example, stores `unsigned long` values. You co
ing `double` values or `string` objects. The cod
object stored. However, rather than write new
could define a stack in a generic (that is, type-
specific type as a parameter to the class. Then
produce stacks of different kinds of values. In
`typedef` as a first pass at dealing with this des
drawbacks. First, you have to edit the header f
you can use the technique to generate just on
can't have a `typedef` represent two different t
method to define a stack of `int`s and a stack o

C++'s class templates provide a better way
originally did not support templates, and since
ued to evolve, so it is possible that your comp
presented here.) Templates provide *parameteriz*
a type name as an argument to a recipe for bu

```
    Stack();
    bool isempty() const;
    bool isfull() const;
    // push() returns false if stack alre
    bool push(const Item & item);   // ad
    // pop() returns false if stack alrea
    bool pop(Item & item);          // po
};
```

The template approach will replace the St
and the Stack member functions with templ
functions, you preface a template class with c

```
template <class Type>
```

The keyword template informs the comp
The part in angle brackets is analogous to an
the keyword class as serving as a type name
and of *Type* as representing a name for this v

Using class here doesn't mean that *Type*
serves as a generic type specifier for which a
plate is used. Newer C++ implementations a
typename instead of class in this context:

```
template <typename Type>  // newer choice
```

```
...
}
```

becomes the following:

```
template <class Type>
bool Stack<Type>::push(const Type & item)
{
...
}
```

If you define a method within the class de
the template preface and the class qualifier.

Listing 14.13 shows the combined class an
to realize that these templates are not class an
are instructions to the C++ compiler about h
definitions. A particular actualization of a tem
string objects, is called an *instantiation* or a *sp*
functions in a separate implementation file w
provide the keyword export to allow such a s
vendors implemented it. C++11 discontinues
keyword for possible future use.) Because the
compiled separately. Templates have to be use
instantiations of templates. The simplest way t

```
template <class Type>
Stack<Type>::Stack()
{
    top = 0;
}

template <class Type>
bool Stack<Type>::isempty()
{
    return top == 0;
}

template <class Type>
bool Stack<Type>::isfull()
{
    return top == MAX;
}

template <class Type>
bool Stack<Type>::push(const Type & item)
{
    if (top < MAX)
    {
```

Using a Template Class

Merely including a template in a program do
ask for an instantiation. To do this, you declar
ing the generic type name with the particula
you would create two stacks, one for stacking

```
Stack<int> kernels;      // create a stac
Stack<string> colonels;  // create a stac
```

Seeing these two declarations, the compile
generate two separate class declarations and t
`Stack<int>` class declaration will replace `Typ`
`Stack<string>` class declaration will replace
the algorithms you use have to be consistent
ple, assumes that you can assign one item to a
types, structures, and classes (unless you make
for arrays.

Generic type identifiers such as `Type` in th
that they act something like variables, but ins
parameter, you assign a type to it. So in the `k`
has the value `int`.

Notice that you have to provide the desire
nary function templates, for which the compi
to figure out what kind of function to genera

```
std::string po;
cout << "Please enter A to add a purc
     << "P to process a PO, or Q to q
while (cin >> ch && std::toupper(ch)
{
    while (cin.get() != '\n')
        continue;
    if (!std::isalpha(ch))
    {
        cout << '\a';
        continue;
    }
    switch(ch)
    {
        case 'A':
        case 'a': cout << "Enter a PO
                  cin >> po;
                  if (st.isfull())
                      cout << "stack
                  else
                      st.push(po);
                  break;
        case 'P':
        case 'p': if (st.isempty())
                      cout << "stack
```

```
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
A
Enter a PO number to add: silver747boeing
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
P
PO #silver747boeing popped
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
P
PO #blueR8audi popped
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
P
PO #red911porsche popped
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
P
stack already empty
Please enter A to add a purchase order,
P to process a PO, or Q to quit.
Q
Bye
```

```
char * po;
```

The idea is to use a `char` pointer instead o
input. This approach fails immediately becaus
space to hold the input strings. (The program
crash after `cin` tried to store input in some in
Version 2 replaces

```
string po;
```

with

```
char po[40];
```

This allocates space for an input string. Fu
placed on the stack. But an array is fundamen
the `pop()` method:

```
template <class Type>
bool Stack<Type>::pop(Type & item)
{
    if (top > 0)
    {
        item = items[--top];
        return true;
    }
```

Using a Stack of Pointers Correctly

One way to use a stack of pointers is to have
pointers, with each pointer pointing to a diffe
then makes sense because each pointer will re
responsibility of the calling program, not the s
stack's job is to manage the pointers, not creat

For example, suppose you have to simulate
ered a cart of folders to Plodson. If Plodson's
from the cart and places it in his in-basket. If
file from the basket, processes it, and places it
empty nor full, Plodson may process the top f
file from the cart and put it into his in-basket
cap self-expression, he flips a coin to decide w
investigate the effects of his method on the or

You can model this with an array of point
cart. Each string will contain the name of the
stack to represent the in-basket, and you can u
the out-basket. Adding a file to the in-basket
input array onto the stack, and processing a fi
the stack and adding it to the out-basket.

```
    Stack(const Stack & st);
    ~Stack() { delete [] items; }
    bool isempty() { return top == 0; }
    bool isfull() { return top == stacksi
    bool push(const Type & item);   // ad
    bool pop(Type & item);          // po
    Stack & operator=(const Stack & st);
};

template <class Type>
Stack<Type>::Stack(int ss) : stacksize(ss
{
    items = new Type [stacksize];
}

template <class Type>
Stack<Type>::Stack(const Stack & st)
{
    stacksize = st.stacksize;
    top = st.top;
    items = new Type [stacksize];
    for (int i = 0; i < top; i++)
        items[i] = st.items[i];
}
```

```
template <class Type>
Stack<Type> & Stack<Type>::operator=(const
{
    if (this == &st)
        return *this;
    delete [] items;
    stacksize = st.stacksize;
    top = st.top;
    items = new Type [stacksize];
    for (int i = 0; i < top; i++)
        items[i] = st.items[i];
    return *this;
}

#endif
```

Notice that the prototype declares the retu
tion to be a reference to `Stack`, and the actua
type as `Stack<Type>`. The former is an abbrev
within the class scope. That is, you can use `St`
inside the template function definitions, but o

```
// in basket
    const char * in[Num] = {
            " 1: Hank Gilgamesh", " 2: Ki
            " 3: Betty Rocker", " 4: Ian
            " 5: Wolfgang Kibble", " 6: P
            " 7: Joy Almondo", " 8: Xaver
            " 9: Juan Moore", "10: Misha
            };
 // out basket
    const char * out[Num];

    int processed = 0;
    int nextin = 0;
    while (processed < Num)
    {
        if (st.isempty())
            st.push(in[nextin++]);
        else if (st.isfull())
            st.pop(out[processed++]);
        else if (std::rand() % 2  && next
            st.push(in[nextin++]);
        else
            st.pop(out[processed++]);
```

```
Bye

Please enter stack size: 5
 3: Betty Rocker
 5: Wolfgang Kibble
 6: Portia Koop
 4: Ian Flagranti
 8: Xaverie Paprika
 9: Juan Moore
10: Misha Mache
 7: Joy Almondo
 2: Kiki Ishtar
 1: Hank Gilgamesh
Bye
```

Program Notes

The strings in Listing 14.16 never move. Push
new pointer to an existing string. That is, it cr
an existing string. And popping a string off th
`out` array.

The program uses `const char *` as a type
to a set of string constants.

```
//arraytp.h  -- Array Template
#ifndef ARRAYTP_H_
#define ARRAYTP_H_

#include <iostream>
#include <cstdlib>

template <class T, int n>
class ArrayTP
{
private:
    T ar[n];
public:
    ArrayTP() {};
    explicit ArrayTP(const T & v);
    virtual T & operator[](int i);
    virtual T operator[](int i) const;
};

template <class T, int n>
ArrayTP<T,n>::ArrayTP(const T & v)
{
    for (int i = 0; i < n; i++)
        ar[i] = v;
}
```

```
#endif
```

Note the template heading in Listing 14.1

```
template <class T, int n>
```

The keyword `class` (or, equivalently in thi
parameter, or type argument. `int` identifies `n` a
parameter, one that specifies a particular type i
is called a *non-type*, or *expression*, *argument*. Sup

```
ArrayTP<double, 12> eggweights;
```

This causes the compiler to define a class c
`eggweights` object of that class. When definin
`double` and `n` with `12`.

Expression arguments have some restrictio
ger type, an enumeration type, a reference, or
`double & rm` and `double * pm` are allowed. A
of the argument or take its address. Thus, in th
or `&n` would not be allowed. Also when you i
expression argument should be a constant exp

Template Versatility

You can apply the same techniques to templa
plate classes can serve as base classes, and they
selves be type arguments to other templates.
template by using an array template. Or you
construct an array whose elements are stacks
have code along the following lines:

```
template <typename T>   // or <class T>
class Array
{
private:
    T entry;
    ...
};

template <typename Type>
class GrowArray : public Array<Type> {...

template <typename Tp>
class Stack
{
      Array<Tp> ar;       // use an Arra
     ...
```

cout.width(2) causes the next item to be di
ters, unless a larger width is needed to show t

Listing 14.18 **twod.cpp**

```
// twod.cpp -- making a 2-d array
#include <iostream>
#include "arraytp.h"
int main(void)
{
    using std::cout;
    using std::endl;
    ArrayTP<int, 10> sums;
    ArrayTP<double, 10> aves;
    ArrayTP< ArrayTP<int,5>, 10> twodee;


    int i, j;

    for (i = 0; i < 10; i++)
    {
        sums[i] = 0;
        for (j = 0; j < 5; j++)
        {
            twodee[i][j] = (i + 1) * (j +
```

of twodee, each of which is a five-element ar
average of an element of twodee:

```
 1  2  3  4  5 : sum =  15, average = 1.5
 2  4  6  8 10 : sum =  30, average = 3
 3  6  9 12 15 : sum =  45, average = 4.5
 4  8 12 16 20 : sum =  60, average = 6
 5 10 15 20 25 : sum =  75, average = 7.5
 6 12 18 24 30 : sum =  90, average = 9
 7 14 21 28 35 : sum = 105, average = 10.
 8 16 24 32 40 : sum = 120, average = 12
 9 18 27 36 45 : sum = 135, average = 13.
10 20 30 40 50 : sum = 150, average = 15
Done.
```

Using More Than One Type Parameter

You can have a template with more than one
want a class that holds two kinds of values. Yo
for holding two disparate values. (Incidentally
pair.) The short program in Listing 14.19 sh
and second() const methods report the sto
methods, by virtue of returning references to
the stored values by using assignment.

```
    return a;
}
template<class T1, class T2>
T2 & Pair<T1,T2>::second()
{
    return b;
}

int main()
{
    using std::cout;
    using std::endl;
    using std::string;
    Pair<string, int> ratings[4] =
    {
        Pair<string, int>("The Purpled Duc
        Pair<string, int>("Jaquie's Frisco
        Pair<string, int>("Cafe Souffle", 
        Pair<string, int>("Bertie's Eats",
    };

    int joints = sizeof(ratings) / sizeof
    cout << "Rating:\t Eatery\n";
    for (int i = 0; i < joints; i++)
```

```
6:      Bertie's Tab Eats
```

Default Type Template Parameters

Another new class template feature is that yo
parameters:

```
template <class T1, class T2 = int> class
```

This causes the compiler to use int for th

```
Topo<double, double> m1; // T1 is double,
Topo<double> m2;         // T1 is double,
```

The STL (discussed in Chapter 16) often
a class.

Although you can provide default values f
do so for function template parameters. How
type parameters for both class and function t

Template Specializations

Class templates are like function templates in
explicit instantiations, and explicit specializat

```
template class ArrayTP<string, 100>; // ge
```

In this case, the compiler generates the clas even though no object of the class has yet bee implicit instantiation, the general template is u specialization.

Explicit Specializations

An *explicit specialization* is a definition for a pa instead of the general template. Sometimes yo to behave differently when instantiated for a p explicit specialization. Suppose, for example, t represents a sorted array for which items are s

```
template <typename T>
class SortedArray
{
     ...// details omitted
};
```

Also suppose the template uses the `>` opera numbers. It will work if `T` represents a class ty `T::operator>()` method. But it won't work `char *`. Actually, the template will work, but

```
SortedArray<int> scores;          // use
SortedArray<const char *> dates;  // use
```

Partial Specializations

C++ allows for *partial specializations*, which p
For example, a partial specialization can prov
parameters:

```
// general template
    template <class T1, class T2> class P
// specialization with T2 set to int
    template <class T1> class Pair<T1, in
```

The `<>` following the keyword `template` c
unspecialized. So the second declaration speci
specifying all the types leads to an empty brac

```
// specialization with T1 and T2 set to i
    template <> class Pair<int, int> {...
```

The compiler uses the most specialized te
would happen given the preceding three tem

```
    template <class T1, class T2, class T3
// specialization with T3 set to T2
    template <class T1, class T2> class Tr
// specialization with T3 and T2 set to T1
    template <class T1> class Trio<T1, T1*
```

Given these declarations, the compiler wou

```
Trio<int, short, char *> t1; // use genera
Trio<int, short> t2; // use Trio<T1, T2, T
Trio<char, char *, char *> t3; use Trio<T1
```

Member Templates

A template can be a member of a structure, cl
feature to fully implement its design. Listing 1
class with a nested template class and a templa

Listing 14.20 **tempmemb.cpp**

```
// tempmemb.cpp -- template members
#include <iostream>
using std::cout;
using std::endl;
```

```
};

int main()
{
    beta<double> guy(3.5, 3);
    cout << "T was set to double\n";
    guy.Show();
    cout << "V was set to T, which is dou
    cout << guy.blab(10, 2.3) << endl;
    cout << "U was set to int\n";
    cout << guy.blab(10.0, 2.3) << endl;
    cout << "U was set to double\n";
    cout << "Done\n";
    return 0;
}
```

The hold template is declared in the priv
only within the beta class scope. The beta c
data members:

```
hold<T> q;              // template object
hold<int> n;            // template object
```

```
28.2609
U was set to double
Done
```

Note that replacing 10 with 10.0 in the ca
double, making the return type double, whic

As mentioned previously, the type of the se
declaration of the guy object. Unlike the first
parameter is not set by the function call. For i
implement blah() as blah(int, double), an
by the usual function prototype rules:

```
cout << guy.blab(10, 3) << endl;
```

You can declare the hold class and blah m
them outside the beta template. However, suf
plate members at all, and others that accept th
definitions outside the class. However, if your
defining the template methods outside the be

```
template <typename T>
class beta
{
private:
    template <typename V>  // declaration
```

```
// member definition
template <typename T>
  template <typename U>
    U beta<T>::blab(U u, T t)
    {
       return (n.Value() + q.Value()) * u
    }
```

The definitions have to identify `T`, `V`, and `U`
plates are nested, you have to use the

```
template <typename T>
  template <typename V>
```

syntax instead of this syntax:

```
template<typename T, typename V>
```

The definitions also must indicate that `ho`
class, and they use the scope-resolution opera

Templates As Parameters

You've seen that a template can have type pa
type parameters, such as `int n`. A template c

declaration:

```
Crab<Stack> nebula;
```

Hence, in this case, `Thing<int>` is instantia
instantiated as `Stack<double>`. In short, the te
whatever template type is used as a template a

The `Crab` class declaration makes three fur
represented by `Thing`. The class should have a
`pop()` method, and these methods should hav
use any template class that matches the `Thing`
site `push()` and `pop()` methods. This chapter
template defined in `stacktp.h`, so the examp

Listing 14.21 **tempparm.cpp**

```
// tempparm.cpp - templates as parameters
#include <iostream>
#include "stacktp.h"

template <template <typename T> class Thin
class Crab
{
private:
    Thing<int> s1;
```

```
    while (nebula.pop(ni, nb))
           cout << ni << ", " << nb << en
    cout << "Done.\n";

    return 0;
}
```

Here is a sample run of the program in Li

```
Enter int double pairs, such as 4 3.5 (0
50 22.48
25 33.87
60 19.12
0 0
60, 19.12
25, 33.87
50, 22.48
Done.
```

You can mix template parameters with re declaration could start out like this:

```
template <template <typename T> class Thi
class Crab
```

- Unbound template friends, meaning th
to each specialization of the class

Let's look at examples of each.

Non-Template Friend Functions to Templ

Let's declare an ordinary function in a templa

```
template <class T>
class HasFriend
{
public:
    friend void counts();     // friend to
    ...
};
```

This declaration makes the `counts()` func
the template. For example, it would be a frien
`HasFriend<string>` class.

The `counts()` function is not invoked by
tion), and it has no object parameters, so how
are several possibilities. It could access a globa
using a global pointer; it could create its own
bers of a template class, which exist separately

```
    friend void report(HasFriend<int> &);
    ...
};
```

That is, report() with a HasFriend<int

HasFriend<int> class. Similarly, report() w

be an overloaded version of report() that is

Note that report() is not itself a templat

template. This means that you have to define

plan to use:

```
void report(HasFriend<short> &) {...}; //
void report(HasFriend<int> &) {...};   //
```

Listing 14.22 illustrates these points. The

Note that this means that each particular spe

member. The counts() method, which is a f

reports the value of ct for two particular spe

HasFriend<double>. The program also provi

is a friend to one particular HasFriend speci

```
// non-template friend to all HasFriend<T>
void counts()
{
    cout << "int count: " << HasFriend<int
    cout << "double count: " << HasFriend<
}

// non-template friend to the HasFriend<in
void reports(HasFriend<int> & hf)
{
    cout <<"HasFriend<int>: " << hf.item <
}

// non-template friend to the HasFriend<do
void reports(HasFriend<double> & hf)
{
    cout <<"HasFriend<double>: " << hf.ite
}

int main()
{
    cout << "No objects declared: ";
    counts();
```

```
HasFriend<int>: 10
HasFriend<int>: 20
HasFriend<double>: 10.5
```

Bound Template Friend Functions to Te

You can modify the preceding example by m
selves. In particular, you can set things up for
tion of a class gets a matching specialization f
complex than for non-template friends and i

For the first step, you declare each templa

```
template <typename T> void counts();
template <typename T> void report(T &);
```

Next, you declare the templates again as f
declare specializations based on the class tem

```
template <typename TT>
class HasFriendT
{
...
    friend void counts<TT>();
    friend void report<>(HasFriendT<TT> &
};
```

```
};
```

One specialization is based on TT, which b
HasFriendT<TT>, which becomes HasFriend
counts<int>() and report<HasFriendT<in
HasFriendT<int> class.

The third requirement the program must r
the friends. Listing 14.23 illustrates these thre
count() function that is a friend to all HasFr
count() functions, one of which is a friend t
the count() function calls have no function p
deduce the desired specialization, these calls u
forms to indicate the specialization. For the ca
use the argument type to deduce the speciali
same effect:

```
report<HasFriendT<int> >(hfi2);  // same a
```

Listing 14.23 **tmp2tmp.cpp**

```
// tmp2tmp.cpp -- template friends to a te
#include <iostream>
using std::cout;
using std::endl;
```

```
template <typename T>
void counts()
{
    cout << "template size: " << sizeof(H
    cout << "template counts(): " << HasF
}

template <typename T>
void report(T & hf)
{
    cout << hf.item << endl;
}

int main()
{
    counts<int>();
    HasFriendT<int> hfi1(10);
    HasFriendT<int> hfi2(20);
    HasFriendT<double> hfdb(10.5);
    report(hfi1);  // generate report(Has
    report(hfi2);  // generate report(Has
    report(hfdb);  // generate report(Has
    cout << "counts<int>() output:\n";
    counts<int>();
```

of a template declared outside a class. An `int`
cialization, and so on. By declaring a template
friend functions for which every function spe
ization. For unbound friends, the friend temp
template class type parameters:

```
template <typename T>
class ManyFriend
{
...
    template <typename C, typename D> frie
};
```

Listing 14.24 shows an example that uses a
`show2(hfi1, hfi2)` gets matched to the foll

```
void show2<ManyFriend<int> &, ManyFriend<i
          (ManyFriend<int> & c, ManyFrienc
```

Because it is a friend to all specializations o
the `item` members of all specializations. But i
objects.

Similarly, `show2(hfd, hfi2)` gets matched

```
void show2<ManyFriend<double> &, ManyFrien
          (ManyFriend<double> & c, ManyFri
```

```
    cout << c.item << ", " << d.item << e
}

int main()
{
    ManyFriend<int> hfi1(10);
    ManyFriend<int> hfi2(20);
    ManyFriend<double> hfdb(10.5);
    cout << "hfi1, hfi2: ";
    show2(hfi1, hfi2);
    cout << "hfdb, hfi2: ";
    show2(hfdb, hfi2);

    return 0;
}
```

Here's the output of the program in Listin

```
hfi1, hfi2: 10, 20
hfdb, hfi2: 10.5, 20
```

```
arrtype<int> days;          // days is t
arrtype<std::string> months;  // months is
```

In short, `arrtype<T>` means type `std::ar`

C++11 extends the `using =` syntax to no
equivalent to an ordinary `typedef`:

```
typedef const char * pc1;       // typedef
using pc2 = const char *;       // using =
typedef const int *(*pa1)[10];  // typedef
using pa2 = const int *(*)[10]; // using =
```

As you get used to it, you may find the ne
the type name from type information more c

Another C++11 addition to templates is t
define a template class or function that can ta
18, "Visiting with the New C++ Standard," l

Summary

C++ provides several means for reusing code
13, "Class Inheritance," enables you to model
able to reuse the code of base classes. Private a
base-class code, this time modeling *has-a* relat

bases. You can use class qualifiers to resolve n
avoid multi-inherited bases. However, using v
writing initialization lists for constructors an

Class templates let you create a generic cl
type, is represented by a type parameter. A ty

```
template <class T>
class Ic
{
     T v;
     ...
public:
     Ic(const T & val) : v(val) { }
     ...
};
```

Here `T` is the type parameter, and it acts a
later time. (This parameter can have any vali
choices.) You can also use `typename` instead

```
template <typename T>  // same as templat
class Rev {...} ;
```

Class definitions (instantiations) are gener
specify a particular type. For example, the fol

```
public:
      Ic(const char * s) : str(s) { }
      ...
};
```

Then a declaration of the following form
rather than using the general template:

```
class Ic<char *> chic;
```

A class template can specify more than one
parameters:

```
template <class T, class TT, int n>
class Pals {...};
```

The following declaration would generate
`string` for `TT`, and `6` for `n`:

```
Pals<double, string, 6> mix;
```

A class template can also have parameters

```
template < template <typename T> class CL,
class Trophy {...};
```

Here `z` stands for an `int` value, `U` stands for
template declared using `template <typename`

```
class Person
class Person
class Person, class Automobile
```

2. Suppose you have the following defini

```
class Frabjous {
private:
      char fab[20];
public:
      Frabjous(const char * s = "C+
      virtual void tell() { cout <<
};

class Gloam {
private:
      int glip;
      Frabjous fb;
public:
      Gloam(int g = 0, const char *
      Gloam(int g, const Frabjous &
      void tell();
};
```

provide definitions for the three `Gloam`

4. Suppose you have the following definit
14.13 and the `Worker` class of Listing 1

```
Stack<Worker *> sw;
```

Write out the class declaration that will
not the non-inline class methods.

5. Use the template definitions in this cha

   - An array of `string` objects
   - A stack of arrays of `double`
   - An array of stacks of pointers to `W`

   How many template class definitions ar

6. Describe the differences between virtua

```
Wine(const char * l, int y);
```

The `Wine` class should have a method `G`
`y` years, prompts the user to enter the c
bottle counts. A method `Label()` shou
method `sum()` should return the total
`valarray<int>` object in the `Pair` obj

The program should prompt the user t
ments of the array, and the year and bo
ment. The program should use this data
the information stored in the object. F

```
// pe14-1.cpp  -- using Wine class w
#include <iostream>
#include "winec.h"

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

    cout << "Enter name of wine: ";
    char lab[50];
```

```
Enter name of wine: Gully Wash
Enter number of years: 4
Enter Gully Wash data for 4 year(s):
Enter year: 1988
Enter bottles for that year: 42
Enter year: 1994
Enter bottles for that year: 58
Enter year: 1998
Enter bottles for that year: 122
Enter year: 2001
Enter bottles for that year: 144
Wine: Gully Wash
        Year    Bottles
        1988    42
        1994    58
        1998    122
        2001    144
Wine: Gushing Grape Red
        Year    Bottles
        1993    48
        1995    60
        1998    72
Total bottles for Gushing Grape Red:
Bye
```

value. (Optionally, you could define a C
and use a Card return value for Draw()
show() function. The BadDude class de
PokerPlayer classes. It has a Gdraw() m
and a Cdraw() member that returns th
Show() function. Define all these classe
sary methods (such as methods for sett
program similar to that in Listing 14.12

5. Here are some class declarations:

```
// emp.h -- header file for abstr_em

#include <iostream>
#include <string>

class abstr_emp
{
private:
    std::string fname;    // abstr_e
    std::string lname;    // abstr_e
    std::string job;
```

```
{
private:
    int inchargeof;        // number
protected:
    int InChargeOf() const { return
    int & InChargeOf(){ return incha
public:
    manager();
    manager(const std::string & fn,
            const std::string & j, i
    manager(const abstr_emp & e, int
    manager(const manager & m);
    virtual void ShowAll() const;
    virtual void SetAll();
};

class fink: virtual public abstr_emp
{
private:
    std::string reportsto;        //
protected:
    const std::string ReportsTo() co
    std::string & ReportsTo(){ retur
```

```
};
```

Note that the class hierarchy uses MI v
special rules for constructor initializatio
of some protected-access methods. Thi
`highfink` methods. (Note, for example
calls `fink::ShowAll()` and `manager::`
`abstr_emp::ShowAll()` twice.) Provic
the classes in a program. Here is a mini

```
// pe14-5.cpp
// useemp1.cpp -- using the abstr_em

#include <iostream>
using namespace std;
#include "emp.h"

int main(void)
{
    employee em("Trip", "Harris", "'
    cout << em << endl;
    em.ShowAll();
```

Why is no assignment operator defined

Why are ShowAll() and SetAll() virt

Why is abstr_emp a virtual base class?

Why does the highfink class have no c

Why is only one version of operator<

What would happen if the end of the p

```
abstr_emp  tri[4] = {em, fi, hf, hf2
for (int i = 0; i < 4; i++)
     tri[i].ShowAll();
```

- Runtime type identification (RTTI)
- `dynamic_cast` and `typeid`
- `static_cast`, `const_cast`, and `reinte`

This chapter ties up some loose ends and
the C++ language. The loose ends include fr
nested classes, which are classes declared with
here are exceptions, runtime type identificati
C++ exception handling provides a mechan
that otherwise would bring a program to a h
object types. The new type cast operators im
facilities are fairly new to C++, and older co

Friends

Several examples in this book so far use frien
for a class. Such functions are not the only ki
can be a friend. In that case, any method of t
tected members of the original class. Also yo
particular member functions of a class to be f
functions, member functions, or classes are fr
outside. Thus, although friends do grant outs

- Cable or antenna tuning mode
- TV tuner or A/V input

The tuning mode reflects the fact that, in t
channels for channels 14 and up is different fo
cast reception. The input selection chooses be
broadcast TV, and a DVD. Some sets may offer
ray inputs, but this list is enough for the purpo

Also a television has some parameters that
sions vary in the number of channels they can
track that value.

Next, you must provide the class with met
sion sets these days hide their controls behind
visions to change channels, and so on, withou
go up or down one channel at a time but can
there's usually a button for increasing the volu

A remote control should duplicate the con
methods can be implemented by using `Tv` me
cally provides random access channel selection
2 to channel 20 without going through all the
can work in two or more modes, for example
controller.

```
    enum {Antenna, Cable};
    enum {TV, DVD};

    Tv(int s = Off, int mc = 125) : state
        maxchannel(mc), channel(2), mode(
    void onoff() {state = (state == On)?
    bool ison() const {return state == On
    bool volup();
    bool voldown();
    void chanup();
    void chandown();
    void set_mode() {mode = (mode == Ante
    void set_input() {input = (input == T
    void settings() const; // display all
private:
    int state;             // on or off
    int volume;            // assumed to
    int maxchannel;        // maximum num
    int channel;           // current cha
    int mode;              // broadcast c
    int input;             // TV or DVD
};
```

use wraparound, with the lowest channel setti
highest channel setting, maxchannel.

Many of the methods use the conditional
settings:

```
void onoff() {state = (state == On)? Off :
```

Provided that the two state values are true
can be done more compactly by using the co
ment operator (^=), as discussed in Appendix

```
void onoff() {state ^= 1;}
```

In fact, you could store up to eight bivalen
variable and toggle them individually, but that
bitwise operators discussed in Appendix E.

Listing 15.2 **tv.cpp**

```
// tv.cpp -- methods for the Tv class (Rem
#include <iostream>
#include "tv.h"

bool Tv::volup()
{
```

```
    else
        channel = 1;
}

void Tv::chandown()
{
    if (channel > 1)
        channel--;
    else
        channel = maxchannel;
}

void Tv::settings() const
{
    using std::cout;
    using std::endl;
    cout << "TV is " << (state == Off? "O
    if (state == On)
    {
        cout << "Volume setting = " << vo
        cout << "Channel setting = " << c
        cout << "Mode = "
            << (mode == Antenna? "antenna
```

```
    s42.settings();

    Remote grey;

    grey.set_chan(s42, 10);
    grey.volup(s42);
    grey.volup(s42);
    cout << "\n42\" settings after using r
    s42.settings();

    Tv s58(Tv::On);
    s58.set_mode();
    grey.set_chan(s58,28);
    cout << "\n58\" settings:\n";
    s58.settings();
    return 0;
}
```

Here is the output of the program in Listin

```
Initial settings for 42" TV:
TV is Off
```

make the private parts of the Tv class public o
that encompasses both a television and a rem
reflect the fact that a single remote control ca

Friend Member Functions

Looking at the code for the last example, you
ods are implemented by using the public inte
methods don't really need friend status. Indee
private Tv member directly is Remote::set_
to be a friend. You do have the option of mak
another class rather than making the entire cl
You need to be careful about the order in wh
definitions. Let's look at why.

The way to make Remote::set_chan() a
friend in the Tv class declaration:

```
class Tv
{
    friend void Remote::set_chan(Tv & t,
    ...
};
```

method in particular.

Another difficulty remains. In Listing 15.1
such as the following:

```
void onoff(Tv & t) { t.onoff(); }
```

Because this calls a `Tv` method, the compil
tion at this point so that it knows what metho
tion necessarily follows the `Remote` declaratio
`Remote` to method *declarations* and to place th
leads to the following ordering:

```
class Tv;                // forward declara
class Remote { ... };      // Tv-using meth
class Tv { ... };
// put Remote method definitions here
```

The `Remote` prototypes look like this:

```
void onoff(Tv & t);
```

All the compiler needs to know when insp
the forward declaration supplies that informat
actual method definitions, it has already read t
information needed to compile those method

```
    bool voldown(Tv & t);
    void onoff(Tv & t) ;
    void chanup(Tv & t) ;
    void chandown(Tv & t) ;
    void set_mode(Tv & t) ;
    void set_input(Tv & t);
    void set_chan(Tv & t, int c);

};

class Tv
{
public:
    friend void Remote::set_chan(Tv & t,
    enum State{Off, On};
    enum {MinVal,MaxVal = 20};
    enum {Antenna, Cable};
    enum {TV, DVD};

    Tv(int s = Off, int mc = 125) : state
        maxchannel(mc), channel(2), mode(
    void onoff() {state = (state == On)?
    bool ison() const {return state == On
    bool volup();
```

```
inline void Remote::set_chan(Tv & t, int c
 #endif
```

If you include tvfm.h instead of tv.h in t
gram behaves the same as the original. The di
instead of all the Remote methods—is a frienc
difference.

Recall that inline functions have internal l
must be in the file that uses the function. Her
file, so including the header file in the file tha
the right place. You could place the definition
vided that you remove the inline keyword, t

By the way, making the entire Remote class
because the friend statement itself identifies R

```
friend class Remote;
```

Other Friendly Relationships

Other combinations of friends and classes bes
are possible. Let's take a brief look at some of

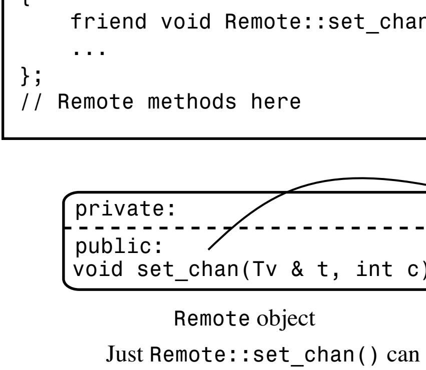


Figure 15.1 Class friends

Suppose the advance of technology bring
interactive remote control unit might let you
a television program, and the television migh
your response is wrong. Ignoring the possibil
gram the viewers, let's just look at the C++ p
benefit from mutual friendship, with some Re

```
    ...
}
```

Because the Remote declaration follows th
defined in the class declaration. However, the
side the Tv declaration so that the definition c
don't want buzz() to be inline, you need to c

Shared Friends

Another use for friends is when a function ne
classes. Logically, such a function should be a
impossible. It could be a member of one class
more reasonable to make the function a frien
have a Probe class that represents some sort o
Analyzer class that represents some sort of pr
internal clock, and you would like to be able
use something along the following lines:

```
class Analyzer;  // forward declaration
class Probe
{
    friend void sync(Analyzer & a, const P
    friend void sync(Probe & p, const Anal
    ...
```

In C++, you can place a class declaration ins
another is called a *nested class*, and it helps avo
scope. Member functions of the class contain
objects of the nested class. The outside world
tion is in the public section and if you use th
of C++ either don't allow nested classes or i

Nesting classes is not the same as contain
a class object as a member of another class. N
create a class member. Instead, it defines a typ
contains the nested class declaration.

The usual reasons for nesting a class are to
and to avoid name conflicts. The `Queue` class
"Classes and Dynamic Memory Allocation,"
nesting a structure definition:

```
class Queue
{
private:
// class scope definitions
    // Node is a nested structure definit
    struct Node {Item item; struct Node *
    ...
};
```

```
class Queue
{
// class scope definitions
    // Node is a nested class definition l
    class Node
    {
    public:
        Item item;
        Node * next;
        Node(const Item & i) : item(i), ne
    };
    ...
};
```

This constructor initializes the node's `item`
`0`, which is one way of writing the null point
including a header file that defines `NULL`. User
`nullptr`.) Because all nodes created by the `Qu`
pointer, this is the only constructor the class n

Next, you need to rewrite `enqueue()` by u

```
bool Queue::enqueue(const Item & item)
{
    if (isfull())
        return false;
```

Scope

If a nested class is declared in a private sectio
second class. This applies, for example, to the
the preceding example. Hence, `Queue` memb
`Node` objects, but other parts of a program do
you were to derive a class from `Queue`, `Node`
derived class can't directly access the private

If the nested class is declared in a protecte
class but invisible to the outside world. Howe
about the nested class and could directly crea

If a nested class is declared in a public sect
second class, to classes derived from the secon
side world. However, because the nested class
qualifier in the outside world. For example, s

```
class Team
{
public:
      class Coach { ... };
      ...
};
```

Access Control

After a class is in scope, access control comes i
nested class that govern access to a regular clas
declaration does not grant the `Queue` class any
nor does it grant the `Node` class any special acc
`Queue` class object can access only the public r
reason, the `Queue` example makes all the mem
the usual practice of making data members pr
mentation feature of the `Queue` class and is no
the `Node` class is declared in the private section
methods can access `Node` members directly, a c

In short, the location of a class declaration
Given that a particular class is in scope, the us
private, friend) determine the access a program

Nesting in a Template

You've seen that templates are a good choice
the `Queue` class. You may be wondering wheth
to converting the `Queue` class definition to a t

```
    Node * rear;        // pointer to rea
    int items;          // current number
    const int qsize;    // maximum number
    QueueTP(const QueueTP & q) : qsize(0)
    QueueTP & operator=(const QueueTP & q
public:
    QueueTP(int qs = Q_SIZE);
    ~QueueTP();
    bool isempty() const
    {
        return items == 0;
    }
    bool isfull() const
    {
        return items == qsize;
    }
    int queuecount() const
    {
        return items;
    }
    bool enqueue(const Item &item); // ad
    bool dequeue(Item &item);       // re
};
```

```
{
    if (isfull())
        return false;
    Node * add = new Node(item);    // cre
// on failure, new throws std::bad_alloc e
    items++;
    if (front == 0)         // if queue is
        front = add;        // place item
    else
        rear->next = add;   // else place
    rear = add;             // have rear p
    return true;
}

// Place front item into item variable and
template <class Item>
bool QueueTP<Item>::dequeue(Item & item)
{
    if (front == 0)
        return false;
    item = front->item;     // set item to
    items--;
    Node * temp = front;    // save locati
```

Listing 15.6 **nested.cpp**

```
// nested.cpp -- using a queue that has a
#include <iostream>

#include <string>
#include "queuetp.h"

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

    QueueTP<string> cs(5);
    string temp;

    while(!cs.isfull())
    {
        cout << "Please enter your name. 
                "served in the order of a
                "name: ";
        getline(cin, temp);
        cs.enqueue(temp);
```

```
The queue is full. Processing begins!
Now processing Kinsey Millhone...
Now processing Adam Dalgliesh...
Now processing Andrew Dalziel...
Now processing Kay Scarpetta...
Now processing Richard Jury...
```

Exceptions

Programs sometimes encounter runtime prob
normally. For example, a program may try to
more memory than is available, or it may enc
grammers try to anticipate such calamities. C
ble tool for dealing with these situations. Exc
C++, so some older compilers haven't implem
feature off by default, so you may have to use

Before examining exceptions, let's look at
available to programmers. As a test case, let's lo
monic mean of two numbers. The *harmonic m*
of the average of the inverses. This can be red

```
2.0   x   y / (x + y)
```

Listing 15.7 **error1.cpp**

```
//error1.cpp -- using the abort() functio
#include <iostream>
#include <cstdlib>
double hmean(double a, double b);

int main()
{
    double x, y, z;

    std::cout << "Enter two numbers: ";
    while (std::cin >> x >> y)
    {
        z = hmean(x,y);
        std::cout << "Harmonic mean of "
            << " is " << z << std::endl;
        std::cout << "Enter next set of n
    }
    std::cout << "Bye!\n";
    return 0;
}

double hmean(double a, double b)
{
```

(You may, perhaps, enjoy the conceit that o
program could avoid aborting by checking th
`hmean()` function. However, it's not safe to re
enough to perform such a check.

Returning an Error Code

A more flexible approach than aborting is to
problem. For example, the `get(void)` membe
ASCII code for the next input character, but
ters the end-of-file. This approach doesn't wo
a valid return value, so there's no special value
of situation, you can use a pointer argument o
to the calling program and use the function r
`istream` family of overloaded `>>` operators us
the calling program of the success or failure, y
actions other than aborting. Listing 15.8 show
`hmean()` as a `bool` function whose return valu
argument for obtaining the answer.

```
    std::cout << "Bye!\n";
    return 0;
}

bool hmean(double a, double b, double * a
{
    if (a == -b)
    {
        *ans = DBL_MAX;
        return false;
    }
    else
    {
        *ans = 2.0 * a * b / (a + b);
        return true;
    }
}
```

Here's a sample run of the program in Lis

```
Enter two numbers: 3 6
Harmonic mean of 3 and 6 is 4
Enter next set of numbers <q to quit>: 10
One value should not be the negative of t
```

The Exception Mechanism

Now let's see how you can handle problems b
exception is a response to an exceptional circu
ning, such as an attempt to divide by zero. Ex
from one part of a program to another. Handl

- Throwing an exception
- Catching an exception with a handler
- Using a `try` block

A program throws an exception when a pr
modify `hmean()` in Listing 15.7 to throw an e
tion. A throw statement, in essence, is a jump;
ments at another location. The `throw` keywor
followed by a value, such as a character string
exception.

A program catches an exception with an *e*
where you want to handle the problem. The c
exception. A handler begins with the keyword
parentheses) that indicates the type of excepti
lowed by a brace-enclosed block of code that
keyword, along with the exception type, serve

```
        try {                    // start
            z = hmean(x,y);
        }                        // end of
        catch (const char * s)  // start
        {
            std::cout << s << std::endl;
            std::cout << "Enter a new pai
            continue;
        }                        // end of
        std::cout << "Harmonic mean of "
            << " is " << z << std::endl;
        std::cout << "Enter next set of n
    }
    std::cout << "Bye!\n";
    return 0;
}

double hmean(double a, double b)
{
    if (a == -b)
        throw "bad hmean() arguments: a =
    return 2.0 * a * b / (a + b);
}
```

Throwing an exception looks like this:

```
if (a == -b)
    throw "bad hmean() arguments: a = -b n
```

In this case, the thrown exception is the str
`not allowed"`. The exception type can be a s
class type is the usual choice, as later examples

Executing the `throw` is a bit like executing
function execution. However, instead of retur
causes a program to back up through the sequ
the function that contains the `try` block. In L
calling function. Soon you'll see an example i
tion. Meanwhile, in this case, the `throw` passes
program looks for an exception handler (follo
of exception thrown.

The handler, or `catch` block, looks like thi

```
catch (char * s)  // start of exception ha
{
    std::cout << s << std::endl;
    sdt::cout << "Enter a new pair of numb
    continue;
}                       // end of handler
```

loop illustrates the fact that handler statemen
line acts like a label directing program flow (

You might wonder what happens if a func
block or else no matching handler. By defaul
function, but you can modify that behavior. V

Using Objects as Exceptions

Typically, functions that throw exceptions thr
is that you can use different exception types
situations that produce exceptions. Also an ob
can use this information to help identify the
thrown. Also in principle a `catch` block coul
course of action to pursue. Here, for example
be thrown by the `hmean()` function:

```
class bad_hmean
{
private:
    double v1;
    double v2;
public:
    bad_hmean(int a = 0, int b = 0) : v1(
```

```
    cout << "Enter next set of
}
...
double hmean(double a, double b
{
    if (a == -b)
        throw "bad hmean() argu
    return 2.0 * a * b / (a + b
}
```

1. The program calls hmean() within a try

2. hmean() throws an exception, transferri
   assigning the exception string to s.

3. The catch block transfers execution back

Figure 15.2 Program

A bad_hmean object can be initialized to th
method can be used to report the problem, in
can use code like this:

```
if (a == -b)
    throw bad_hmean(a,b);
```

gmean() throws a bad_gmean exception, the
and gets caught by the second.

Listing 15.10 **exc_mean.h**

```
// exc_mean.h  -- exception classes for h
#include <iostream>

class bad_hmean
{
private:
    double v1;
    double v2;
public:
    bad_hmean(double a = 0, double b = 0)
    void mesg();
};

inline void bad_hmean::mesg()
{
    std::cout << "hmean(" << v1 << ", " <
              << "invalid arguments: a =
}
```

```
double gmean(double a, double b);
int main()
{
    using std::cout;
    using std::cin;
    using std::endl;

    double x, y, z;

    cout << "Enter two numbers: ";
    while (cin >> x >> y)
    {
        try {                   // start of
            z = hmean(x,y);
            cout << "Harmonic mean of " <<
                << " is " << z << endl;
            cout << "Geometric mean of " <
                << " is " << gmean(x,y) <<
            cout << "Enter next set of nu
        }// end of try block
        catch (bad_hmean & bg)    // start
        {
            bg.mesg();
            cout << "Try again.\n";
```

```
{
    if (a < 0 || b < 0)
        throw bad_gmean(a,b);
    return std::sqrt(a * b);
}
```

Here's a sample run of the program, one t
`gmean()` function:

```
Enter two numbers: 4 12
Harmonic mean of 4 and 12 is 6
Geometric mean of 4 and 12 is 6.9282
Enter next set of numbers <q to quit>: 5
hmean(5, -5): invalid arguments: a = -b
Try again.
5 -2
Harmonic mean of 5 and -2 is -6.66667
gmean() arguments should be >= 0
Values used: 5, -2
Sorry, you don't get to play any more.
Bye!
```

One point to notice is that the `bad_hmear`
the `bad_gmean` handler uses a `break` statemer

try block. However, that can be accomplishe
reason was to allow the compiler to add code
tion specification was violated. This can happe
not throw an exception, but it might call a fu
exception. And maybe that function didn't th
but after a library update, it now does. Anywa
gramming community, particularly among th
exception-safe code, was that this feature is be
exception specifications with the blessings of

However, C++11 does allow for one speci
noexcept can be used to indicate a function

```
double marm() noexcept;  // marm() doesn't
```

There is some debate about the necessity a
some feeling it's better to avoid using it (at le
enough about the need to introduce a new k
function shouldn't throw an exception can he
should be thought of as a promise made by th

There also is a noexcept() operator (see A
its operand could throw an exception.

return address on the stack, the program con
return address that resides in a `try` block (see
exception handlers at the end of the block ra
function call. This process is called *unwinding*
`throw` mechanism is that, just as with functio
any automatic class objects on the stack. How
put on the stack by that function, whereas th
the stack by the entire sequence of function
Without the unwinding-the-stack feature, a
automatic class objects placed on the stack by

Listing 15.12 provides an example of unw
which in turn calls `hmean()` and `gmean()`. Th
better to do, calculates the mean of the arithr
`main()` and `means()` create objects of the `de`
when its constructor and destructor are used
objects when exceptions are thrown. The `try`
and `bad_gmean` exceptions, and the `try` bloc
exception. This `catch` block has the followin

```
catch (bad_hmean & bg) // start of catch
{
    bg.mesg();
    std::cout << "Caught in means()\n";
    throw;              // rethrows the ex
}
```

```
{
   ...
}
```

usi

Figure 15.3 thr

After the code responds by displaying mess
means, in this case, sending the exception on
exception rises to the next try-catch combi
exception. If no handler is found, the program
same header file (exc_mean.h in listing 15.10

Listing 15.12 **error5.cpp**

```
//error5.cpp -- unwinding the stack
#include <iostream>
#include <cmath> // or math.h, unix users
#include <string>
#include "exc_mean.h"

class demo
{
private:
    std::string word;
public:
    demo (const std::string & str)
```

```
using std::cin;
using std::endl;

double x, y, z;
{
    demo d1("found in block in main()
    cout << "Enter two numbers: ";
    while (cin >> x >> y)
    {
           try {                    //
               z = means(x,y);
               cout << "The mean mean
                       << " is " << z
               cout << "Enter next pa
           } // end of try block
           catch (bad_hmean & bg)
           {
               bg.mesg();
               cout << "Try again.\n"
               continue;
           }
           catch (bad_gmean & hg)
           {
               cout << hg.mesg();
```

```
    if (a < 0 || b < 0)
        throw bad_gmean(a,b);
    return std::sqrt(a * b);
}

double means(double a, double b)
{
    double am, hm, gm;
    demo d2("found in means()");
    am = (a + b) / 2.0;    // arithmetic m
    try
    {
        hm = hmean(a,b);
        gm = gmean(a,b);
    }
    catch (bad_hmean & bg) // start of cat
    {
        bg.mesg();
        std::cout << "Caught in means()\n'
        throw;             // rethrows the
    }
    d2.show();
    return (am + hm + gm) / 3.0;
}
```

```
Bye!
```

Program Notes

Let's trace through the course of the sample
the demo constructor announces, an object is
means() is called, and another demo object is
values 6 and 12 on to hmean() and gmean(),
means(), which calculates a result and return
invokes d2.show(). After returning the result
d2 is called automatically:

```
demo found in means() lives!
demo found in means() destroyed
```

The next input cycle sends the values 6 an
a new demo object and relays the values to hm
bad_hmean exception, which is caught by the
following output:

```
hmean(6, -6): invalid arguments: a = -b
Caught in means()
```

```
demo found in means() destroyed
```

Finally, the bad_gmean handler in main() c

```
gmean() arguments should be >= 0
Values used: 6, -8
Sorry, you don't get to play any more.
```

Then the program terminates normally, di
calling the destructor for d1. If the catch bloc
break, the program would terminate immedi

```
demo found in main() lives!
Bye!
```

However, you would still see this message:

```
demo found in main() destroyed
```

Again, the exception mechanism would at
stack.

More Exception Features

Although the throw-catch mechanism is sim
return mechanism, there are a few differences
that a return statement in a function fun() tr

```
catch(problem & p)
{
// statements
}
```

Here, p would refer to a copy of oops rath oops no longer exists after super() terminat struction with the throw:

```
throw problem();     // construct and thr
```

At this point you might wonder why the a copy. After all, the usual reason for using a not having to make a copy. The answer is tha erty: A base-class reference can also refer to c collection of exception types that are related specification need only list a reference to the derived objects thrown.

Suppose you have a class hierarchy of exce types separately. A base-class reference can ca object can only catch that object and objects object is caught by the first catch block that catch blocks in inverse order of derivation:

```
{ // statements }
```

If the bad_1 & handler were first, it would tions. With the inverse order, a bad_3 excepti handler.

> **Tip**
>
> If you have an inheritance hierarchy of except the catch blocks so that the exception of th thest down the class hierarchy sequence) is caught last.

Arranging catch blocks in the proper sequ each type of exception is handled. But someti exception to expect. For instance, say you wri you don't know whether that other function exception even if you don't know the type. Th an ellipsis for the exception type:

```
catch (...) { // statements }  // catches
```

The main intent for C++ exceptions is to pr
fault-tolerant programs. That is, exceptions m
into a program design so you don't have to ta
dling as an afterthought. The flexibility and re
encourage programmers to integrate fault han
appropriate. In short, exceptions are the kind
approach to programming.

Newer C++ compilers are incorporating
the `exception` header file (formerly `excepti`
class that C++ uses as a base class for other ex
Your code, too, can throw an `exception` obje
One virtual member function is named `what`
which is implementation dependent. Howeve
redefine it in a class derived from `exception`

```
#include <exception>
class bad_hmean : public std::exception
{
public:
    const char * what() { return "bad arg
...
};
class bad_gmean : public std::exception
{
```

```
class logic_error : public exception {
public:
explicit logic_error(const string& what_ar
...
};

class domain_error : public logic_error {
public:
explicit domain_error(const string& what_a
...
};
```

Note that the constructors take a `string` o
the character data returned as a C-style string

These two new classes serve, in turn, as bas
`logic_error` family describes, as you might e
sound programming could avoid such errors, l
The name of each class indicates the sort of er

```
domain_error
invalid_argument
length_error
out_of_bounds
```

out_of_bounds exception if the index used i

Next, the runtime_error family describe but that could not easily be predicted and pre sort of error it is intended to report:

```
range_error
overflow_error
underflow_error
```

Each class has a constructor like that of ru string to be returned by the what() method.

An underflow error can occur in floating- smallest nonzero magnitude that a floating-po would produce a smaller value would cause a occur with either integer or floating-point ty calculation would exceed the largest represen result can lie outside the valid range of a func flow, and you can use the range_error excep

In general, an exception of the logic_err tible to a programming fix, whereas a runtim unavoidable trouble. All these error classes ha distinction is that the different class names all

throw a bad_alloc exception. The new header file includes a declaration for the bad_alloc class, which is publicly derived from the exce
returned a null pointer when it couldn't alloc

Listing 15.13 demonstrates the current app
gram displays the implementation-dependent
method and terminates early.

Listing 15.13 **newexcp.cpp**

```
// newexcp.cpp -- the bad_alloc exception
#include <iostream>
#include <new>
#include <cstdlib>  // for exit(), EXIT_FA
using namespace std;

struct Big
{
    double stuff[20000];
};

int main()
{
    Big * pb;
    try {
```

In this case, the what() method returns th

If the program runs without allocation pro
the amount of memory requested.

The Null Pointer and `new`

Much code was written when new (the old n
Some compilers handled the transition to the
switch to choose which behavior she wanted
native form of new that still returns a null poi

```
int * pi = new (std::nothrow) int;
int * pa = new (std::nowthrow) int[500];
```

Using this form, you could rewrite the co

```
Big * pb;


pb = new (std::nothrow) Big[10000]; // 1,
if (pb == 0)
{
    cout << "Could not allocate memory. B
    exit(EXIT_FAILURE);
}
```

```
    private:
        int bi;  // bad index value
    public:
        explicit bad_index(int ix,
            const std::string & s = "Index
        int bi_val() const {return bi;}
        virtual ~bad_index() throw() {}
    };
    explicit Sales(int yy = 0);
    Sales(int yy, const double * gr, int n
    virtual ~Sales() { }
    int Year() const { return year; }
    virtual double operator[](int i) const
    virtual double & operator[](int i);
private:
    double gross[MONTHS];
    int year;
};

class LabeledSales : public Sales
{
  public:
    class nbad_index : public Sales::bad_i
    {
```

available as a type to client `catch` blocks. No
be identified as `Sales::bad_index`. This class
class. The `bad_index` class has the ability to st
array index.

The `nbad_index` class is nested in the pub
able to client code as `LabeledSales::nbad_`
ability to store and report the label of a `Label`
from `logic_error`, `nbad_index` also ultimate

Both classes have overloaded `operator[]`
individual array elements stored in an object
of bounds.

Both the `bad_index` and `nbad_index` clas
The reason is that both eventually inherit fro
destructor uses the `throw()` exception specif
C++11, the `exception` destructor doesn't ha

Listing 15.15 shows the implementation o
inline in Listing 15.14. Note that nested class
tor more than once. Also note that the `opera`
array index is out of bounds.

```
    for (i = 0; i < lim; ++i)
        gross[i] = gr[i];
    // for i > n and i < MONTHS
    for ( ; i < MONTHS; ++i)
        gross[i] = 0;
}

double Sales::operator[](int i) const
{
    if(i < 0 || i >= MONTHS)
        throw bad_index(i);
    return gross[i];
}

double & Sales::operator[](int i)
{
    if(i < 0 || i >= MONTHS)
        throw bad_index(i);
    return gross[i];
}

LabeledSales::nbad_index::nbad_index(const
            const string & s ) : Sales::bad
{
```

```
{
    if(i < 0 || i >= MONTHS)
        throw nbad_index(Label(), i);
    return Sales::operator[](i);
}
```

Listing 15.16 uses the classes in a program
array in the `LabeledSales` object `sales2` an
`Sales` object `sales1`. These attempts are mad
each kind of exception.

Listing 15.16 **use_sales.cpp**

```
// use_sales.cpp  -- nested exceptions
#include <iostream>
#include "sales.h"

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

```
            cout << sales1[i] << ' ';
            if (i % 6 == 5)
                cout << endl;
        }
        cout << "Year = " << sales2.Year()
        cout << "Label = " << sales2.Label
        for (i = 0; i <= 12; ++i)
        {

            cout << sales2[i] << ' ';
            if (i % 6 == 5)
                cout << endl;
        }
        cout << "End of try block 1.\n";
    }
    catch(LabeledSales::nbad_index & bad)
    {
        cout << bad.what();
        cout << "Company: " << bad.label_v
        cout << "bad index: " << bad.bi_va
    }
    catch(Sales::bad_index & bad)
    {
        cout << bad.what();
```

```
}
```

Here is the output of the program in Listi

```
First try block:
Year = 2011
1220 1100 1122 2212 1232 2334
2884 2393 3302 2922 3002 3544
Year = 2012
Label = Blogstar
12 11 22 21 32 34
28 29 33 29 32 35
Index error in LabeledSales object
Company: Blogstar
bad index: 12

Next try block:
Index error in Sales object
bad index: 20
done
```

```
terminate_handler set_terminate(terminate_
void terminate();              // C++98
void terminate() noexcept;     // C++11
```

Here the `typedef` makes `terminate_hand`
tion that has no arguments and no return valu
its argument, the name of a function (that is, i
`void` return type. It returns the address of the
the `set_terminate()` function more than on
most recent call to `set_terminate()`.

Let's look at an example. Suppose you wan
to print a message to that effect and then call
value of `5`. First, you include the `exception` h
available with a `using` directive or appropriate
`std::` qualifier:

```
#include <exception>
using namespace std;
```

Next, you design a function that does the
prototype:

```
void myQuit()
{
    cout << "Terminating due to uncaught e
```

It's good to know which exceptions to cat
default, aborts the program.

However, there's a bit more to the story. In
include exceptions thrown by functions calle
`Argh()` calls a `Duh()` function that can throw
should appear in the `Argh()` exception speci
specification. Unless you write all the functio
tee that this will get done correctly. You migh
library whose functions don't have exception
look more closely at what happens if a functi
exception specification. (It also suggests that
might be unwieldy, which is part of the reaso

The behavior is much like that for uncaug
exception, the program calls the `unexpected`
`unexpected()` function? No one expects the
turn, calls `terminate()`, which, by default, ca
`set_terminate()` function that modifies the
`set_unexpected()` function that modifies th
functions are also declared in the exception h

```
typedef void (*unexpected_handler)();
unexpected_handler set_unexpected(unexpec
unexpected_handler set_unexpected(unexpec
```

exception type and is declared in the e

- If the newly thrown exception does not and if the original exception specificatio type, the unmatched exception is replac std::bad_exception type.

In short, if you'd like to catch all exception thing like the following. First, you make sure available:

```
#include <exception>
using namespace std;
```

Next, you design a replacement function th bad_exception type and that has the proper

```
void myUnexpected()
{
    throw std::bad_exception();  //or just
}
```

Just using throw without an exception cau However, the exception will be replaced with specification includes that type.

From the preceding discussion of using excep
rectly) that exception handling should be des
Doing this has some disadvantages, though. F
and subtracts from the speed of a program. E
templates because template functions might t
ing on the particular specialization used. Exc
don't always work that well together.

Let's look a little further at dynamic mem
the following function:

```
void test1(int n)
{
    string mesg("I'm trapped in an endles
    ...
    if (oh_no)
        throw exception();
    ...
    return;
}
```

The `string` class uses dynamic memory a
for `mesg` would be called when the function
stack unwinding, the `throw` statement, even t
turely, still allows the destructor to be called.

```
    try {
        if (oh_no)
            throw exception();
    }
    catch(exception & ex)
    {
        delete [] ar;
        throw;
    }
    ...
    delete [] ar;
    return;
}
```

However, this clearly enhances the opportu
Another solution is to use one of the smart po
"The `string` Class and the Standard Template

In short, although exception handling is ex
have costs in terms of programming effort, pro
hand, the cost of no error checking can be mu

already provided ways to do so for their own
support in C++, each vendor's mechanism is
vendors. Creating a language standard for RT
patible with each other.

What Is RTTI For?

Suppose you have a hierarchy of classes desce
a base-class pointer to point to an object of a
call a function that, after processing some info
an object of that type, and returns its address,
How can you tell what kind of object it poin

Before answering this question, you need
the type. Perhaps you want to invoke the cor
case, you don't really need to know the objec
function possessed by all members of the clas
object has an uninherited method. In that cas
Or maybe, for debugging purposes, you woul
objects were generated. For these last two cas

particular type. Let's look at what that means.

```
class Grand { // has virtual methods};
class Superb : public Grand { ... };
class Magnificent : public Superb { ... };
```

Next, suppose you have the following poin

```
Grand * pg = new Grand;
Grand * ps = new Superb;
Grand * pm = new Magnificent;
```

Finally, consider the following type casts:

```
Magnificent * p1 = (Magnificent *) pm;
Magnificent * p2 = (Magnificent *) pg;
Superb * p3 = (Magnificent *) pm;
```

Which of these type casts are safe? Depend could be safe, but the only ones guaranteed to the same type as the object or else a direct or ple, Type Cast #1 is safe because it sets a type `Magnificent` object. Type Cast #2 is not safe object (`Grand`) to a derived-class (`Magnificen` the base-class object to have derived-class pro `Magnificent` object, for example, might have pointed-to object (`*pt`) is of type `Type` or els

```
dynamic_cast<Type *>(pt)
```

Otherwise, the expression evaluates to `0`, th

Listing 15.17 illustrates the process. First it `Grand`, `Superb`, and `Magnificent`. The `Grand` which each of the other classes redefines. The tion, which `Magnificent` redefines (see Figu function that randomly creates and initializes returns the address as a type `Grand *` pointer. active user making decisions.) A loop assigns pg and then uses `pg` to invoke the `Speak()` fu code correctly invokes the `Speak()` version t

```
for (int i = 0; i < 5; i++)
{
    pg = GetOne();
    pg->Speak();
    ...
}
```

You can't use this exact approach (using a p
tion; it's not defined for the `Grand` class. Howe
to see if `pg` can be type cast to a pointer to `Su`
type `Superb` or `Magnificent`. In either case, y

```
if (ps = dynamic_cast<Superb *>(pg))
    ps->Say();
```

Recall that the value of an assignment exp
Thus, the value of the `if` condition is `ps`. If th
If the type cast fails, which it will if `pg` points
15.17 shows the full code. (By the way, some
use the `==` operator in an `if` statement condit
assignment.)

Listing 15.17 **rtti1.cpp**

```
// rtti1.cpp -- using the RTTI dynamic_cas
#include <iostream>
#include <cstdlib>
#include <ctime>

using std::cout;
```

```
public:
    Magnificent(int h = 0, char c = 'A')
    void Speak() const {cout << "I am a m
    void Say() const {cout << "I hold the
               " and the integer "  << Va
};

Grand * GetOne();

int main()
{
    std::srand(std::time(0));
    Grand * pg;
    Superb * ps;
    for (int i = 0; i < 5; i++)
    {
        pg = GetOne();
        pg->Speak();
        if( ps = dynamic_cast<Superb *>(p
            ps->Say();
    }
    return 0;
}
```

functions when possible and RTTI only wher

```
I am a superb class!!
I hold the superb value of 68!
I am a magnificent class!!!
I hold the character R and the integer 68!
I am a magnificent class!!!
I hold the character D and the integer 12!
I am a magnificent class!!!
I hold the character V and the integer 59!
I am a grand class!
```

As you can see, the `Say()` methods were in classes. (The output will vary from run to run the object type.)

You can use `dynamic_cast` with reference no reference value corresponding to the null- value that can be used to indicate failure. Inste `dynamic_cast` throws a type `bad_cast` excep class and defined in the `typeinfo` header file. where `rg` is a reference to a `Grand` object:

```
#include <typeinfo> // for bad_cast
...
try {
```

exception type is derived from the exceptio
header file.

The implementation of the `type_info` cl
`name()` member that returns an implementat
not necessarily) the name of the class. For ex
string defined for the class of the object to w

```
cout << "Now processing type " << typeid(
```

Listing 15.18 modifies Listing 15.17 so th
member function. Note that they are used fo
functions don't handle. The `typeid` test is us
method, so it can't be invoked by a class poin
how the method can be used in debugging. N
header file.

Listing 15.18 **rtti2.cpp**

```
// rtti2.cpp  -- using dynamic_cast, type
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <typeinfo>
using namespace std;
```

```
public:
    Magnificent(int h = 0, char cv = 'A')
    void Speak() const {cout << "I am a ma
    void Say() const {cout << "I hold the
            " and the integer "  << Val
};

Grand * GetOne();
int main()
{
    srand(time(0));
    Grand * pg;
    Superb * ps;
    for (int i = 0; i < 5; i++)
    {
        pg = GetOne();
        cout << "Now processing type " <<
        pg->Speak();
        if( ps = dynamic_cast<Superb *>(pg
            ps->Say();
        if (typeid(Magnificent) == typeid(
            cout << "Yes, you're really ma
    }
    return 0;
```

```
Yes, you're really magnificent.
Now processing type Superb.
I am a superb class!!
I hold the superb value of 37!
Now processing type Grand.
I am a grand class!
Now processing type Superb.
I am a superb class!!
I hold the superb value of 18!
Now processing type Grand.
I am a grand class!
```

As with the preceding example, the exact
program uses `rand()` to select types. Also son
when `name()` is called, for example, `5Grand` i

Misusing RTTI

RTTI has many vocal critics within the C++
sary, a potential source of program inefficienc
ming practices. Without delving into the deb
programming that you should avoid.

```
        pm->Speak();
        pm->Say();
    }
    else if (typeid(Superb) == typeid(*pg)
    {
        ps = (Superb *) pg;
        ps->Speak();
        ps->Say();
    }
    else
        pg->Speak();
}
```

Not only is this uglier and longer than the
each class explicitly. Suppose, for example, tha
`Insufferable` class from the `Magnificent` cl
`Speak()` and `Say()`. With the version that use
would have to modify the `for` loop code, add
version, however, requires no changes at all. T
derived from `Grand`:

```
pg->Speak();
```

And this statement works for all classes der

```
if( ps = dynamic_cast<Superb *>(pg))
      ps->Say();
```

implausible, none of them make much sense.
allowed? In C, all of them are. Stroustrup's re
is allowable for a general type cast and to add
discipline for the casting process:

```
dynamic_cast
const_cast
static_cast
reinterpret_cast
```

Instead of using a general type cast, you ca
ular purpose. This documents the intended re
a chance to check that you did what you tho

You've already seen the `dynamic_cast` op
are two classes, that `ph` is type `High *`, and th
statement assigns a `Low *` pointer to `pl` only
indirect) to `High`:

```
pl = dynamic_cast<Low *> ph;
```

Otherwise, the statement assigns the null p
this syntax:

```
dynamic_cast < type-name > (expression)
```

declare the value as const and use const_cas
could be done using the general type cast, but
ously change the type:

```
High bar;
const High * pbar = &bar;
...
High * pb = (High *) (pbar);       // vali
Low * pl = (Low *) (pbar);         // also
```

Because the simultaneous change of type a
programming slip, using the const_cast oper

The const_cast is not all powerful. It can
the effect of attempting to change a quantity
clarify this statement with the short example

Listing 15.19 **constcast.cpp**

```
// constcast.cpp -- using const_cast<>
#include <iostream>
using std::cout;
using std::endl;
```

The const_cast operator can remove the
the compiler to accept the following stateme

```
*pc += n;
```

However, because pop2 is declared as cons
change, as is shown by the following sample

```
pop1, pop2: 38383, 2000
pop1, pop2: 38280, 2000
```

As you can see, the calls to change() alter
change() is declared as const int *, so it c
pointed-to int. The pointer pc has the cons
pointed-to value, but only if that value wasn'
alter pop1 but not pop2.

The static_cast operator has the same

```
static_cast < type-name > (expression)
```

It's valid only if *type_name* can be conver
expression has, or vice versa. Otherwise, th
base class to Low and that Pond is an unrelate
and Low to High are valid, but a conversion f

```
High bar;
Low blow;
...
```

```
dat * pd = reinterpret_cast< dat *> (&valu
cout << hex << pd->a;   // display first 2
```

Typically, such type casts would be used fo
gramming and would not be portable. For ex
multibyte value in a different order than does

The reinterpret_cast operator doesn't
you can cast a pointer type to an integer type
representation, but you can't cast a pointer to
point type. Another restriction is that you can
or vice versa.

The plain type cast in C++ is also restricte
type casts can do, plus some combinations, suc
followed by a const_cast, but it can't do any
allowed in C but, typically, not in C++ becau
type is too small to hold a pointer implement

```
char ch = char (&d);          // type cast
```

These restrictions make sense, but if you fi
sive, you still have C available.

The RTTI features allow a program to de
dynamic_cast operator is used to cast a deri
main purpose is to ensure that it's okay to inv
operator returns a type_info object. Two ty
determine whether an object is of a specific
be used to obtain information about an obje

The dynamic_cast, static_cast, const_
provide safer, better-documented type casts t

Chapter Review

1. What's wrong with the following atten

a.
```
class snap {
    friend clasp;
     ...
};
class clasp { ... };
```

b.
```
class cuff {
public:
      void snip(muff &) { ...
      ...
```

```
{
private:
    class Sauce
    {
          int soy;
          int sugar;
   public:
          Sauce(int s1, int s2) : so
   };
   ...
};
```

4. How does `throw` differ from `return`?

5. Suppose you have a hierarchy of except
exception class. In what order should yo

6. Consider the `Grand`, `Superb`, and `Magni`
pose `pg` is a type `Grand *` pointer that i
these three classes and that `ps` is a type
how the following two code samples be

```
if (ps = dynamic_cast<Superb *>(pg))
    ps->say();  // sample #1
```

objects need not hold the bad values; t

3. This exercise is the same as Programmi should be derived from a base class (its two argument values, the exceptions sh as well as the function name, and a sing exemption should be used for both exc loop to terminate.

4. Listing 15.16 uses two `catch` blocks af exception leads to the `label_val()` m that it uses a single `catch` block after e invoking `label_val()` only when app

- Container classes
- Iterators
- Function objects (functors)
- STL algorithms
- The `initializer_list` template

By now you are familiar with the C++ g
when you can reuse code written by others.
are many commercially available C++ class l
as part of the C++ package. For example, yo
ported by the `ostream` header file. This chap
your programming pleasure.

You've already encountered the `string` c
sively. Then the chapter looks at "smart point
dynamic memory a bit easier. Next, the chap
(STL), a collection of useful templates for ha
The STL exemplifies the programming parad
chapter looks at the `initializer_list` tem
using initializer-list syntax with STL objects.

constructors used in Listing 16.1, in that orde
The constructor representations are simplified
really is a `typedef` for a template specializatio
an optional argument relating to memory ma
chapter and in Appendix F, "The `string` Tem
implementation-dependent integral type defi
defines `string::npos` as the maximum possi
equal the maximum value of an `unsigned in`
tion NBTS for null-byte-terminated string—
minated with a null character.

Table 16.1 **`string` Class Constructors**

Constructor	D
`string(const char * s)`	I t
`string(size_type n, char c)`	C i
`string(const string & str)`	I s

(C++11)

Listing 16.1 **str1.cpp**

```
// str1.cpp -- introducing the string cla
#include <iostream>
#include <string>
// using string constructors

int main()
{
    using namespace std;
    string one("Lottery Winner!");      //
    cout << one << endl;                //
    string two(20, '$');                //
    cout << two << endl;
    string three(one);                  //
    cout << three << endl;
    one += " Oops!";                    //
    cout << one << endl;
    two = "Sorry! That was ";
    three[0] = 'P';
    string four;                        //
```

```
Lottery Winner!
Lottery Winner! Oops!
Sorry! That was Pottery Winner!
All's well that ends!
well, well...
That was Pottery in motion!
```

Program Notes

The start of the program in Listing 16.1 illust
to a regular C-style string and display it by us

```
string one("Lottery Winner!");    // ctor
cout << one << endl;              // overl
```

The next constructor initializes the `strin`
characters:

```
string two(20, '$');              // ctor
```

The copy constructor initializes the `strin`

```
string three(one);                // ctor
```

The overloaded `+=` operator appends the s

```
one += " Oops!";                  // overl
```

might expect, the + operator concatenates its
The operator is multiply overloaded, so the s
C-style string or a char value.

The fifth constructor takes a C-style strin
ger indicating how many characters to copy:

```
char alls[] = "All's well that ends well"
string five(alls,20);             // ctor
```

Here, as the output shows, just the first 20
used to initialize the five object. As Table 16
length of the C-style string, the requested nu
ing 20 with 40 in the preceding example wo
at the end of five. (That is, the constructor
lowing the string "All's well that ends

The sixth constructor has a template argu

```
template<class Iter> string(Iter begin, I
```

The intent is that *begin* and *end* act like
ory. (In general, *begin* and *end* can be iterato
used in the STL.) The constructor then uses
by *begin* and *end* to initialize the string ob
borrowed from mathematics, means the rang

Now suppose you want to use this constru
string object—say, the object five. The follo

```
string seven(five + 6, five + 10);
```

The reason is that the name of an object, u
the address of an object, hence five is not a p
ever, five[6] is a char value, so &five[6] is
to the constructor:

```
string seven(&five[6], &five[10]);// ctor
```

The seventh constructor copies a portion o
object:

```
string eight(four, 7, 16);          // ctor
```

This statement copies 16 characters from f
eighth character) in four.

C++11 Constructors

The string(string && str) noexcept con
that the new string is a copy of str. Howev
guarantee that str will be treated as const. T
constructor. The compiler can use it in some sit

```
string stuff;
cin >> stuff;             // read a word
getline(cin, stuff);      // read a line,
```

Both versions of `getline()` allow for an
acter to use to delimit input:

```
cin.getline(info,100,':');    // read up
getline(stuff, ':');          // read up
```

The main operational difference is that th
get `string` object to hold the input characte

```
char fname[10];
string lname;
cin >> fname;       // could be a problem
cin >> lname;       // can read a very, ve
cin.getline(fname, 10);  // may truncate
getline(cin, fname);     // no truncation
```

The automatic sizing feature allows the st
the numeric parameter that limits the numbe

A design difference is that the C-style stri
`istream` class, whereas the `string` versions a
invoking object for C-style string input and
This applies to the `>>` form, too, which is evi

the stream. In this system, setting eofbit regi
failbit registers detecting an input error; set
failure, such as a hardware failure; and setting
17, "Input, Output, and Files," discusses this fu

The operator>>() function for the strin
of reading to and discarding a delimiting char
and leaves that character in the input queue. A
or tab character or more generally, any charac

So far in this book, you've seen several exa
input functions for string objects work with
can also use them for file input. Listing 16.2 s
the file. It assumes that the file contains string
the getline() method of specifying a delimi
one string to an output line.

Listing 16.2 **strfile.cpp**

```
// strfile.cpp -- read strings from a file
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
int main()
```

Here is a sample `tobuy.txt` file:

```
sardines:chocolate ice cream:pop corn:lee
cottage cheese:olive oil:butter:tofu:
```

Typically, for the program to find the text
tory as the executable program or sometimes
you can provide the full path name. On a Wi
string the escape sequence \\ represents a sin

```
fin.open("C:\\CPP\\Progs\\tobuy.txt"); //
```

Here is the output of the program in Listi

```
1: sardines
2: chocolate ice cream
3: pop corn
4: leeks
5:
cottage cheese
6: olive oil
7: butter
8: tofu
9:

Done
```

```
if (snake1 < snake 2)          // operator
    ...
if (snake1 == snake3)          // operator
    ...
if (snake3 != snake2)          // operator
    ...
```

You can determine the size of a string. Bot
tions return the number of characters in a stri

```
if (snake1.length() == snake2.size())
    cout << "Both strings have the same le
```

Why two functions that do the same thing
versions of the `string` class, and `size()` was
You can search a string for a given substrin
16.2 provides a short description of four varia
`string::npos` is the maximum possible numb
largest `unsigned int` or `unsigned long` valu

The `string` library also provides the relat
`find_last_of()`, `find_first_not_of()`, an
set of overloaded function signatures as the f
the last occurrence of a substring or characte
first occurrence in the invoking string of any
example, the following statement would retu
the index 3) because that's the first occurren

```
int where = snake1.find_first_of("hark");
```

The `find_last_of()` method works the
Thus, the following statement would return

```
int where = snake1.last_first_of("hark");
```

The `find_first_not_of()` method finds
is not a character in the argument. So the fol
`cobra` because `c` is not found in `hark`:

```
int where = snake1.find_first_not_of("har
```

(You'll learn about `find_last_not_of()`

```
    "plaid", "quilt", "remote", "stolid",
    "valid", "whence", "xenon", "yearn", "

int main()
{
    using std::cout;
    using std::cin;
    using std::tolower;
    using std::endl;
    std::srand(std::time(0));
    char play;
    cout << "Will you play a word game? <y
    cin >> play;
    play = tolower(play);
    while (play == 'y')
    {
        string target = wordlist[std::ranc
        int length = target.length();
        string attempt(length, '-');
        string badchars;
        int guesses = 6;
        cout << "Guess my secret word. It
            << " letters, and you guess\n'
            << "one letter at a time. You
```

```
            attempt[loc]=letter;
            // check if letter appear
            loc = target.find(letter,
            while (loc != string::npo
            {
                attempt[loc]=letter;
                loc = target.find(let
            }
        }
        cout << "Your word: " << atte
        if (attempt != target)
        {
            if (badchars.length() > 0
                cout << "Bad choices:
            cout << guesses << " bad
        }
    }
    if (guesses > 0)
        cout << "That's right!\n";
    else
        cout << "Sorry, the word is "
```

```
Your word: a--a--
Bad choices: e
5 bad guesses left
Guess a letter: t
Oh, bad guess!
Your word: a--a--
Bad choices: et
4 bad guesses left
Guess a letter: r
Good guess!
Your word: a--ar-
Bad choices: et
4 bad guesses left
Guess a letter: y
Good guess!
Your word: a--ary
Bad choices: et
4 bad guesses left
Guess a letter: i
Good guess!
Your word: a-iary
Bad choices: et
4 bad guesses left
Guess a letter: p
```

failure to find a character or a string.

The program makes use of the fact that o
ator lets you append individual characters to

```
badchars += letter;  // append a char to
```

The heart of the program begins by chec
tery word:

```
int loc = target.find(letter);
```

If `loc` is a valid value, the letter can be pla
answer string:

```
attempt[loc]=letter;
```

However, a given letter might occur more
gram has to keep checking. The program use
which specifies a starting place in the string f
letter was found at location `loc`, the next sea
keeps the search going until no more occurr
`find()` indicates failure if `loc` is after the en

```
// check if letter appears again
loc = target.find(letter, loc + 1);
while (loc != string::npos)
{
```

exceeds that size, the program allocates a new
grow without continuous resizing. The capac
rent block, and the `reserve()` method allows
block. Listing 16.4 shows an example that use

Listing 16.4 **str2.cpp**

```
// str2.cpp -- capacity() and reserve()
#include <iostream>
#include <string>
int main()
{
    using namespace std;
    string empty;
    string small = "bit";
    string larger = "Elephants are a girl
    cout << "Sizes:\n";
    cout << "\tempty: " << empty.size() <<
    cout << "\tsmall: " << small.size() <<
    cout << "\tlarger: " << larger.size()
    cout << "Capacities:\n";
    cout << "\tempty: " << empty.capacity
    cout << "\tsmall: " << small.capacity
    cout << "\tlarger: " << larger.capacit
    empty.reserve(50);
```

```
string filename;
cout << "Enter file name: ";
cin >> filename;
ofstream fout;
```

The bad news is that the open() method
news is that the c_str() method returns a p
contents as the invoking string object. So y

```
fout.open(filename.c_str());
```

String Varieties

This section treats the string class as if it we
tioned earlier, the string library really is base

```
template<class charT, class traits = char
       class Allocator = allocator<charT
basic_string {...};
```

The basic_string template comes with
typedef name:

```
typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
typedef basic_string<char16_t> u16string;
typedef basic_string<char32_t> u32string
```

```
    str = ps;
    return;
}
```

You probably see its flaw. Each time the fu
the heap but never frees the memory, thus cre
solution—just remember to free the allocated
just before the `return` statement:

```
delete ps;
```

However, a solution involving the phrase "j
tion. Sometimes you won't remember. Or you
comment out the code. And even if you do re
sider the following variation:

```
void remodel(std::string & str)
{
    std::string * ps = new std::string(st
    ...
    if (weird_thing())
        throw exception();
    str = *ps;
    delete ps;
    return;
}
```

rectly) by new. When the smart pointer expir
memory. Thus, if you assign an address return
have to remember to free the memory later;
pointer object expires. Figure 16.2 illustrates
and a regular pointer. The shared_ptr and u
situation.

To create one of these smart pointer objec
which includes the template definitions. Then
instantiate the kind of pointer you require. T
the following constructor:

```
template<class X> class auto_ptr {
public:
    explicit auto_ptr(X* p =0) throw();
...};
```

(The throw() notation, recall, means this
auto_ptr, it is deprecated.) Thus, asking for a
auto_ptr object that points to type X:

```
auto_ptr<double> pd(new double);  // pd a
                                  // (use
auto_ptr<string> ps(new string);  // ps a
                                  // (use
```

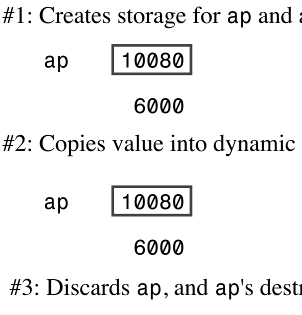


Figure 16.2 A regular p

Here new double is a pointer returned by
It is the argument to the auto_ptr<double>
corresponding to the formal parameter p in th
an actual argument for a constructor. The oth

```
unique_ptr<double> pdu(new double);  // pd
shared_ptr<string> pss(new string);  // ps
```

C++11 shared_ptr and unique_ptr classes
the pointer expires when execution leaves th
ods to report when an object is created or de

Listing 16.5 **smrtptrs.cpp**

```
// smrtptrs.cpp -- using three kinds of s
// requires support of C++11 shared_ptr a
#include <iostream>
#include <string>
#include <memory>

class Report
{
private:
    std::string str;
public:
    Report(const std::string s) : str(s)
           { std::cout << "Object created
    ~Report() { std::cout << "Object dele
    void comment() const { std::cout << s
};

int main()
{
```

```
Object created!
using unique_ptr
Object deleted!
```

Each of these classes has an `explicit` cons
Thus, there is no automatic type cast from a p

```
shared_ptr<double> pd;
double *p_reg = new double;
pd = p_reg;                          // not
pd = shared_ptr<double>(p_reg);      // all
shared_ptr<double> pshared = p_reg; // not
shared_ptr<double> pshared(p_reg);  // all
```

The smart pointer template classes are defi
object acts like a regular pointer. For example
can dereference it (`*ps`), use it to access struct
to a regular pointer that points to the same ty
object to another of the same type, but that ra

But first, here's something you should avoi

```
string vacation("I wandered lonely as a cl
shared_ptr<string> pvac(&vacation);  // NO
```

When `pvac` expires, the program would ap
memory, which is wrong.

the object. Then have assignment trans auto_ptr and for unique_ptr, althoug

- Create an even smarter pointer that ke to a particular object. This is called *refe* increase the count by one, and the exp by one. Only when the final pointer e shared_ptr strategy.

The same strategies we've discussed for as copy constructors.

Each approach has its uses. Listing 16.6 sh poorly suited.

Listing 16.6 **fowl.cpp**

```
// fowl.cpp  -- auto_ptr a poor choice
#include <iostream>
#include <string>
#include <memory>

int main()
{
    using namespace std;
    auto_ptr<string> films[5] =
```

```
Segmentation fault (core dumped)
```

The "core dumped" message should help f
can be a problem. (The behavior for this sort
encounter different behavior, depending on y
following statement transfers ownership from

```
pwin = films[2];   // films[2] loses owne
```

That causes `films[2]` to no longer refer t
ownership of an object, it no longer provides
goes to print the string pointed to by `films[`
ently is an unpleasant surprise.

Suppose you go back to Listing 16.6 but u
need a compiler that supports the C++11 `sh`
and gives this output:

```
The nominees for best avian baseball film
Fowl Balls
Duck Walks
Chicken Runs
Turkey Errors
Goose Eggs
The winner is Chicken Runs!
```

```
auto_ptr<string> p2;
p2 = p1;
```

When, in statement #3, p2 takes over own
of ownership. This, recall, is good because it p
from trying to delete the same object. But it
to use p1 because p1 no longer points to vali

Now consider the unique_ptr equivalent

```
unique_ptr<string> p3(new string("auto");
unique_ptr<string> p4;
p4 = p3;
```

In this case, the compiler does not allow s
not pointing to valid data. Hence, unique_pt
tial program crash) than auto_ptr.

But there are times when assigning one sm
gerous orphan behind. Suppose we have this

```
unique_ptr<string> demo(const char * s)
{
    unique_ptr<string> temp(new string(s)
    return temp;
}
```

source of mischief. Assignment #2 leaves no u
unique_ptr constructor, which constructs a t
ownership is transferred to pu3. This selective
is superior to auto_ptr, which would allow b
that auto_ptrs are banned (by recommendati
container objects, whereas unique_ptrs are al
something along the lines of assignment #1 to
ptrs, you get a compile-time error. If the algo
#2, that's okay, and the program proceeds. Wi
could lead to undefined behavior and mysteri

Of course, it could be possible that you rea
The assignment is unsafe only if you use the a
manner, such as dereferencing it. But you cou
new value to it. C++ has a standard library fu
assign one unique_ptr to another. Here is an
demo() function, which returns a unique_ptr

```
using namespace std;
unique_ptr<string> ps1, ps2;
ps1 = demo("Uniquely special");
ps2 = move(ps1);                 // enable
ps1 = demo(" and more");
cout << *ps2 << *ps1 << endl;
```

you might have an STL container of pointers
or assignment operations that will work with
(you'll get a compiler warning) or `auto_ptr`
piler doesn't offer `shared_ptr`, you can get a

If the program doesn't need multiple poin
It's a good choice for the return type for a fu
cated by `new`. That way, ownership is transferr
value, and that smart pointer takes on the res
`unique_ptr` objects in an STL container prov
rithms, such as `sort()`, that copy or assign or
assuming the proper `includes` and `using` stat
could be used in a program:

```
unique_ptr<int> make_int(int n)
{
    return unique_ptr<int>(new int(n));
}
void show(unique_ptr<int> & pi)
{
    cout << *a << ' ';
}
int main()
{
...
```

ously owned by the unique_ptr.

You would use auto_ptr in the same situa
ferred. If your compiler doesn't provide uniqu
library scoped_ptr, which is similar.

The Standard Template Li

The STL provides a collection of templates re
objects, and algorithms. A container is a unit, l
STL containers are homogeneous; that is, they
are recipes for accomplishing particular tasks, s
lar value in a list. Iterators are objects that let y
pointers let you move through an array; they a
objects are objects that act like functions; they
(including function names because a function
construct a variety of containers, including arr
a variety of operations, including searching, so

Alex Stepanov and Meng Lee developed S
ing the implementation in 1994. The ISO/AN
as a part of the C++ Standard. The STL is no
ming. Instead, it represents a different program
This makes STL interesting both in terms of v

vector object, assign one vector object to a
vector elements. To make the class generic, y
STL does, defining a vector template in the

To create a vector template object, you u
type to be used. Also the vector template use
use an initialization argument to indicate hov

```
#include vector
using namespace std;
vector<int> ratings(5);       // a vector
int n;
cin >> n;
vector<double> scores(n);     // a vector
```

After you create a vector object, operator
the usual array notation for accessing individu

```
ratings[0] = 9;
for (int i = 0; i < n; i++)
    cout << scores[i] << endl;
```

Allocators Again

Like the string class, the various STL contain
ment that specifies what allocator object to u
template begins like this:

```
    using std::cout;
    using std::endl;

    vector<int> ratings(NUM);
    vector<string> titles(NUM);
    cout << "You will do exactly as told.
         << NUM << " book titles and your
    int i;
    for (i = 0; i < NUM; i++)
    {
        cout << "Enter title #" << i + 1 <
        getline(cin,titles[i]);
        cout << "Enter your rating (0-10):
        cin >> ratings[i];
        cin.get();
    }
    cout << "Thank you. You entered the fo
          << "Rating\tBook\n";
    for (i = 0; i < NUM; i++)
    {
        cout << ratings[i] << "\t" << titl
    }

    return 0;
}
```

dynamically allocated array. The next section
methods.

Things to Do to Vectors

Besides allocating storage, what else can the v
containers provide certain basic methods, inc
elements in a container, `swap()`, which excha
`begin()`, which returns an iterator that refers
`end()`, which returns an iterator that represer

What's an iterator? It's a generalization of
can be an object for which pointer-like opera
`operator*()`) and incrementing (for exampl
you'll see later, generalizing pointers to iterat
interface for a variety of container classes, inc
wouldn't work. Each container class defines a
ator is a class scope `typedef` called `iterator`
type `double` specialization of `vector`, you wo

```
vector<double>::iterator pd;  // pd an it
```

Suppose `scores` is a `vector<double>` obj

```
vector<double> scores;
```

```
for (pd = scores.begin(); pd != scores.end
    cout << *pd << endl;;
```

All containers have the methods just discus
methods that only some STL containers have.
adds an element to the end of a `vector`. Whil
ment so that the `vector` size increases to acco
can write code like the following:

```
vector<double> scores;  // create an empty
double temp;
while (cin >> temp && temp >= 0)
    scores.push_back(temp);
cout << "You entered " << scores.size() <<
```

Each loop cycle adds one more element to
the number of element when you write the p
long as the program has access to sufficient m
necessary.

The `erase()` method removes a given ran
ments that define the range to be removed. It'
STL defines ranges using two iterators. The fi
range, and the second iterator is one beyond t

things.begin()

Figure 16.3 The S

Note

A range [it1, it2) is specified by two iterato
not including, it2.

An insert() method complements eras
first gives the position ahead of which new e
third iterator parameters define the range to
another container object. For example, the fo
of the new_v vector ahead of the first elemen

```
vector<int> old_v;
vector<int> new_v;
...
old_v.insert(old_v.begin(), new_v.begin()
```

Incidentally, this is a case where having a p
makes it simple to append something to the
is inserted ahead of old.end(), meaning it's

```
old_v.insert(old_v.end(), new_v.begin() +
```

```
vector<Review> books;
Review temp;
while (FillReview(temp))
    books.push_back(temp);
int num = books.size();
if (num > 0)
{
    cout << "Thank you. You entered th
        << "Rating\tBook\n";
    for (int i = 0; i < num; i++)
        ShowReview(books[i]);
    cout << "Reprising:\n"
        << "Rating\tBook\n";
    vector<Review>::iterator pr;
    for (pr = books.begin(); pr != boo
        ShowReview(*pr);
    vector <Review> oldlist(books);
    if (num > 3)
    {
        // remove 2 items
        books.erase(books.begin() + 1,
        cout << "After erasure:\n";
        for (pr = books.begin(); pr !=
            ShowReview(*pr);
```

```
    std::cout << "Enter book rating: ";
    std::cin >> rr.rating;
    if (!std::cin)
        return false;
    // get rid of rest of input line
    while (std::cin.get() != '\n')
        continue;
    return true;
}

void ShowReview(const Review & rr)
{
    std::cout << rr.rating << "\t" << rr.
}
```

Here is a sample run of the program in Li

```
Enter book title (quit to quit): The Cat
Enter book rating: 5
Enter book title (quit to quit): Candid C
Enter book rating: 7
Enter book title (quit to quit): Warriors
Enter book rating: 4
Enter book title (quit to quit): Quantum
```

```
7       Candid Canines
4       Warriors of Wonk
8       Quantum Manners
```

More Things to Do to Vectors

There are many things programmers commor
them, randomize the order, and so on. Does tl
these common operations? No! The STL take
tions for these operations. Thus, instead of def
for each container class, it defines a single `fin`
for all container classes. This design philosoph
suppose you had 8 container classes and 10 op
member function, you'd need 8 10, or 80, sep
the STL approach, you'd need just 10 separate
you defined a new container class, provided th
could use the existing 10 nonmember functio

On the other hand, the STL sometimes de
defines a nonmember function for the same ta
is a class-specific algorithm that is more efficie
fore, the vector `swap()` will be more efficient

domly rearranges the order of all the elemen

```
random_shuffle(books.begin(), books.end()
```

Unlike `for_each`, which works with any c
container class allow random access, which th

The `sort()` function, too, requires that th
in two versions. The first version takes two it
range by using the `<` operator defined for the
example, the following sorts the contents of c
built-in `<` operator to compare values:

```
vector<int> coolstuff;
...
sort(coolstuff.begin(), coolstuff.end());
```

If the container elements are user-defined
`operator<()` function defined that works wi
For example, you could sort a vector contain
`Review` member function or a nonmember fu
a structure, its members are public, and a non

```
bool operator<(const Review & r1, const R
{
    if (r1.title < r2.title)
        return true;
```

```
    else
        return false;
}
```

With this function in place, you can use th
tor of `Review` objects in order of increasing ra

```
sort(books.begin(), books.end(), WorseThan
```

Note that the `WorseThan()` function does
ordering `Review` objects. If two objects have t
function sorts by using the rating member. Bu
ber, `WorseThan()` treats them as equivalent. Th
ordering, and the second kind is called *strict wea*
and $b < a$ are false, then a and b must be ident
They might be identical, or they might just ha
`rating` member in the `WorseThan()` example
identical, the best you can say for strict weak o

Listing 16.9 illustrates the use of these STL

Listing 16.9 **vect3.cpp**

```
// vect3.cpp -- using STL functions
#include <iostream>
#include <string>
```

```
            << books.size() << " ratings
             << "Rating\tBook\n";
        for_each(books.begin(), books.end

        sort(books.begin(), books.end());
        cout << "Sorted by title:\nRating
        for_each(books.begin(), books.end

        sort(books.begin(), books.end(),
        cout << "Sorted by rating:\nRatin
        for_each(books.begin(), books.end

        random_shuffle(books.begin(), boo
        cout << "After shuffling:\nRating
        for_each(books.begin(), books.end
    }
    else
        cout << "No entries. ";
    cout << "Bye.\n";
    return 0;
}

bool operator<(const Review & r1, const R
{
```

```
    std::cin >> rr.rating;
    if (!std::cin)
        return false;
    // get rid of rest of input line
    while (std::cin.get() != '\n')
        continue;
    return true;
}

void ShowReview(const Review & rr)
{
    std::cout << rr.rating << "\t" << rr.t
}
```

Here's a sample run of the program in Listi

```
Enter book title (quit to quit): The Cat W
Enter book rating: 8
Enter book title (quit to quit): The Dogs
Enter book rating: 6
Enter book title (quit to quit): The Wimps
Enter book rating: 3
Enter book title (quit to quit): Farewell
Enter book rating: 7
```

```
6       The Dogs of Dharma
8       The Cat Who Can Teach You Weight
Bye.
```

The Range-Based for Loop (C++

The range-based for loop, mentioned in Ch
is designed to work with the STL. To review,

```
double prices[5] = {4.99, 10.99, 6.87, 7.
for (double x : prices)
    cout << x << std::endl;
```

The contents of the parentheses for the fo
in a container and then the name of the cont
named variable to access each container elem
ment from Listing 16.9:

```
for_each(books.begin(), books.end(), Show
```

It can be replaced with the following rang

```
for (auto x : books) ShowReview(x);
```

The compiler will use the type of books, w
type Review, and the loop will pass each Revi

Why Iterators?

Understanding iterators is perhaps the key to make algorithms independent of the type of d independent of the type of container used. Th STL's generic approach.

To see why iterators are needed, let's look a tion for two different data representations and approach. First, let's consider a function that se particular value. You could write the function

```
double * find_ar(double * ar, int n, const
{
    for (int i = 0; i < n; i++)
        if (ar[i] == val)
            return &ar[i];
    return 0;  // or, in C++11, return nul
}
```

If the function finds the value in the array, value is found; otherwise, it returns the null po through the array. You could use a template to operator. Nonetheless, this algorithm is still tie the linked list.

If you consider details of implementation,
rithms: One uses array indexing to move thro
`start` to `start->p_next`. But broadly, the tw
value with each value in the container in seq

The goal of generic programming in this
that would work with arrays or linked lists or
should the function be independent of the da
independent of the data structure of the cont
resentation for the data type stored in a conta
tion of the process of moving through the va
generalized representation.

What properties should an iterator have in
a short list:

- You should be able to dereference an it
  it refers. That is, if `p` is an iterator, `*p` sh
- You should be able to assign one iterat
  the expression `p = q` should be defined
- You should be able to compare one ite
  `q` are iterators, the expressions `p == q`

```
typedef double * iterator;
iterator find_ar(iterator begin, iterator
{
    iterator ar;
    for (ar = begin; ar != end;  ar++)
        if (*ar == val)
            return ar;
    return end;   // indicates val not fou
}
```

For the `find_ll()` function, you can defin operators:

```
struct Node
{
    double item;
    Node * p_next;
};

class iterator
{
    Node * pt;
public:
    iterator() : pt(0) {}
    iterator (Node * pn) : pt(pn) {}
```

```
{
    iterator start;
    for (start = head; start!= 0; ++start
        if (*start == val)
            return start;
    return 0;
}
```

This is very nearly the same as find_ar() functions determine whether they've reached find_ar() function uses an iterator to one-p value stored in the final node. Remove that d tions identical. For example, you could requi ment after the last official element. That is, yo list have a past-the-end element, and you cou the past-the-end position. Then find_ar() a detecting the end of data and become identic end element moves from making requiremen the container class.

The STL follows the approach just outline deque, and so on) defines an iterator type app tor might be a pointer; for another, it might b the iterator will provide the needed operation

```
list<double>::iterator pr;
for (pr = scores.begin(); pr != scores.end
    cout << *pr << endl;
```

The only change is in the type declared fo
priate iterators and designing the classes in a u
same code for containers that have quite dissi

With C++ automatic type deduction, you
code with either the vector or the list:

```
for (auto pr = scores.begin(); pr != score
    cout << *pr << endl;
```

Actually, as a matter of style, it's better to av
possible, you should use an STL function, such
for you. Alternatively, use the C++11 range-b

```
for (auto x : scores) cout << x << endl;
```

So to summarize the STL approach, you st
tainer. You express it in as general terms as pos
and container type. To make the general algor
iterators that meet the needs of the algorithm
design. That is, basic iterator properties and co
placed on the algorithm.

them) and can be compared for equality (usin
inequality (using the `!=` operator, possibly ove
dereferencing one should produce the same v

```
iter1 == iter2
```

is true, then the following is also true:

```
*iter1 == *iter2
```

Of course, these properties hold true for b
requirements are guides for what you must d
iterator class. Now let's look at other iterator

Input Iterators

The term *input* is used from the viewpoint o
the container to the program is considered in
the program is considered input. So an *input*
values from a container. In particular, derefere
to read a value from a container, but it needn
algorithms that require an input iterator are a
container.

An input iterator has to allow you to acce
supporting the `++` operator, both in prefix an
the first element in a container and incremen

In short, you can use an input iterator for s
put operator for single-pass, write-only algori

Forward Iterators

Like input and output iterators, forward iterat
through a container. So a forward iterator can
element at a time. However, unlike input and
a sequence of values in the same order each ti
forward iterator, you can still dereference the
the same value. These properties make multipl

A forward iterator can allow you to both r
to read it:

```
int * pirw;       // read-write iterator
const int * pir;  // read-only iterator
```

Bidirectional Iterators

Suppose you have an algorithm that needs to
tions. For example, a reverse function could sv
pointer to the first element, decrement the po
process. A bidirectional iterator has all the feat
for the two decrement operators (prefix and p

`a < b`	True if `b - a > 0`
`a > b`	True if `b < a`
`a >= b`	True if `!(a < b)`
`a <= b`	True if `!(a > b)`

Expressions such as `a + n` are valid only i
the container (including past-the-end).

Iterator Hierarchy

You have probably noticed that the iterator k
all the capabilities of an input iterator and of
bidirectional iterator has all the capabilities o
And a random access iterator has all the capa
capabilities. Table 16.4 summarizes the main
`n` is an integer.

Table 16.4 **Iterator Capabilities**

Iterator Capability	Input	Output
Dereferencing read	Yes	No
Dereferencing write	No	Yes

random access iterator, can be used just with c

Note that the various iterator kinds are no characterizations. As mentioned earlier, each c name called `iterator`. So the `vector<int>` c `vector<int>::iterator`. But the documenta iterators are random access iterators. That, in t any iterator type because a random access iter a `list<int>` class has iterators of type `list<i` doubly linked list, so it uses a bidirectional iter random access iterators, but it can use algorith

Concepts, Refinements, and Mode

The STL has several features, such as kinds of language. That is, although you can design, say iterator, you can't have the compiler restrict an son is that the forward iterator is a set of requi be satisfied by an iterator class you've designed nary pointer. An STL algorithm works with a requirements. STL literature uses the word *co* there is an input iterator concept, a forward it do need iterators for, say, a container class you which include iterator templates for the stand

The STL `sort()` function, recall, takes as
element in a container and an iterator pointin
just `Receipts`) is the address of the first elem
`+ SIZE`) is the address of the element followi
lowing function call sorts the array:

```
sort(Receipts, Receipts + SIZE);
```

C++ guarantees that the expression `Recei`
in the array or one past-the-end. Thus, C++
pointers into an array, and this makes it possib
arrays. Thus, the fact that pointers are iterator
it possible to apply STL algorithms to ordina
rithms to data forms of your own design, pro
(which may be pointers or objects) and past-

`copy()`, `ostream_iterator`, and `ist`

The STL provides some predefined iterators.
There is an algorithm called `copy()` for copy
algorithm is expressed in terms of iterators, so
another or even from or to an array, because
tors. For example, the following copies an arr

The out_iter iterator now becomes an in
play information. The first template argument
being sent to the output stream. The second t
cates the character type used by the output st
wchar_t.) The first constructor argument (cou
being used. It could also be a stream used for
ment is a separator to be displayed after each i

You could use the iterator like this:

```
*out_iter++ = 15;   // works like cout <<
```

For a regular pointer, this would mean assi
tion and then incrementing the pointer. For t
ment means send 15 and then a string consist
by cout. Then it should get ready for the next
with copy() as follows:

```
copy(dice.begin(), dice.end(), out_iter);
```

This would mean to copy the entire range
stream—that is, to display the contents of the

Or you could skip creating a named iterato
instead. That is, you could use the adapter like

```
copy(dice.begin(), dice.end(), ostream_ite
```

contents to the output stream:

```
ostream_iterator<int, char> out_iter(cout
copy(dice.begin(), dice.end(), out_iter);
```

Now suppose you want to print the conte forming time-reversal studies.) There are seve than wallow in them, let's go to one that doe called `rbegin()` that returns a reverse iterato `rend()` that returns a reverse iterator pointin a reverse iterator makes it decrement, you ca contents backward:

```
copy(dice.rbegin(), dice.rend(), out_iter
```

You don't even have to declare a reverse i

Note

Both `rbegin()` and `end()` return the same (`reverse_iterator` versus `iterator`). Si same value (an iterator to the first element),

Reverse pointers have to make a special c initialized to `dice.rbegin()`. What should *r end, you shouldn't try to dereference that add of the first element, `copy()` stops one locatio

```
    // copy from vector to output
    copy(dice.begin(), dice.end(), out_ite
    cout << endl;
    cout <<"Implicit use of reverse iterat
    copy(dice.rbegin(), dice.rend(), out_i
    cout << endl;
    cout <<"Explicit use of reverse iterat
    vector<int>::reverse_iterator ri;
    for (ri = dice.rbegin(); ri != dice.re
        cout << *ri << ' ';
    cout << endl;

    return 0;
}
```

Here is the output of the program in Listin

```
Let the dice be cast!
6 7 2 9 4 11 8 7 10 5
Implicit use of reverse iterator.
5 10 7 8 11 4 9 2 7 6
Explicit use of reverse iterator.
5 10 7 8 11 4 9 2 7 6
```

front_insert_iterator inserts items at the
items in front of the location specified as an a
tor. All three of these iterators are models of t

There are restrictions. A back_insert_it
types that allow rapid insertion at the end. (*R*
section "Container Concepts," later in this ch
further.) The vector class qualifies. A front_
container types that allow constant time inse
doesn't qualify, but the queue class does. The
tions. Thus, you can use it to insert material a
front_insert_iterator does so faster for th

Tip

You can use an insert_iterator to conve
inserts data.

These iterators take the container type as
tainer identifier as a constructor argument. T
a vector<int> container called dice, you us

```
back_insert_iterator<vector<int> > back_i
```

The reason you have to declare the contai
use of the appropriate container method. The

```
{
    using namespace std;
    string s1[4] = {"fine", "fish", "fashi
    string s2[2] = {"busy", "bats"};
    string s3[2] = {"silly", "singers"};
    vector<string> words(4);
    copy(s1, s1 + 4, words.begin());
    for_each(words.begin(), words.end(), o
    cout << endl;
// construct anonymous back_insert_iterato
    copy(s2, s2 + 2, back_insert_iterator<
    for_each(words.begin(), words.end(), o
    cout << endl;

// construct anonymous insert_iterator obj
    copy(s3, s3 + 2, insert_iterator<vecto

    for_each(words.begin(), words.end(), o
    cout << endl;
    return 0;
}
```

predefined iterators multiply the capabilities

Kinds of Containers

The STL has both container concepts and co
gories with names such as container, sequenc
container types are templates you can use to
11 container types are `deque`, `list`, `queue`, `p`
`multimap`, `set`, `multiset`, and `bitset`. (This
container for dealing with data at the bit leve
`unordered_map`, `unordered_multimap`, `unor`
moves `bitset` from the container category i
concepts categorize the types, let's start with

Container Concepts

No type corresponds to the basic container c
common to all the container classes. It's sort
tual because the container classes don't actua
it another way, the container concept lays do
tainer classes must satisfy.

A *container* is an object that stores other o
stored objects may be objects in the OOP se
Data stored in a container is *owned* by the co

`X::value_type`	`T`	The type f
`X u;`		Creates a
`X();`		Creates a container
`X u(a);`		Copy cons `u == a`
`X u = a;`		Same effe
`r = a;`	`X&`	Copy assi `r == a`
`(&a)->~X();`	`void`	Applies de a containe
`a.begin()`	iterator	Returns a element o
`a.end()`	iterator	Returns a end value
`a.size()`	unsigned integral type	Returns a `a.end()`
`a.swap(b)`	`void`	Swaps co

`a == b` has linear complexity because the `==`
element of the container. Actually, that is a w
ferent sizes, no individual comparisons need

Constant-Time and Linear-Time Compl

Imagine a long, narrow box filled with large p
box is open at just one end. Suppose your ta
This is a constant time task. Whether there
one at the end makes no difference.

Now suppose your task is to fetch the packa
time task. If there are 10 packages altogeth
one at the closed end. If there are 100 pack
end. Assuming that you are a tireless worke
task will take 10 times longer than the first

Now suppose your task is to fetch an arbitra
you are supposed to get is the first one at h
packages you have to move is still proportio
so the task still has linear-time complexity.

Replacing the long, narrow box with a similar
to constant-time complexity because then yo
and remove it without moving the others.

Expression	Return Type	Descrip
`X u(rv);`		Move c the valu constru
`X u = rv;`		Same e
`a = rv;`	`X&`	Move a the valu assignm
`a.cbegin()`	`const_iterator`	Returns first ele
`a.cend()`	`const_iterator`	Returns the-end

The difference between copy construction
move construction and move assignment on t
the original unchanged, whereas a move oper
ring ownership without doing any copying W
operations can provide more efficient code th
cusses move semantics further.

Table 16.7 **Sequence Requirements**

Expression	Return Type	Descrip
`X a(n,t);`		Declare
`X(n, t)`		Creates
`X a(i, j)`		Declare range [i
`X(i, j)`		Creates content
`a.insert(p,t)`	iterator	Inserts
`a.insert(p,n,t)`	`void`	Inserts
`a.insert(p,i,j)`	`void`	Inserts
`a.erase(p)`	iterator	Erases
`a.erase(p,q)`	iterator	Erases
`a.clear()`	`void`	Is the s

Because the `deque`, `list`, `queue`, `priorit`
are all models of the sequence concept, they

push_front() is defined for list and deque
insert a new value at the front of a vector of 1
move element 99 to position 100, and then yo
and so on. This is an operation with linear-tim
would take 100 times as long as moving a sing
are supposed to be implemented only if they c
plexity. The design for lists and double-ended
added to the front without moving the other
implement push_front() with constant-time
push_front() and push_back().

Let's take a closer look at the six sequence

vector

You've already seen several examples using the
vector header file. In brief, vector is a class r
automatic memory management that allows th
cally, growing and shrinking as elements are ad
to elements. Elements can be added to or rem
insertion and removal from the beginning and

In addition to being a sequence, a vector c
tainer concept. This adds two more class metho
element of the reversed sequence, and rend()

Figure 16.4 push_fro

The iterator returned by the two method
Recall that incrementing such an iterator ca
in reverse order.

The vector template class is the simplest
type that should be used by default unless th
the particular virtues of the other types.

deque

The deque template class (declared in the de
queue, a type often called a *deque* (pronounce
STL, it's a lot like a vector container, suppo
that inserting and removing items from the b
time operations instead of being linear-time
most operations take place at the beginning
using a deque data structure.

The goal of constant-time insertion and r
design of a deque object more complex than
offer random access to elements and linear-ti
of a sequence, the vector container should a

complete list of STL methods and functions, s
parameter is one you normally don't have to v

Table 16.9 **Some `list` Member Functions**

Function	Desc
void merge(list<T, Alloc>& x)	Merg be s invok linea
void remove(const T & val)	Rem funct
void sort()	Sorts comp
void splice(iterator pos, list<T, Alloc> x)	Inser pos, time
void unique()	Colla elem linea

```
cout << "List one: ";
for_each(one.begin(),one.end(), outin
cout << endl << "List two: ";
for_each(two.begin(), two.end(), outi
cout << endl << "List three: ";
for_each(three.begin(), three.end(),
three.remove(2);
cout << endl << "List three minus 2s:
for_each(three.begin(), three.end(),
three.splice(three.begin(), one);
cout << endl << "List three after spl
for_each(three.begin(), three.end(),
cout << endl << "List one: ";
for_each(one.begin(), one.end(), outi
three.unique();
cout << endl << "List three after uni
for_each(three.begin(), three.end(),
three.sort();
three.unique();
cout << endl << "List three after sor
for_each(three.begin(), three.end(),
two.sort();
three.merge(two);
```

```
for (auto x : three) cout << x << " ";
```

The main difference between `insert()` an
of the original range into the destination, whe
into the destination. Thus, after the contents o
(The `splice()` method has additional prototy
of elements.) The `splice()` method leaves ite
iterator to point to an element in `one`, that ite
`splice()` relocates it in `three`.

Notice that `unique()` only reduces adjacen
program executes `three.unique()`, `three` stil
weren't adjacent. But applying `sort()` and the
gle appearance.

There is a nonmember `sort()` function (L
iterators. Because the trade-off for rapid insert
use the nonmember `sort()` function with a l
version that works within the restrictions of th

The `list` Toolbox

The `list` methods form a handy toolbox. Sup
ing lists to organize. You could sort each list, m
remove multiple entries.

add an element to the rear of a queue, remov
the values of the front and rear elements, che
a queue is empty. Table 16.10 lists these opera

Note that pop() is a data removal method
use a value from a queue, you first use front
to remove it from the queue.

Table 16.10 **queue Operations**

Method	Description
bool empty() const	Returns true i
size_type size() const	Returns the nu
T& front()	Returns a refer
T& back()	Returns a refer
void push(const T& x)	Inserts x at the
void pop()	Removes the e

priority_queue

The priority_queue template class (declare
class. It supports the same operations as queu
that with priority_queue, the largest item g

`bool empty() const`	Returns `true` i
`size_type size() const`	Returns the nu
`T& top()`	Returns a refer
`void push(const T& x)`	Inserts `x` at the
`void pop()`	Removes the e

Much as with `queue`, if you want to use a
retrieve the value, and then you use `pop()` to

`array` (C++11)

The `array` template class, introduced in Chap
not an STL container because it has a fixed si
tainer, such as `push_back()` and `insert()`, ar
functions that do make sense, such as `operato`
use many standard STL algorithms, such as `co`

Associative Containers

An *associative container* is another refinement o
tainer associates a value with a key and uses th
values could be structures representing emplo
office number, home and work phones, health

more than one value with the same key. For e
`multiset` object could hold, say `1, 2, 2, 2, 3, 5`

For the `map` type, the value type is differen
with only one value per key. The `multimap` t
associated with multiple values.

There's too much information about these
G does list the methods), so let's just look at a
example that uses `multimap`.

A `set` Example

The STL `set` models several concepts. It is an
and the keys are unique, so it can hold no mo
and `list`, `set` uses a template parameter to p

```
set<string> A;  // a set of string object
```

An optional second template argument ca
or object to be used to order the key. By defa
used. Older C++ implementations may not p
explicit template parameter:

```
set<string, less<string> > A;  // older i
```

automatically satisfy the precondition for usin
tainer be sorted. The `set_union()` function ta
define a range in one set, the second two defin
ator is an output iterator that identifies a locat
example, to display the union of sets `A` and `B`, y

```
set_union(A.begin(), A.end(), B.begin(), B
          ostream_iterator<string, char>
```

Suppose you want to place the result into a
you would want the last argument to be an ite
`C.begin()`, but that doesn't work for two reas
treat keys as constant values, so the iterator ret
and can't be used as an output iterator. The sec
that `set_union()`, like `copy()`, overwrites exi
container to have sufficient space to hold the
satisfy that requirement. But the `insert_iter`
problems. Earlier you saw that it converts copy
iterator concept, so you can use it to write to
mous `insert_iterator` to copy information
of the container and an iterator as arguments:

```
set_union(A.begin(), A.end(), B.begin(), B
          insert_iterator<set<string> >(C
```

```
#include <string>
#include <set>
#include <algorithm>
#include <iterator>

int main()
{
    using namespace std;
    const int N = 6;
    string s1[N] = {"buffoon", "thinkers"
    string s2[N] = {"metal", "any", "food

    set<string> A(s1, s1 + N);
    set<string> B(s2, s2 + N);

    ostream_iterator<string, char> out(co
    cout << "Set A: ";
    copy(A.begin(), A.end(), out);
    cout << endl;
    cout << "Set B: ";
    copy(B.begin(), B.end(), out);
    cout << endl;
```

```
    cout << endl;

    cout << "Showing a range:\n";
    copy(C.lower_bound("ghost"),C.upper_bo
    cout << endl;

    return 0;
}
```

Here is the output of the program in Listin

```
Set A: buffoon can for heavy thinkers
Set B: any deliver elegant food for metal
Union of A and B:
any buffoon can deliver elegant food for h
Intersection of A and B:
for
Difference of A and B:
buffoon can heavy thinkers
Set C:
any buffoon can deliver elegant food for h
Set C after insertion:
any buffoon can deliver elegant food for g
Showing a range:
grungy heavy metal
```

```
multimap<int,string> codes;
```

An optional third template argument can
an object to be used to order the key. By defa
used with the key type as its parameter. Olde
template parameter explicitly.

To keep information together, the actual v
data type into a single pair. To do this, the ST
class for storing two kinds of values in a singl
`datatype` is the type of the stored data, the v
`datatype>`. For example, the value type for t
`pair<const int, string>`.

Suppose that you want to store city names
fit the `codes` declaration, which uses an `int` f
approach is to create a pair and then insert it

```
pair<const int, string> item(213, "Los An
codes.insert(item);
```

Or you can create an anonymous `pair` ob

```
codes.insert(pair<const int, string> (213
```

Because items are sorted by key, there's no tion feature, which allows you to simplify the

```
auto range = codes.equal_range(718);
cout << "Cities with area code 718:\n";
for (auto it = range.first; it != range.se
    cout <<  (*it).second    << endl;
```

Listing 16.14 demonstrates most of these te
some of the code writing.

Listing 16.14 **multmap.cpp**

```
// multmap.cpp -- use a multimap
#include <iostream>
#include <string>
#include <map>
#include <algorithm>

typedef int KeyType;
typedef std::pair<const KeyType, std::stri
typedef std::multimap<KeyType, std::string

int main()
{
```

```
        = codes.equal_range(718);
    cout << "Cities with area code 718:\n
    for (it = range.first; it != range.se
        cout <<  (*it).second    << endl;

    return 0;
}
```

Here is the output of the program in Listi

```
Number of cities with area code 415: 2
Number of cities with area code 718: 2
Number of cities with area code 510: 2
Area Code   City
    415     San Francisco
    415     San Rafael
    510     Oakland
    510     Berkeley
    718     Brooklyn
    718     Staten Island
Cities with area code 718:
Brooklyn
Staten Island
```

```
{
private:
    double slope;
    double y0;
public:
    Linear(double sl_ = 1, double y_ = 0)
        : slope(sl_), y0(y_) {}
    double operator()(double x) {return y0
};
```

The overloaded `()` operator then allows yo

```
Linear f1;
Linear f2(2.5, 10.0);
double y1 = f1(12.5);   // right-hand side
double y2 = f2(0.4);
```

Here `y1` is calculated using the expression expression `10.0 + 2.5 * 0.4`. In the express `slope` come from the constructor for the obje ment to `operator()()`.

Remember the `for_each` function? It appl a range:

```
for_each(books.begin(), books.end(), ShowR
```

Just as the STL defines concepts for containe

- A *generator* is a functor that can be calle
- A *unary function* is a functor that can be
- A *binary function* is a functor that can be

For example, the functor supplied to for_
it is applied to one container element at a tim
Of course, these concepts come with refin

- A unary function that returns a bool v
- A binary function that returns a bool v

Several STL functions require predicate or
Listing 16.9 uses a version of sort() that tak

```
bool WorseThan(const Review & r1, const R
...
sort(books.begin(), books.end(), WorseTha
```

The list template has a remove_if() me
It applies the predicate to each member in th
for which the predicate returns true. For exa
elements greater than 100 from the list three

objects to different cut-off values to be used i
trates the technique.

Listing 16.15 **functor.cpp**

```
// functor.cpp -- using a functor
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

template<class T>  // functor class define
class TooBig
{
private:
    T cutoff;
public:
    TooBig(const T & t) : cutoff(t) {}
    bool operator()(const T & v) { return
};

void outint(int n) {std::cout << n << " ";
```

```
    for_each(etcetera.begin(), etcetera.e
    cout << endl;
    return 0;
}
```

One functor (`f100`) is a declared object, ar anonymous object created by a constructor c Listing 16.15:

```
Original lists:
50 100 90 180 60 210 415 88 188 201
50 100 90 180 60 210 415 88 188 201
Trimmed lists:
50 100 90 60 88
50 100 90 180 60 88 188
```

Suppose that you already have a template

```
template <class T>
bool tooBig(const T & val, const T & lim)
{
    return val > lim;
}
```

As noted in the listing, C++11's initialize
can replace

```
int vals[10] = {50, 100, 90, 180, 60, 210,
list<int> yadayada(vals, vals + 10); // ra
list<int> etcetera(vals, vals + 10);
```

with this:

```
list<int> yadayada = {50, 100, 90, 180, 60
list<int> etcetera {50, 100, 90, 180, 60,
```

Predefined Functors

The STL defines several elementary functors.
values and comparing two values for equality.
functions that take functions as arguments. Fo
function. It has two versions. The first version
ments are iterators that specify a range in a co
that approach.) The third is an iterator that sp
functor that is applied to each element in the
result. For example, consider the following:

```
const int LIM = 5;
double arr1[LIM] = {36, 39, 42, 45, 48};
```

But then you'd have to define a separate fu
define a template, except that you don't have
functional (formerly function.h) header d
including one called plus<>().

Using the plus<> class for ordinary additi

```
#include <functional>
...
plus<double> add;  // create a plus<doubl
double y = add(2.2, 3.4); // using plus<d
```

But it makes it easy to provide a function

```
transform(gr8.begin(), gr8.end(), m8.begi
```

Here, rather than create a named object, th
to construct a functor to do the adding. (The
structor; what's passed to transform() is the

The STL provides functor equivalents for
ical operators. Table 16.12 shows the names f
used with the C++ built-in types or with an
sponding operator.

Caution

Older C++ implementations use the functor n

Adaptable Functors and Function

The predefined functors in Table 16.12 are all concepts: adaptable generators, adaptable unar adaptable predicates, and adaptable binary pre

What makes a functor adaptable is that it c argument types and return type. The member `first_argument_type`, and `second_argumen` like. For example, the return type of a `plus<i` `plus<int>::result_type`, and this would be

The significance of a functor being adaptab adapter objects, which assume the existence o function with an argument that is an adaptabl to declare a variable that matches the function

simplify using the `binder1st` class. You give i
struct a `binder1st` object, and it returns an o
vert the binary function `multiplies()` to a u
2.5. Just do this:

```
bind1st(multiplies<double>(), 2.5)
```

Thus, the solution to multiplying every el
results is this:

```
transform(gr8.begin(), gr8.end(), out,
        bind1st(multiplies<double>(), 2.5
```

The `binder2nd` class is similar, except that
ment instead of to the first. It has a helper fun
to `bind1st`.

```
    for_each(gr8.begin(), gr8.end(), Show)
    cout << endl;
    cout << "m8: \t";
    for_each(m8.begin(), m8.end(), Show);
    cout << endl;

    vector<double> sum(LIM);
    transform(gr8.begin(), gr8.end(), m8.b
              plus<double>());
    cout << "sum:\t";
    for_each(sum.begin(), sum.end(), Show)
    cout << endl;

    vector<double> prod(LIM);
    transform(gr8.begin(), gr8.end(), prod
              bind1st(multiplies<double>()
    cout << "prod:\t";
    for_each(prod.begin(), prod.end(), Sho
    cout << endl;

    return 0;
}
```

ranges to be processed and to identify where
object argument to be used as part of the dat

There are two main generic components
use templates to provide generic types. Secon
representation for accessing data in a contain
a container that holds type `double` values in
values in a linked list, or with a container tha
ture, such as is used by `set`. Because pointers
such as `copy()` can be used with ordinary ar

The uniform container design allows mea
different kinds. For example, you can use `cop`
a `vector` object, from a `vector` object to a `l`
object. You can use `==` to compare different k
`vector`. This is possible because the overload
compare contents, so a `deque` object and a `ve`
same content in the same order.

Algorithm Groups

The STL divides the algorithm library into f

- Nonmodifying sequence operations
- Mutating sequence operations

General Properties of Algorithms

As you've seen again and again in this chapter
iterator ranges. The function prototype indica
tors. For example, the copy() function has th

```
template<class InputIterator, class Output
OutputIterator copy(InputIterator first, I
                    OutputIterator result
```

Because the identifiers InputIterator and
they just as easily could have been T and U. H
template parameter names to indicate the con
laration tells you that the range parameters m
iterator indicating where the result goes must

One way of classifying algorithms is on th
is placed. Some algorithms do their work in p
when the sort() function is finished, the resu
inal data did. So sort() is an *in-place algorithm*
result of its work to another location, so it is a
tion can do both. Like copy(), it uses an outp
Unlike copy(), transform() allows the outp
input range, so it can copy the transformed va

action conditionally, depending on the result
ment. These versions typically append `_if` to
`replace_if()` replaces an old value with a n
value returns the value `true`. Here's the proto

```
template<class ForwardIterator, class Pre
void replace_if(ForwardIterator first, Fo
             Predicate pred, const T& ne
```

(Recall that a predicate is a unary functio
version called `replace_copy_if()`. You can
prototype is like.

As with `InputIterator`, `Predicate` is a t
easily be called `T` or `U`. However, the STL ch
that the actual argument should be a model
uses terms such as `Generator` and `BinaryPre`
model other function object concepts. Keep
can remind you what the iterator or functor
thing the compiler can check. If you use the
a long list of error messages as the compiler t

```
    string letters;
    cout << "Enter the letter grouping (qu
    while (cin >> letters && letters != "q
    {
        cout << "Permutations of " << lett
        sort(letters.begin(), letters.end
        cout << letters << endl;
        while (next_permutation(letters.be
            cout << letters << endl;
        cout << "Enter next sequence (quit
    }
    cout << "Done.\n";
    return 0;
}
```

Here's a sample run of the program in List

```
Enter the letter grouping (quit to quit):
Permutations of awl
alw
awl
law
lwa
wal
```

```
la.remove(4);  // remove all 4s from the
```

After this method call, all elements with th
list is automatically resized.

There is also an STL algorithm called `rem`
invoked by an object, it takes range argumen
the function could look like this:

```
remove(lb.begin(), lb.end(), 4);
```

However, because this `remove()` is not a m
Instead, it makes sure all the nonremoved ite
returns an iterator to the new past-the-end v
list size. For example, you can use the list `era`
describes the part of the list that is no longer
process works.

Listing 16.18 **listrmv.cpp**

```
// listrmv.cpp -- applying the STL to a s
#include <iostream>
#include <list>
#include <algorithm>
```

```cpp
    cout << "After using the erase() metho
    cout << "lb:\t";
    for_each(lb.begin(), lb.end(), Show);
    cout << endl;
    return 0;
}

void Show(int v)
{
    std::cout << v << ' ';
}
```

Here's the output of the program in Listin

```
Original list contents:
    4 5 4 2 2 3 4 8 1 4
After using the remove() method:
la: 5 2 2 3 8 1
After using the remove() function:
lb: 5 2 2 3 8 1 4 8 1 4
After using the erase() method:
lb: 5 2 2 3 8 1
```

```
vector<string> words;
string input;
while (cin >> input && input != "quit")
    words.push_back(input);
```

What about getting the alphabetic word l
`unique()`, but that approach overwrites the d
algorithm. There is an easier way that avoids
object and copy (using an insert iterator) the
matically sorts its contents, which means you
only one copy of a key, so that takes the plac
called for ignoring the case differences. One
instead of `copy()` to copy data from the vect
tion, you can use one that converts a string t

```
set<string> wordset;
transform(words.begin(), words.end(),
    insert_iterator<set<string> > (wordse
```

The `ToLower()` function is easy to write.
`tolower()` function to each element in the s
destination. Remember, `string` objects, too,
returning the string as a reference means the
out having to make copies. Here's the code f

```
    wordmap.insert(pair<string, int>(*si,
    words.end(), *si)));
```

The map class has an interesting feature:Yo
as indexes to access the stored values. For exa
value associated with the key "the", which in
the string "the". Because the wordset contai
can use the following code as an alternative a

```
for (si = wordset.begin(); si != wordset.e
    wordmap[*si] = count(words.begin(), wo
```

Because si points to a string in the words
a key for wordmap. This code places both keys

Similarly, you can use the array notation to

```
for (si = wordset.begin(); si != wordset.e
    cout << *si << ": " << wordmap[*si] <<
```

If a key is invalid, the corresponding value

Listing 16.19 puts these ideas together and
the three containers (a vector with the input,
word count).

```
        words.push_back(input);

    cout << "You entered the following wo
    for_each(words.begin(), words.end(), 
    cout << endl;

    // place words in set, converting to 
    set<string> wordset;
    transform(words.begin(), words.end(),
        insert_iterator<set<string> > (wo
        ToLower);
    cout << "\nAlphabetic list of words:\
    for_each(wordset.begin(), wordset.end
    cout << endl;

    // place word and frequency in map
    map<string, int> wordmap;
    set<string>::iterator si;
    for (si = wordset.begin(); si != word
        wordmap[*si] = count(words.begin(

    // display map contents
    cout << "\nWord frequency:\n";
```

```
You entered the following words:
The dog saw the cat and thought the cat fa

Alphabetic list of words:
and cat dog fat perfect saw the thought

Word frequency:
and: 1
cat: 4
dog: 1
fat: 1
perfect: 1
saw: 1
the: 5
thought: 2
```

The moral here is that your attitude when
much code as possible. STL's generic and flex
Also the STL designers are algorithm people
ciency. So the algorithms are well chosen and

class template, on the other hand, is oriented
part of the STL. It doesn't have push_back()
does provide a simple, intuitive interface for
is designed as a substitute for the built-in arr
ciency of that type with a better, safer interfa
push_back() and insert(), but it does offe
begin(), end(), rbegin(), and rend(), maki
objects.

Suppose, for example, that you have these

```
vector<double> ved1(10), ved2(10), ved3(1
array<double, 10> vod1, vod2, vod3;
valarray<double> vad1(10), vad2(10), vad3
```

Furthermore, assume that ved1, ved2, vod
values. Suppose you want to assign the sum o
element of a third array, and so on. With the

```
transform(ved1.begin(), ved1.end(), ved2.
          plus<double>());
```

You can do the same with the array class

```
transform(vod1.begin(), vod1.end(), vod2.
          plus<double>());
```

```
log);
```

The valarray class overloads the usual ma
and to return a valarray object, so you can u

```
vad3 = log(vad1);     // log() overloaded
```

Or you could use the apply() method, wh

```
vad3 = vad1.apply(log);
```

The apply() method doesn't alter the inv
that contains the resulting values.

The simplicity of the valarray interface i
step calculation:

```
vad3 = 10.0* ((vad1 + vad2) / 2.0 + vad1 *
```

The vector-STL version is left as an exerc

The valarray class also provides a sum()
valarray object, a size() method that retur
method that returns the largest value in an ob
smallest value.

As you can see, valarray has a clear notat
cal operations, but it is also much less versatile

vide vad and vad + 10, as the following cod

```
sort(vad, vad + 10);  // NO, vad an objec
```

You can use the address operator:

```
sort(&vad[0], &vad[10]);  // maybe?
```

But the behavior of using a subscript one doesn't necessarily mean using &vad[10] wor pilers used to test this code.) But it does mea fail, you probably would need a very unlikely against the end of the block of memory set a sion depended on your code, you might not

C++11 remedies the situation by providi that take a valarray object as an argument. vad.begin(). These functions return values requirements:

```
sort(begin(vad), end(vad)); // C++11 fix!
```

Listing 16.20 illustrates some of the relativ classes. It uses push_back() and the automat Then after sorting the numbers, the program valarray object of the same size and does a

```
    sq_rts = sqrt(numbers);
    valarray<double> results(size);
    results = numbers + 2.0 * sq_rts;
    cout.setf(ios_base::fixed);
    cout.precision(4);
    for (i = 0; i < size; i++)
    {
        cout.width(8);
        cout << numbers[i] << ": ";
        cout.width(8);
        cout << results[i] << endl;
    }
    cout << "done\n";
    return 0;
}
```

Here is a sample run of the program in Lis

```
Enter numbers (<=0 to quit):
3.3 1.8 5.2 10 14.4 21.6 26.9 0
  1.8000:   4.4833
  3.3000:   6.9332
  5.2000:   9.7607
 10.0000:  16.3246
```

This special subscripting facility allows yo
to represent two-dimensional data. For exam
with 4 rows and 3 columns. You can store the
object. Then a `slice(0,3,1)` object used as
and 2—that is, the first row. Similarly, a `slic`
0, 3, 6, and 9—that is, the first column. Listin

Listing 16.21 **vslice.cpp**

```
// vslice.cpp -- using valarray slices
#include <iostream>
#include <valarray>
#include <cstdlib>

const int SIZE = 12;
typedef std::valarray<int> vint;     // s
void show(const vint & v, int cols);
int main()
{
    using std::slice;                // f
    using std::cout;
    vint valint(SIZE);               // t

    int i;
```

```
    using std::cout;
    using std::endl;

    int lim = v.size();
    for (int i = 0; i < lim; ++i)
    {
        cout.width(3);
        cout << v[i];
        if (i % cols == cols - 1)
            cout << endl;
        else
            cout << ' ';
    }
    if (lim % cols != 0)
        cout << endl;
}
```

The + operator is defined for valarray ob
single int element, such as valint[1]. But as
ator isn't defined for slice-subscripted valar
Therefore, the program constructs full objects

```
vint(valint[slice(1,4,3)])    // calls a s
```

```
12   2  10
19   9  10
```

Because values are set using `rand()`, differ
different values.

There's more, including the `gslice` class t
should be enough to give you a sense of wha

The `initializer_list` Templa

The `initializer_list` template is another
use the initializer-list syntax to initialize an S

```
std::vector<double> payments {45.99, 39.2
```

This would create a container for four ele
values in the list. What makes this possible is
tors taking an `initializer_list<T>` argum
has a constructor that accepts an `initialize`
ous declaration is the same as this:

```
std::vector<double> payments({45.99, 39.2
```

The usual list restrictions on narrowing ap

```
std::vector<int> values = {10, 8, 5.5};  /
```

Here, the element type is `int`, and the imp allowed.

It doesn't make sense to provide an `initia` meant to handle lists of varying sizes. For inst constructor for a class taking a fixed number ration does not provide an `initializer_lis`

```
class Position
{
private:
    int x;
    int y;
    int z;
public:
    Position(int xx = 0, int yy  = 0, int
             : x(xx), y(yy), z(zz) {}
    // no initializer_list constructor
    ...
};
```

```
    using std::cout;

    cout << "List 1: sum = " << sum({2,3,
         <<", ave = " << average({2,3,4})
    std::initializer_list<double> dl = {1
    cout << "List 2: sum = " << sum(dl)
         <<", ave = " << average(dl) << '
    dl = {16.0, 25.0, 36.0, 40.0, 64.0};
    cout << "List 3: sum = " << sum(dl)
         <<", ave = " << average(dl) << '
    return 0;
}

double sum(std::initializer_list<double>
{
    double tot = 0;
    for (auto p = il.begin(); p !=il.end(
        tot += *p;
    return tot;
}

double average(const std::initializer_lis
{
    double tot = 0;
```

choice is not a major performance issue. (The

The function argument can be a list literal

like dl.

The iterator types for initializer_list

in a list:

```
*dl.begin() = 2011.6;             // no
```

But, as Listing 16.22 shows, you can attach

```
dl = {16.0, 25.0, 36.0, 40.0, 64.0};  // a
```

However, the intended use of the initial

constructor or some other function.

Summary

C++ includes a powerful set of libraries that
gramming problems and the tools to simplify
vides a convenient means to handle strings as
management and a host of methods and funct
these methods and functions allow you to co
another, reverse a string, search a string for ch
and output operations.

underlying container class to give it the chara
class template name. Thus, `stack`, although b
and removal only at the top of the stack. C+
`unordered_multiset`, `unordered_map`, and `u

Some algorithms are expressed as contain
as general, nonmember functions. This is mad
between containers and algorithms. One adv
be just one `for_each()` or `copy()` function,
each container. A second advantage is that ST
containers, such as ordinary arrays, `string` ob
design consistent with the STL iterator and c

Both containers and algorithms are charac
or need. You should check that a container fe
algorithm's needs. For example, the `for_each`
minimal requirements are met by all the STL
random access iterators, which not all contain
offer a specialized method as an option if it d
algorithm. For example, the `list` class has a s
iterators, so it can use that method instead of

The STL also provides function objects, o
operator is overloaded—that is, for which the

```
    RQ1(const char * s)
    {st = new char [strlen(s) + 1];
    RQ1(const RQ1 & rq)
    {st = new char [strlen(rq.st) +
    ~RQ1() {delete [] st};
    RQ & operator=(const RQ & rq);
    // more stuff
};
```

Convert this to a declaration that uses a
longer need explicit definitions?

2. Name at least two advantages `string`
of ease-of-use.

3. Write a function that takes a reference
converts the `string` object to all upper

4. Which of the following are not exampl
tically) of `auto_ptr`? (Assume that the

```
auto_ptr<int> pia(new int[20]);
auto_ptr<string> (new string);
int rigue = 7;
auto_ptr<int>pr(&rigue);
auto_ptr dbl (new double);
```

1. A *palindrome* is a string that is the same
and "otto" are rather short palindromes
string and that passes to a `bool` functio
should return `true` if the string is a pal
don't worry about complications such
That is, this simple version should rejec
to scan the list of string methods in Ap

2. Do the same problem as given in Prog
complications such as capitalization, sp
Adam" should test as a palindrome. For
the string to "madamimadam" and the
forget the useful `cctype` library. You m
although not necessary.

3. Redo Listing 16.3 so that it gets it wor
`vector<string>` object instead of an a
`push_back()` to copy how ever many
`vector<string>` object and use the `si`
the word list. Because the program sho
you should use the `>>` operator rather
words separated by spaces, tabs, or new

ments. The first should be the number o
should be the number of spots selected
`vector<int>` object that contains, in so
For example, you could use the functio

```
vector<int> winners;
winners = Lotto(51,6);
```

This would assign to `winners` a vector
from the range 1 through 51. Note that
job because it may produce duplicate va
vector that contains all the possible valu
beginning of the shuffled vector to obta
that lets you test the function.

8. Mat and Pat want to invite their friends
that does the following:

   - Allows Mat to enter a list of his fr
     container and then displayed in so
   - Allows Pat to enter a list of her fr
     ond container and then displayed
   - Creates a third container that me
     displays the contents of this conta

```
clock_t end = clock();
cout << (double)(end - start)/CLOCK
```

This is by no means a definitive test be
factors, including available memory, wh
size of the array or list. (One would ex
array over the list to increase with the
you have a choice between a default bu
for the measurement. With today's spee
as large an array as possible to get mean
100,000 elements, 1,000,000 elements,

10. Modify Listing 16.9 (`vect3.cpp`) as fol

    a. Add a `price` member to the `Rev`

    b. Instead of using a `vector` of `Rev` of `shared_ptr<Review>` objects initialized with a pointer returne

    c. Follow the input stage with a loo for displaying books: in original increasing ratings, in order of dec in order of decreasing price, and

- `istream` class methods
- Stream states
- File I/O
- Using the `ifstream` class for input fro
- Using the `ofstream` class for output to
- Using the `fstream` class file input and
- Command-line processing
- Binary files
- Random file access
- Incore formatting

Discussing C++ input and output (I/O) p
every program uses input and output, and lea
tasks facing someone learning a computer la
of its more advanced language features to im
derived classes, function overloading, virtual
tance. Thus, to really understand C++ I/O, y
started, the early chapters of this book outlin
object `cin` and the `ostream` class object `cou`
using `ifstream` and `ofstream` objects for fil

ments of the target computer. In practice, mo
library functions originally developed for the
recognition of this I/O package, called the St
a mandatory component of the standard C lib
you're familiar with the family of C functions
them in C++ programs. (Newer implementat
these functions.)

However, C++ relies on a C++ solution r
solution is a set of classes defined in the `iost`
(formerly `fstream.h`) header files. This class l
definition (`cin` and `istream` are not keyword
rules for how to do things, such as create class
ate by following those rules. But just as C imp
of functions, C++ comes with a standard libr
library was an informal standard consisting so
and `fstream` header files. The ANSI/ISO C+
library as a standard class library and to add a
cussed in Chapter 16, "The `string` Class and
ter discusses standard C++ I/O. But first, let's
C++ I/O.

a device, such as a keyboard.) Similarly, mana
stream to the program and associating some
plumbing with bytes instead of water (see Fi

Usually, input and output can be handled
block of memory used as an intermediate, te
information from a device to a program or f
such as disk drives transfer information in bl
often process information 1 byte at a time. T
rates of information transfer. For example, as
number of dollar signs in a hard-disk file. Th
file, process it, read the next character from t
at a time from a disk requires a lot of hardwa
is to read a large chunk from the disk, store t
one character at a time. Because it is much q
memory than from a hard disk, this approach
ware. Of course, after the program reaches th
read another chunk of data from the disk. Th
voir that collects megagallons of runoff wate
your home at a more civilized rate of flow (s
gram can first fill the buffer and then transfe
ing the buffer for the next batch of output. T
can come up with your own plumbing-base

Keyboard input provides one character at a
need a buffer to help match different data tran
allows the user to back up and correct input b
program normally flushes the input buffer wh
in this book don't begin processing input unti
a C++ program normally flushes the output b
ter. Depending on the implementation, a prog
too, such as at impending input. That is, when
flushes any output currently in the output bu
tent with ANSI C should behave in that man

Streams, Buffers, and the iostre

The business of managing streams and buffers
`iostream` (formerly `iostream.h`) file brings i
manage streams and buffers for you. The C++
plates in order to support both `char` and `wcha`
`char32_t` specializations. By using the `typede`
tions of these templates mimic the traditional
are some of those classes (see Figure 17.3):

Figure 17.2 A s

- The streambuf class provides memory filling the buffer, accessing buffer conte buffer memory.
- The ios_base class represents general open for reading and whether it's a bin
- The ios class is based on ios_base, an streambuf object.
- The ostream class derives from the io
- The istream class derives from the io
- The iostream class is based on the ist both input and output methods.

To use these facilities, you use objects of t an ostream object such as cout to handle ou automatically creates a buffer, and associates i member functions available to you.

Derived class
Multiple inheritance

Figure 17.3 Sc

Redefining I/O

The ISO/ANSI C++98 Standard revised I/O a
ostream.h to ostream, with ostream placin
the I/O classes have been rewritten. To be ar
handle international character sets that requi
guage added the wchar_t (or "wide") charac
row") type. C++11 adds char16_t and char
facilities. Rather than develop two (or, now, fo
committee developed a template set of I/O c
traits<charT> > and basic_ostream<ch
traits<charT> template, in turn, is a templ
character type, such as how to compare for e
provides char and wchar_t specializations c
ostream are typedefs for char specializatio
wchar_t specializations. For example, there
streams. The ostream header file contains th

Certain type-independent information that use
moved to the new ios_base class. This inclu
ios::fixed, which is now ios_base::fixe
aren't available in the old ios.

ing information relating to output, such as th
the number of places after the decimal to use
gers, and the address of a `streambuf` object th
output flow. A statement such as the followin
`"Bjarne free"` into the buffer managed by c

```
cout << "Bjarne free";
```

The `ostream` class defines the `operator<<`
`ostream` class also supports the `cout` data me
such as the ones this chapter discusses later. F
from the buffer is directed to the standard ou
operating system. In short, one end of a strea
is connected to the standard output, and the
`streambuf` object, manages the flow of bytes

Redirection

The standard input and output streams norma
But many operating systems, including Unix,
facility that lets you change the associations fo
Suppose, for example, that you have an execu
gram called `counter.exe` that counts the num
result. (From most versions of Windows you c
Command Prompt icon to open a command-
like this:

this redirection syntax. (All of these other than
characters between the redirection operators a

The standard output stream, represented by
output. The standard error streams (represente
gram's error messages. By default, all three of t
tor. But redirecting the standard output doesn
these objects to print an error message, a prog
screen even if the regular `cout` output is redir
code fragment:

```
if (success)
    std::cout << "Here come the goodies!\n
else
{
    std::cerr << "Something horrible has h
    exit(1);
}
```

If redirection is not in effect, whichever m
however, the program output has been redirec
would go to the file but the second message, i
way, some operating systems permit redirectin
for example, the `2>` operator redirects the stan

Most often, this book has used cout with the
operator:

```
int clients = 22;
cout << clients;
```

In C++, as in C, by default the << operato
Appendix E, "Other Operators"). An express
representation of x and shift all the bits three
a lot to do with output. But the ostream clas
loading to output for the ostream class. In th
tion operator instead of the left-shift operato
role through its visual aspect, which suggests
tion operator is overloaded to recognize all th

- unsigned char
- signed char
- char
- short
- unsigned short
- int
- unsigned int

also indicates that the function returns a refere
makes it possible to concatenate output, as in

```
cout << "I'm feeling sedimental over " <<
```

If you're a C programmer who has suffered
and the problems that arise when you mismat
almost sinfully easy. (And C++ input, of cours

Output and Pointers

The `ostream` class defines insertion operator f

- `const signed char *`
- `const unsigned char *`
- `const char *`
- `void *`

C++ represents a string, don't forget, by us
The pointer can take the form of the name of
`to-char` or of a quoted string. Thus, all the fo

```
char name[20] = "Dudly Diddlemore";
char * pn = "Violet D'Amore";
cout << "Hello!";
```

tions say that the reference is to the object us
operator function's return value is the same o
`cout << "potluck"` returns the `cout` object
output by using insertion. For example, consi

```
cout << "We have " << count << " unhatche
```

The expression `cout << "We have "` disp
reducing the statement to the following:

```
cout << count << " unhatched chickens.\n"
```

Then the expression `cout << count` displ
returns `cout`, which can then handle the fina
17.4). This design technique really is a nice fe
loading the `<<` operator in the previous chapt

The Other `ostream` Methods

Besides the various `operator<<()` functions,
method for displaying characters and the `wri`

Originally, the put() method had the follo

```
ostream & put(char);
```

The current standard is equivalent, except invoke it by using the usual class method nota

```
cout.put('W');      // display the W chara
```

Here cout is the invoking object and put(operator functions, this function returns a refe concatenate output with it:

```
cout.put('I').put('t'); // displaying It w
```

The function call cout.put('I') returns for the put('t') call.

Given the proper prototype, you can use pu than char, such as int, and let function proto to the correct type char value. For example, y

```
cout.put(65);        // display the A char
cout.put(66.3);      // display the B char
```

the second argument indicates how many ch
write() invokes the char specialization, so th
shows how the write() method works.

Listing 17.1 **write.cpp**

```
// write.cpp -- using cout.write()
#include <iostream>
#include <cstring>  // or else string.h

int main()
{
    using std::cout;
    using std::endl;
    const char * state1 = "Florida";
    const char * state2 = "Kansas";
    const char * state3 = "Euphoria";
    int len = std::strlen(state2);
    cout << "Increasing loop index:\n";
    int i;
    for (i = 1; i <= len; i++)
    {
        cout.write(state2,i);
        cout << endl;
    }
```

```
Kansas
Decreasing loop index:
Kansas
Kansa
Kans
Kan
Ka
K
Exceeding string length:
Kansas Euph
```

Note that the `cout.write()` call returns th
method returns a reference to the object that
invokes it. This makes it possible to concatena
by its return value, `cout`:

```
cout.write(state2,i) << endl;
```

Also, note that the `write()` method doesn
when it reaches the null character. It simply p
if that goes beyond the bounds of a particular
string `"Kansas"` with two other strings so tha
data. Compilers differ in the order in which th
align memory. For example, `"Kansas"` occupi

Then the program *flushes* the buffer, sending
new data. Typically, a buffer is 512 bytes or an
time-saver when the standard output is conn
don't want a program to access the hard disk
effective to collect 512 bytes in a buffer and
operation.

For screen output, however, filling the buf
inconvenient if you had to reword the messa
sumed the prerequisite 512 bytes to fill a buf
the program doesn't necessarily wait until th
the buffer, for example, normally flushes the
implementations flush the buffer when input
lowing code:

```
cout << "Enter a number: ";
float num;
cin >> num;
```

The fact that the program expects input c
flush the `"Enter a number: "` message) imm
a newline character. Without this feature, the
prompting the user with the `cout` message.

If your implementation doesn't flush outp
ing by using one of two manipulators. The `fl`

to hold the number and, if present, a mi

- Strings are displayed in a field equal in w

The default behavior for floating-point typ
ences between ancient and current C++ impl

- **New style**—Floating-point types are d
trailing zeros aren't displayed. (Note tha
nection with the precision to which the
in fixed-point notation or else in E nota
depending on the value of the number.
nent is 6 or larger or -5 or smaller. Agai
number and, if present, a minus sign. Th
standard C library function `fprintf()`

- **Old style**—Floating-point types are dis
decimal, except that trailing zeros aren't
displayed has no connection with the pr
number is displayed in fixed-point nota
depending on the value of the number.
the number and, if present, a minus sign

Because each value is displayed in a width
between values explicitly; otherwise, consecut

```
    char ch = 'K';
    int t = 273;
    cout << ch << ":\n";
    cout << t << ":\n";
    cout << -t <<":\n";

    double f1 = 1.200;
    cout << f1 << ":\n";
    cout << (f1 + 1.0 / 9.0) << ":\n";

    double f2 = 1.67E2;
    cout << f2 << ":\n";
    f2 += 1.0 / 9.0;
    cout << f2 << ":\n";
    cout << (f2 * 1.0e4) << ":\n";


    double f3 = 2.3e-4;
    cout << f3 << ":\n";
    cout << f3 / 10 << ":\n";

    return 0;
}
```

to display integers. By using ios_base memb
and the number of places displayed to the rig
class is an indirect base class for ostream, you
(or descendants), such as cout.

Note

The members and methods found in the ios_
class. Now ios_base is a base class to ios
with char and wchar_t specializations, and

Let's look at how to set the number base to
whether integers are displayed in base 10, base
oct manipulators. For example, the following
state for the cout object to hexadecimal:

```
hex(cout);
```

After you do this, a program will print inte
the format state to another choice. Note that
hence they don't have to be invoked by an ob

Although the manipulators really are funct

```
cout << hex;
```

```
// set to octal mode
    cout << oct << n << "     " << n * n
// alternative way to call a manipulator
    dec(cout);
    cout << n << "     " << n * n << " (d

    return 0;
}
```

Here is some sample output from the prog

```
Enter an integer: 13
n     n*n
13     169 (decimal)
d     a9 (hexadecimal)
15     251 (octal)
13     169 (decimal)
```

right-justification. After that, the field width r
ters and the 24 are printed in fields equal to t

Caution

The width() method affects only the next ite
default value afterward.

C++ never truncates data, so if you attemp
a width of two, C++ expands the field to fit t
with asterisks if the data doesn't fit. The C/C
more important than keeping the columns ne
17.4 shows how the width() member functio

Listing 17.4 **width.cpp**

```
// width.cpp -- using the width method
#include <iostream>

int main()
{
    using std::cout;
    int w = cout.width(30);
    cout << "default field width = " << w
```

The output displays values right-justified i
spaces. That is, `cout` achieves the full field wi
the spaces are inserted to the left of the value
the *fill character*. Right-justification is the defa

Note that the program in Listing 17.4 app
played by the first `cout` statement but not to
method affects only the next single item disp
because `cout.width(30)` returns the previou
just set. The fact that `w` is `0` means that zero is
expands a field to fit the data, this one size fit
align column headings and data by using a w
and a width of eight characters for the secon

Fill Characters

By default, `cout` fills unused parts of a field w
function to change that. For example, the foll
asterisk:

```
cout.fill('*');
```

That can be handy for, say, printing check
two. Listing 17.5 illustrates using this membe

```
Waldo Whipsnade: $****900
Wilmarie Wooper: $***1350
```

Note that, unlike the field width, the new
change it.

Setting Floating-Point Display Precision

The meaning of floating-point *precision* depen
it means the total number of digits displayed.
cussed soon, *precision* means the number of dig
place. The precision default for C++, as you'v
zeros are dropped.) The `precision()` membe
example, the following statement causes `cout`

```
cout.precision(2);
```

Unlike the case with `width()`, but like the
stays in effect until it is reset. Listing 17.6 dem

Listing 17.6 **precise.cpp**

```
// precise.cpp -- setting the precision
#include <iostream>

int main()
```

fourth line displays a total of two digits.

Printing Trailing Zeros and Decimal Poi

Certain forms of output, such as prices or nu
are retained. For example, the output to Listi
$20.4. The iostream family of classes doesn't
accomplish that. However, the ios_base clas
controls several formatting features. The class
as arguments to this function. For example, th
play trailing decimal points:

```
cout.setf(ios_base::showpoint);
```

In the default floating-point format, it also
instead of displaying 2.00 as 2, cout will disp
is in effect. Listing 17.7 adds this statement to

In case you're wondering about the notati
class-scope static constant that is defined in th
means that you have to use the scope-resolut
you use the name outside a member function
a constant defined in the ios_base class.

Here is the output of the program in Listin

```
"Furry Friends" is $20.4000!
"Fiery Fiends" is $2.78889!
"Furry Friends" is $20.!
"Fiery Fiends" is $2.8!
```

This output shows the trailing zeros for the point but no trailing zeros because the precisi have been displayed.

More About `setf()`

The `setf()` method controls several other fo point is displayed, so let's take a closer look at member in which individual bits (called *flags* i aspects, such as the number base and whether is called *setting the flag* (or bit) and means setti ming equivalent to setting DIP switches to co and `oct` manipulators, for example, adjust the The `setf()` function provides another means

The `setf()` function has two prototypes. T

```
fmtflags setf(fmtflags);
```

Note

A *bitmask type* is a type that is used to store
type, an `enum`, or an STL `bitset` container.
accessible and has its own meaning. The io
state information.

Because these formatting constants are de
the scope-resolution operator with them. Th
not just `uppercase`. If you don't use a `using`
the scope-resolution operator to indicate tha
is, you can use `std::ios_base::showpos`, an
are overridden. Listing 17.8 illustrates using s

Listing 17.8 **setf.cpp**

```
// setf.cpp -- using setf() to control fo
#include <iostream>

int main()
{
    using std::cout;
    using std::endl;
    using std::ios_base;

    int temperature = 63;
```

```
0X3F
How 0X1!  oops -- How true!
```

Note that the plus sign is used only with th
and octal values as unsigned; therefore no sign
implementations may still display a plus sign.)

The second `setf()` prototype takes two ar

```
fmtflags setf(fmtflags , fmtflags );
```

This overloaded form of the function is us
than 1 bit. The first argument, as before, is a `fm`
ting. The second argument is a value that first
suppose setting bit 3 to `1` means base 10, settin
to `1` means base 16. Suppose output is in base
only do you have to set bit 5 to `1`, you also ha
bit. The clever `hex` manipulator does both task
requires a bit more work because you use the
clear and then use the first argument to indica
cated as it sounds because the `ios_base` class
this purpose. In particular, you should use the
ond argument and `ios_base::hex` as the first
the following function call has the same effect

```
cout.setf(ios_base::hex, ios_base::basefie
```

ment. Left-justification means starting a value
cation means ending a value at the right end
placing any signs or base prefixes at the left o
right of the field. (Unfortunately, C++ does

Fixed-point notation means using the 123
the size of the number, and scientific notatio
the size of the number. If you are familiar wi
know that the default C++ mode correspond
the `%f` specifier, and `scientific` corresponds

Under the C++ Standard, both fixed and
properties:

- *Precision* means the number of digits to
  number of digits.
- Trailing zeros are displayed.

The `setf()` function is a member functio
class for the `ostream` class, you can invoke th
example, to request left-justification, you use

```
ios_base::fmtflags old = cout.setf(ios::l
```

To restore the previous setting, you use th

```
cout.setf(old, ios::adjustfield);
```

```
cout.setf(ios_base::showpos);
cout.setf(ios_base::showpoint);
cout.precision(3);
// use e-notation and save old format
ios_base::fmtflags old = cout.setf(ios
    ios_base::floatfield);
cout << "Left Justification:\n";
long n;
for (n = 1; n <= 41; n+= 10)
{
    cout.width(4);
    cout << n << "|";
    cout.width(12);
    cout << sqrt(double(n)) << "|\n";
}

// change to internal justification
cout.setf(ios_base::internal, ios_base
// restore default floating-point disp
cout.setf(old, ios_base::floatfield);

cout << "Internal Justification:\n";
for (n = 1; n <= 41; n+= 10)
{
```

```
Left Justification:
+1  |+1.000e+00  |
+11 |+3.317e+00  |
+21 |+4.583e+00  |
+31 |+5.568e+00  |
+41 |+6.403e+00  |
Internal Justification:
+  1|+      1.00|
+ 11|+      3.32|
+ 21|+      4.58|
+ 31|+      5.57|
+ 41|+      6.40|
Right Justification:
  +1|     +1.000|
 +11|     +3.317|
 +21|     +4.583|
 +31|     +5.568|
 +41|     +6.403|
```

Note how a precision of 3 causes the defa
justification in this program) to display a tota
modes display three digits to the right of the
the exponent for e-notation depends on the

If you knew for certain that cout were in t
ios_base::fixed as an argument to unsetf(
regardless of the current state of cout, so it's a

Standard Manipulators

Using setf() is not the most user-friendly ap
manipulators to invoke setf() for you, auton
You've already seen dec, hex, and oct. These r
to older C++ implementations, work like hex
on left-justification and the fixed decimal poi

```
cout << left << fixed;
```

Table 17.3 lists these along with several oth

Table 17.3 **Some Standard Manipulators**

Manipulator
boolalpha
noboolalpha
showbase
noshowbase

`oct`

`fixed`

`scientific`

Tip

If your system supports these manipulators, have the option of using `setf()`.

The `iomanip` Header File

Setting some format values, such as the field tools. To make life easier, C++ supplies additi file. They provide the same services already d ient manner. The three most commonly used sion, `setfill()` for setting the fill character, Unlike the manipulators discussed previously manipulator takes an integer argument that s ulator takes a `char` argument that indicates th takes an integer argument that specifies the fi

```
    cout << fixed << right;

    // use iomanip manipulators
    cout << setw(6) << "N" << setw(14) <<
         << setw(15) << "fourth root\n";

    double root;
    for (int n = 10; n <=100; n += 10)
    {
        root = sqrt(double(n));
        cout << setw(6) << setfill('.') <<
               << setw(12) << setprecision
               << setw(14) << setprecision
               << endl;
    }

    return 0;
}
```

Here is the output of the program in Listin

```
     N   square root   fourth root
....10         3.162        1.7783
....20         4.472        2.1147
```

Typically, you use `cin` as follows:

```
cin >> value_holder;
```

Here *value_holder* identifies the memor
be the name of a variable, a reference, a deref
or of a class. How `cin` interprets the input de
The `istream` class, defined in the `iostream`
ator to recognize the following basic types:

- `signed char &`
- `unsigned char &`
- `char &`
- `short &`
- `unsigned short &`
- `int &`
- `unsigned int &`
- `long &`
- `unsigned long &`
- `long long &` (C++11)
- `unsigned long long &` (C++11)

`int`. In this case, the compiler matches

```
cin >> staff_size;
```

to the following prototype:

```
istream & operator>>(int &);
```

The function corresponding to that protot
sent to the program—say, the characters `2`, `3`, `1`
the function then converts these characters to
integer `23184`. If, on the other hand, `staff_s`
`operator>>(double &)` to convert the same
sentation of the value `23184.0`.

Incidentally, you can use the `hex`, `oct`, and
integer input is to be interpreted as hexadecim
the following statement causes an input of `12`
decimal `18`, and it causes `ff` or `FF` to be read a

```
cin >> hex;
```

The `istream` class also overloads the `>>` ex

- `signed char *`
- `char *`
- `unsigned char *`

The various versions of the extraction operat
input stream. They skip over white space (bla
a non-white-space character. This is true even
which the argument is type `char`, `unsigned`
C's character input functions (see Figure 17.5
operator reads that character and assigns it to
the operator reads in one unit of the indicate
initial non-white-space character up to the fi
tion type.

For example, consider the following code:

```
int elevation;
cin >> elevation;
```

Suppose you type the following characters

```
-123Z
```

The operator will read the `-`, `1`, `2`, and `3` ch
integer. But the `Z` character isn't valid, so the
`Z` remains in the input stream, and the next `c`
Meanwhile, the operator converts the charact
assigns it to `elevation`.

whether input meets the program requiremen

Listing 17.11 **check_it.cpp**

```
// check_it.cpp -- checking for valid inpu
#include <iostream>

int main()
{
    using namespace std;
    cout << "Enter numbers: ";

    int sum = 0;
    int input;
    while (cin >> input)
    {
        sum += input;
    }

    cout << "Last value entered = " << inp
    cout << "Sum = " << sum << endl;
    return 0;
}
```

non-accessible file or trying to write to a wr
The `badbit` element is set when some undia
stream. (Implementations don't necessarily ag
which set `badbit`.) When all three of these st
grams can check the stream state and use tha
Table 17.4 lists these bits, along with some `io`
stream state.

Table 17.4 **Stream States**

Member	Description
`eofbit`	Is set to `1` if
`badbit`	Is set to `1` if there could
`failbit`	Is set to `1` if characters o expected ch
`goodbit`	Just another
`good()`	Returns `tru` cleared).
`eof()`	Returns `tru`

state, but they do so in a different fashion. The
ment. Thus, the following call uses the default
bits (eofbit, badbit, and failbit):

```
clear();
```

Similarly, the following call makes the state
the other two state bits are cleared:

```
clear(eofbit);
```

The setstate() method, however, affects
Thus, the following call sets eofbit without a

```
setstate(eofbit);
```

So if failbit was already set, it stays set.

Why would you reset the stream state? For
is to use clear() with no argument to reopen
input or end-of-file; whether doing so makes
ing to accomplish. You'll see some examples sh
to provide a means for input and output funct
is an int, the following call can result in oper
failbit or eofbit:

```
cin >> num;  // read an int
```

```
cin.exceptions(badbit | eofbit);
```

Listing 17.12 modifies Listing 17.11 so tha
tion if `failbit` is set.

Listing 17.12 **cinexcp.cpp**

```
// cinexcp.cpp -- having cin throw an exc
#include <iostream>
#include <exception>

int main()
{
    using namespace std;
    // have failbit cause an exception to
    cin.exceptions(ios_base::failbit);
    cout << "Enter numbers: ";
    int sum = 0;
    int input;
    try {
        while (cin >> input)
        {
            sum += input;
        }
    } catch(ios_base::failure & bf)
```

tion for badbit because that circumstance wo
designed to read numbers from a data file up t
an exception for failbit because that would

Stream State Effects

An if or while test such as the following test
bits cleared):

```
while (cin >> input)
```

If a test fails, you can use the member func
possible causes. For example, you could modif
like this:

```
while (cin >> input)
{
    sum += input;
}
if (cin.eof())
    cout << "Loop terminated because EOF e
```

Setting a stream state bit has a very importa
ther input or output until the bit is cleared. Fo

```
while (cin >> input)
{
```

way is to keep reading characters until reachi
Chapter 6, "Branching Statements and Logica
returns `true` if its argument is a white-space
line instead of just the next word:

```
while (cin.get() != '\n')
    continue;  // get rid rest of line
```

This example assumes that the loop termi
pose, instead, that the loop terminated becau
failure. Then the new code disposing of bad i
using the `fail()` method to test whether the
cal reasons, `fail()` returns `true` if either fai
exclude the latter case. The following code sh

```
while (cin >> input)
{
    sum += input;
}
cout << "Last value entered = " << input
cout << "Sum = " << sum << endl;
if (cin.fail() && !cin.eof() ) // failed
{
```

as it is, without skipping over white space and

Let's look at these two groups of istream

Single-Character Input

When used with a char argument or no argu next input character, even if it is a space, tab, c version assigns the input character to its argum input character, converted to an integer type (

The get(char &) Member Function

Let's try get(char &) first. Suppose you have

```
int ct = 0;
char ch;
cin.get(ch);
while (ch != '\n')
{
    cout << ch;
    ct++;
    cin.get(ch);
}
cout << ct << endl;
```

responding output to this:

```
IC++clearly.
```

Worse, the loop would never terminate! B
lines, the code would never assign the newlin
would never terminate the loop.

The `get(char &)` member function retur
invoke it. This means you can concatenate ot

```
char c1, c2, c3;
cin.get(c1).get(c2) >> c3;
```

First, `cin.get(c1)` assigns the first input c
object, which is `cin`. This reduces the code to
ond input character to `c2`. The function call r
This, in turn, assigns the next non-white-spac
wind up being assigned white space, but `c3` c

If `cin.get(char &)` encounters the end o
board (Ctrl+Z for DOS and Windows comm
of a line for Unix), it does not assign a value
the program has reached the end of the file, t
the method calls `setstate(failbit)`, which

```
cout << c1 << endl;
```

The `get(void)` member function returns t
depending on the character set and locale). Th

```
char c1, c2, c3;
cin.get().get() >> c3;  // not valid
```

Here `cin.get()` returns a type `int` value. I
object, you can't apply the membership opera
ever, you can use `get()` at the end of an extra

```
char c1;
cin.get(c1).get();   // valid
```

The fact that `get(void)` returns type `int` i
operator. But because `cin.get(c1)` returns ci
particular code would read the first input char
ond input character and discard it.

Upon reaching the end-of-file, real or simu
which is a symbolic constant provided by the
allows the following construction for reading

```
int ch;
while ((ch = cin.get()) != EOF)
{
```

space is convenient for offering menu choice

```
cout  << "a. annoy client         b. bill
      << "c. calm client          d. dece
      << "q.\n";
cout  << "Enter a, b, c, d, or q: ";
char ch;
cin >> ch;
while (ch != 'q')
{
    switch(ch)
    {
        ...
    }
    cout << "Enter a, b, c, d, or q: ";
    cin >> ch;
}
```

To enter, say, a `b` response, you type **b** and p
response `b\n`. If you used either form of `get(`
`\n` character on each loop cycle, but the extra
programmed in C, you've probably encounter
to the program as an invalid response. It's an e

input as a string.) The third argument specifie
The versions with just two arguments use the
tion reads up to the maximum characters or u
whichever comes first.

For example, the following code reads char

```
char line[50];
cin.get(line, 50);
```

The cin.get() function quits reading inpu
acters or, by default, after encountering a new
chief difference between get() and getline(
in the input stream, making it the first charact
getline() extracts and discards the newline c

Chapter 4 illustrated using the two-argume
Now let's look at the three-argument versions
ter. Encountering the delimiter character caus
number of characters hasn't been reached. So
if they reach the end of a line before reading t
the default case, get() leaves the delimiter ch
does not.

Listing 17.13 demonstrates how getline(
ignore() member function. ignore() takes t
mum number of characters to read and a char

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

    char input[Limit];

    cout << "Enter a string for getline()
    cin.getline(input, Limit, '#');
    cout << "Here is your input:\n";
    cout << input << "\nDone with phase 1

    char ch;
    cin.get(ch);
    cout << "The next input character is

    if (ch != '\n')
        cin.ignore(Limit, '\n');    // di

    cout << "Enter a string for get() pro
    cin.get(input, Limit, '#');
    cout << "Here is your input:\n";
```

```
want my
Done with phase 2
The next input character is #
```

Note that the getline() function discards

and the get() function does not.

Unexpected String Input

Some forms of input for get(char *, int) a

the other input functions, encountering end-o

the stream, such as device failure, sets badbit.

input that meets or exceeds the maximum nu

call. Let's look at those cases now.

If either method fails to extract any charac

the input string and uses setstate() to set f

extract any characters? One possibility is if an

of-file. For get(char *, int), another possib

```
char temp[80];
while (cin.get(temp,80))  // terminates on
      ...
```

too many input characters caused the metho
next section) to examine the next input char
read the entire line. If it's not a newline, then
the end. This technique doesn't necessarily w
reads and discards the newline, so looking at
But if you use `get()`, you have the option of
read. The next section includes an example o
marizes these behaviors.

Table 17.6 **Input Behavior**

Method	Behavior
`getline(char *, int)`	Sets `failbit` as a character Sets `failbit` and more are s
`get(char *, int)`	Sets `failbit`

Other `istream` Methods

Other `istream` methods besides the ones dis
`gcount()`, and `putback()`. The `read()` funct
them in the specified location. For example,
from the standard input and place them in th

The call to `cin.peek()` peeks at the next i
Then the `while` loop test condition checks th
this is the case, the loop reads the character in
When the loop terminates, the period or new
positioned to be the first character read by the
appends a null character to the array, making i

The `gcount()` method returns the number
extraction method. That means characters read
`read()` method but not by the extraction ope
ular data types. For example, suppose you've ju
line into the `myarray` array and you want to k
could use the `strlen()` function to count the
quicker to use `cin.gcount()` to report how m
input stream.

The `putback()` function inserts a characte
acter then becomes the first character read by
method takes one `char` argument, which is th
type `istream &`, which allows the call to be c
Using `peek()` is like using `get()` to read a ch
the character back in the input stream. Howev
ting back a character that is different from the

```
        else
        {
            cin.putback(ch);    // reinse
            break;
        }
    }

    if (!cin.eof())
    {
        cin.get(ch);
        cout << endl << ch << " is next i
    }
    else
    {
        cout << "End of file reached.\n";
        std::exit(0);
    }

    while(cin.peek() != '#')    // look a
    {
        cin.get(ch);
        cout << ch;
    }
    if (!cin.eof())
```

```
while(cin.get(ch))            // terminate
{
    if (ch != '#')
        cout << ch;
    else
    {
        cin.putback(ch);  // reinsert char
        break;
    }
}
```

The expression `cin.get(ch)` returns `fals` simulating end-of-file from the keyboard term first, the program puts the character back in th terminate the loop.

The second approach is simpler in appeara

```
while(cin.peek() != '#')     // look ahead
{
    cin.get(ch);
    cout << ch;
}
```

```
    if (cin.peek() != '\n')
        cout << "Sorry, we only have enou
                << name << endl;
    eatline();
    cout << "Dear " << name << ", enter y
    cin.get(title,SLEN);
    if (cin.peek() != '\n')
        cout << "We were forced to trunca
    eatline();
    cout << " Name: " << name
         << "\nTitle: " << title << endl;

    return 0;

}
```

Here is a sample run of the program in Li

```
Enter your name: Ella Fishsniffer
Sorry, we only have enough room for Ella
Dear Ella Fish, enter your title:
Executive Adjunct
We were forced to truncate your title.
```

and so on. Unless you're programming on the
have to worry about those things. What you d
file, a way to have a program read the content
ate and write to files. Redirection (as discusse
file support, but it is more limited than explic
rection comes from the operating system, not
tems. This book has already touched on file I/
thoroughly.

The C++ I/O class package handles file in
input and output. To write to a file, you create
methods, such as the `<<` insertion operator or
`ifstream` object and use the `istream` metho
`get()`. Files require more management than t
example, you have to associate a newly opene
read-only mode, write-only mode, or read-an
might want to create a new file, replace an ol
want to move back and forth through a file. T
eral new classes in the `fstream` (formerly `fst`
class for file input and the `ofstream` class for
class for simultaneous file I/O. These classes a
header file, so objects of these new classes are
learned.

```
ofstream fout("jar.txt");  // create fout
```

When you've gotten this far, you use `fout` manner as `cout`. For example, if you want to can do the following:

```
fout << "Dull Data";
```

Indeed, because `ostream` is a base class for `ostream` methods, including the various inse methods and manipulators. The `ofstream` cla cates space for an output buffer when it creat create two `ofstream` objects, the program cr `ofstream` object such as `fout` collects outpu the buffer is filled, it transfers the buffer cont disk drives are designed to transfer data in lar approach greatly speeds up the transfer rate o

Opening a file for output this way creates file by that name exists prior to opening it fo that output starts with a clean file. Later in th file and retain its contents.

```
char ch;
fin >> ch;              // read a charact
char buf[80];
fin >> buf;             // read a word fr
fin.getline(buf, 80);   // read a line fr
string line;
getline(fin, line);     // read from a fi
```

Input, like output, is buffered, so creating a
input buffer, which the `fin` object manages. A
faster than byte-by-byte transfer.

The connections with a file are closed auto
objects expire—for example, when the progra
tion with a file explicitly by using the `close(`

```
fout.close();    // close output connectio
fin.close();     // close input connection
```

Closing such a connection does not elimin
file. However, the stream management apparat
object still exists, along with the input buffer i
nect the stream to the same file or to another

Let's look at a short example. The program
ates a file that has that name, writes some info
file flushes the buffer, guaranteeing that the fil

```
    fout << "For your eyes only!\n";
    cout << "Enter your secret number: ";
    float secret;
    cin >> secret;
    fout << "Your secret number is " << s
    fout.close();           // close file

// create input stream object for new fil
    ifstream fin(filename.c_str());
    cout << "Here are the contents of " <
    char ch;
    while (fin.get(ch))     // read chara
        cout << ch;         // write it t
    cout << "Done\n";
    fin.close();

    return 0;
}
```

Here is a sample run of the program in Li

```
Enter name for new file: pythag
Enter your secret number: 3.14159
```

```
if (fin.fail())  // open attempt failed
{
   ...
}
```

Or because an `ifstream` object, like an `is`
where a `bool` type is expected, you could use

```
fin.open(argv[file]);
if (!fin)  // open attempt failed
{
   ...
}
```

However, newer C++ implementations ha
been opened—the `is_open()` method:

```
if (!fin.is_open())  // open attempt faile
{
   ...
}
```

The reason this is better is that it tests for s
miss, as discussed in the following Caution.

want to count how many times a name appea
open a single stream and associate it with eac
resources more effectively than opening a sep
approach, you declare an `ifstream` object wi
method to associate the stream with a file. Fo
reading two files in succession:

```
ifstream fin;           // create stream
fin.open("fat.txt");    // associate stre
...                     // do stuff
fin.close();            // terminate asso
fin.clear();            // reset fin (may
fin.open("rat.txt");    // associate stre
...
fin.close();
```

We'll look at an example shortly, but first,
files to a program in a manner that allows the

Command-Line Processing

File-processing programs often use command
line arguments are arguments that appear on th

so on. The following loop would print each c

```
for (int i = 1; i < argc; i++)
      cout << argv[i] << endl;
```

Starting with `i = 1` just prints the comma would print the command name as well.

Command-line arguments, of course, go ha systems such as the Windows command prom still allow you to use command-line argumen

- Many Windows IDEs (Integrated Devel providing command-line arguments. Ty of menu choices that lead to a box into arguments. The exact set of steps varies upgrade, so check your documentation.
- Many Windows IDEs can produce exec command prompt mode.

Listing 17.17 combines the command-line count characters in files listed on the comma

```
        if (!fin.is_open())
        {
            cerr << "Could not open " <<
            fin.clear();
            continue;
        }
        count = 0;
        while (fin.get(ch))
            count++;
        cout << count << " characters in
        total += count;
        fin.clear();            // needed
        fin.close();            // disconn
    }
    cout << total << " characters in all

    return 0;
}
```

Note

Some C++ implementations require using fi
ers do not. It depends on whether associatin
cally resets the stream state. In does no har

filename or by using the open() method, you
fies the file mode:

```
ifstream fin("banjo", mode1);  // construc
ofstream fout();
fout.open("harp", mode2);      // open() w
```

The ios_base class defines an openmode ty
and iostate types, it is a bitmask type. (In th
from several constants defined in the ios_bas
the constants and their meanings. C++ file I/
compatible with ANSI C file I/0.

Table 17.7 **File Mode Constants**

Constant	Meaning
ios_base::in	Open file for reading.
ios_base::out	Open file for writing.
ios_base::ate	Seek to end-of-file upon
ios_base::app	Append to end-of-file.
ios_base::trunc	Truncate file if it exists.
ios_base::binary	Binary file.

You can expect to find some differences a
example, some allow you to omit the ios_ba
don't. If you aren't using the default mode, th
elements explicitly. Some compilers don't sup
may offer choices beyond those in the table.
you may have to make some alterations in th
system. The good news is that the developme
uniformity.

Standard C++ defines parts of file I/O in
C++ statement like

```
ifstream fin(filename, c++mode);
```

is implemented as if it uses the C `fopen()` fu

```
fopen(filename, cmode);
```

Here *c++mode* is a type `openmode` value, su
sponding C-mode string, such as `"r"`. Table 1
C++ modes and C modes. Note that ios_ba
it doesn't cause truncation when combined w
such as `ios_base::in | ios_base::trunc`,
`is_open()` method detects this failure.

C++ mode	C mode
`ios_base::in`	`"r"`
`ios_base::out`	`"w"`
`ios_base::out | ios_base::trunc`	`"w"`
`ios_base::out | ios_base::app`	`"a"`
`ios_base::in | ios_base::out`	`"r+"`
`ios_base::in | ios_base::out | ios_base::trunc`	`"w+"`
c++mode | `ios_base::binary`	"*cmodeb*"
c++mode | `ios_base::ate`	"*cmode*"

older compilers don't quite conform to the cu
modes such as `nocreate` that are not part o
ers require the `fin.clear()` call before ope

Listing 17.18 **append.cpp**

```
// append.cpp -- appending information to
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>      // (for exit()

const char * file = "guests.txt";
int main()
{
    using namespace std;
    char ch;
```

```
    while (getline(cin,name) && name.size(
    {
          fout << name << endl;
    }
    fout.close();

// show revised file
    fin.clear();    // not necessary for s
    fin.open(file);
    if (fin.is_open())
    {
        cout << "Here are the new contents
             << file << " file:\n";
        while (fin.get(ch))
            cout << ch;
        fin.close();
   }
    cout << "Done.\n";
    return 0;

}
```

```
Greta Greppo
LaDonna Mobile
Fannie Mae

Here are the new contents of the guests.t
Ghengis Kant
Hank Attila
Charles Bigg
Greta Greppo
LaDonna Mobile
Fannie Mae
Done.
```

You should be able to read the contents o
the editor you use to write your source code

Binary Files

When you store data in a file, you can store t
form means you store everything as text, ever
2.324216e+07 in text form means storing th
That requires converting the computer's inter
ber to character form, and that's exactly what
mat, on the other hand, means storing the co

Each format has advantages. The text form
ordinary editor or word processor to read and
file from one computer system to another. Th
bers because it stores the exact internal repres
errors or round-off errors. Saving data in bina
conversion and because you may be able to sa
mat usually takes less space, depending on the
system can be a problem, however, if the new
tion for values. Even different compilers on th
representations for structure layouts. In these c
program to translate one data format to anoth

Let's look at a more concrete example. Co
declaration:

```
const int LIM = 20;
struct planet
{
    char name[LIM];      // name of plane
    double population;   // its population
    double g;            // its accelerati
};
planet pl;
```

to save data in binary form, you should use t
using the `ios_base::binary` constant in the
should do this on a Windows system, check t
"Binary Files and Text Files."

Binary Files and Text Files

Using a binary file mode causes a program t
versa, without any hidden translation taking
default text mode. For example, consider Wi
two-character combination: carriage return, l
with a carriage return. Unix and Linux files re
grew up on Unix, also represents a newline w
program automatically translates the C++ ne
writing to a text mode file; and a Macintosh
riage return when writing to a file. When rea
newline back to the C++ form. The text form
a byte in the middle of a `double` value could
for the newline character. Also there are diff
should use the binary file mode when saving
one file mode, so on them the binary mode i

To save data in binary form instead of tex
function. This method, recall, copies a specifi
This chapter used it earlier to copy text, but

gramming Exercise 6.)

Tip

The `read()` and `write()` member functions
recover data that has been written to a file wi

Listing 17.19 uses these methods to create
is similar to Listing 17.18, but it uses `write()`
tor and the `get()` method. It also uses manipu

Note

Although the binary file concept is part of AN
not provide support for the binary file mode.
systems have only one file type in the first p
as `read()` and `write()` with the standard f
rejects `ios_base::binary` as a valid const
your implementation doesn't support the `fi`
`cout.setf(ios_base::fixed, ios_base`
`cout.setf(ios_base::right, ios_base`
stitute `ios` for `ios_base`. Other compilers, p
idiosyncrasies.

```
// show initial contents
    ifstream fin;
    fin.open(file, ios_base::in |ios_base
    //NOTE: some systems don't accept the
    if (fin.is_open())
    {
    cout << "Here are the current content
        << file << " file:\n";
    while (fin.read((char *) &pl, sizeof
    {
        cout << setw(20) << pl.name << ":
              << setprecision(0) << setw(
              << setprecision(2) << setw(
    }
    fin.close();
    }

// add new data
    ofstream fout(file,
             ios_base::out | ios_base::ap
    //NOTE: some systems don't accept the
    if (!fout.is_open())
    {
```

```
    fin.open(file, ios_base::in | ios_base
    if (fin.is_open())
    {
        cout << "Here are the new contents
             << file << " file:\n";
        while (fin.read((char *) &pl, size
        {
            cout << setw(20) << pl.name <<
                 << setprecision(0) << set
                 << setprecision(2) << set
        }
        fin.close();
    }
    cout << "Done.\n";
    return 0;
}
```

Here is a sample initial run of the program

```
Enter planet name (enter a blank line to q
Earth
Enter planetary population: 6928198253
Enter planet's acceleration of gravity: 9.
```

```
while (std::cin.get() != '\n') continue;
```

This reads and discards input up through t
input statement in the loop:

```
cin.get(pl.name, 20);
```

If the newline were left in place, this stater
line, terminating the loop.

You might wonder if this program could u
array for the `name` member of the `planet` str
out major changes in design. The problem is
the string within itself; instead, it contains a p
string is stored. So if you copy the structure t
just copy the address of where the string was
that address is meaningless.

Random Access

For our last file example, let's look at random
to any location in the file instead of moving t
approach is often used with database files. A p
giving the location of data in the main data f
read the data there, and perhaps modify it. Th

of actions indefinitely, but this version perform
approach allows you to examine several aspect
down in matters of program design.

> **Caution**
>
> This program assumes that the `planets.da`
> `binary.cpp` program in Listing 17.19.

The first question to answer is what file m
need the `ios_base::in` mode. For binary I/
(Again, on some nonstandard systems you can
mode.) In order to write to the file, you need
mode. However, the append mode allows a pr
only. The rest of the file is read-only; that is, y
it—so, to be able to modify the data, you have
cates, using the `in` and `out` modes simultaneo
have to add the binary element. As mentioned
modes. Thus, you need the following statemen

```
finout.open(file,ios_base::in | ios_base::
```

Next, you need a way to move through a
for this: `seekg()` moves the input pointer to
output pointer to a given file location. (Actua

class was replaced with the basic_istrean
were replaced with the template-based types
streampos and streamoff continue to exis
off_type. Similarly, you can use the wstre
seekg() with a wistream object.

Let's take a look at the arguments to the fi
streamoff type are used to measure offsets, i
The streamoff argument represents the file
one of three locations. (The type may be defi
seek_dir argument is another integer type th
ues, in the ios_base class. The constant ios_
the beginning of the file. The constant ios_b
current position. The constant ios_base::en
the file. Here are some sample calls, assuming

```
fin.seekg(30, ios_base::beg);    // 30 by
fin.seekg(-1, ios_base::cur);    // back
fin.seekg(0, ios_base::end);     // go to
```

Now let's look at the second prototype. Va
in a file. It can be a class, but, if so, the class in
ment and a constructor with an integer argum
to streampos values. A streampos value repr

```
    {
        finout.seekg(0);    // go to beginning
        cout << "Here are the current contents
              << file << " file:\n";
        while (finout.read((char *) &pl, sizeo
        {
            cout << ct++ << ": " << setw(LIM)
            << setprecision(0) << setw(12) <<
            << setprecision(2) << setw(6) << p
        }
        if (finout.eof())
            finout.clear(); // clear eof flag
        else
        {
            cerr << "Error in reading " << fil
            exit(EXIT_FAILURE);
        }
    }
    else
    {
        cerr << file << " could not be opened
        exit(EXIT_FAILURE);
    }
```

program asks the user to enter a record numb
of bytes in a record yields the byte number fo
the record number, the desired byte number

```
cout << "Enter the record number you wish
long rec;
cin >> rec;
eatline();              // get rid of new
if (rec < 0 || rec >= ct)
{
    cerr << "Invalid record number -- bye
    exit(EXIT_FAILURE);
}
streampos place = rec * sizeof pl;  // co
finout.seekg(place);    // random access
```

The variable `ct` represents the number of
beyond the limits of the file.

Next, the program displays the current rec

```
finout.read((char *) &pl, sizeof pl);
cout << "Your selection:\n";
cout << rec << ": " << setw(LIM) << pl.na
<< setprecision(0) << setw(12) << pl.popu
<< setprecision(2) << setw(6) << pl.g <<
if (finout.eof())
    finout.clear();     // clear eof flag
```

Note

The older the implementation, the more likely
systems don't recognize the binary flag, the f

Listing 17.20 **random.cpp**

```
// random.cpp -- random access to a binary
#include <iostream>     // not required by
#include <fstream>
#include <iomanip>
#include <cstdlib>      // for exit()
const int LIM = 20;
struct planet
{
    char name[LIM];     // name of planet
    double population;  // its population
    double g;           // its acceleratio
};

const char * file = "planets.dat";  // ASS
inline void eatline() { while (std::cin.ge

int main()
{
```

```
            else
            {
                cerr << "Error in reading " <
                exit(EXIT_FAILURE);
            }
        }
        else
        {
            cerr << file << " could not be op
            exit(EXIT_FAILURE);
        }

    // change a record
        cout << "Enter the record number you
        long rec;
        cin >> rec;
        eatline();              // get rid of
        if (rec < 0 || rec >= ct)
        {
            cerr << "Invalid record number --
            exit(EXIT_FAILURE);
        }
        streampos place = rec * sizeof pl;  /
        finout.seekg(place);    // random acc
```

```
    if (finout.fail())
    {
        cerr << "Error on attempted write\
        exit(EXIT_FAILURE);
    }

// show revised file
    ct = 0;
    finout.seekg(0);              // go to b
    cout << "Here are the new contents of
         << " file:\n";
    while (finout.read((char *) &pl, sizeo
    {
        cout << ct++ << ": " << setw(LIM)
             << setprecision(0) << setw(12
             << setprecision(2) << setw(6)
    }
    finout.close();
    cout << "Done.\n";
    return 0;
}
```

```
5:           Taanagoot:    361000004 10.
6:               Marin:       252409  9.
Done.
```

By using the techniques in this program, y
new material and delete records. If you were
idea to reorganize it by using classes and func
`planet` structure to a class definition; then yo
so that `cout << pl` displays the class data me
example doesn't bother to verify input, so yo
where appropriate.

Working with Temporary Files

Developing applications often requires the u
sient and must be controlled by the program
this in C++? It's really quite easy to create a
file, and delete the file. First of all, you need
temporary file(s). But wait...how can you ens
The `tmpnam()` standard function declared in

```
char* tmpnam( char* pszName );
```

The `tmpnam()` function creates a temporary
pointed to by `pszName`. The constants `L_tm`

More generally, by using tmpnam(), you can ...
up to L_tmpnam characters per name. The na...
can run this program to see what names you...

Incore Formatting

The iostream family supports I/O between a... ily uses the same interface to provide I/O bet... also provides an sstream family, which uses th... program and a string object. That is, you can... with cout to write formatted information int... methods such as getline() to read informati... reading formatted information from a string... to a string object is termed *incore* formatting... (The sstream family of string support super... support.)

The sstream header file defines an ostrin... ostream class. (There is also a wostringstrea... ter sets.) If you create an ostringstream obje... stores. You can use the same methods with an ... cout. That is, you can do something like the f...

```
ostringstream outstr;
double price = 380.0;
```

```
    string hdisk;
    cout << "What's the name of your hard
    getline(cin, hdisk);
    int cap;
    cout << "What's its capacity in GB? "
    cin >> cap;
    // write formatted information to str
    outstr << "The hard disk " << hdisk <
           << cap << " gigabytes.\n";
    string result = outstr.str();   // sa
    cout << result;                 // sh

    return 0;

}
```

Here's a sample run of the program in Lis

```
What's the name of your hard disk? Datara
What's its capacity in GB? 2000
The hard disk Datarapture has a capacity
```

The `istringstream` class lets you use the from an `istringstream` object, which can b

```
    string lit = "It was a dark and stormy
                 " the full moon glowed br
    istringstream instr(lit);   // use buf
    string word;
    while (instr >> word)       // read a
        cout << word << endl;
    return 0;

}
```

Here is the output of the program in Listin

```
It
was
a
dark
and
stormy
day,
and
the
full
moon
glowed
brilliantly.
```

The I/O class library provides a variety of
versions of the extraction operator (>>) that r
convert character input to those types. The ge
method provide further support for single-ch
the ostream class defines versions of the inser
basic C++ types and that convert them to su
provides further support for single-character
provide similar support for wide characters.

You can control how a program formats o
by using manipulators (functions that can be
iostream and iomanip files. These methods a
base, the field width, the number of decimal p
floating-point values, and other elements.

The fstream file provides class definitions
I/O. The ifstream class derives from the ist
object with a file, you can use all the istrea
ciating an ofstream object with a file lets yo
And associating an fstream object with a file
methods with the file.

To associate a file with a stream, you can p
stream object or you can first create a file stre
to associate the stream with a file. The close

1. What role does the `iostream` file play i

2. Why does typing a number such as 121
   version?

3. What's the difference between the stand

4. Why is `cout` able to display various C+
   instructions for each type?

5. What feature of the output method def

6. Write a program that requests an intege
   hexadecimal forms. Display each form o
   ters wide, and use the C++ number bas

7. Write a program that requests the follow
   shown:

```
Enter your name: Billy Gruff
Enter your hourly wages: 12
Enter number of hours worked: 7.5
First format:
                   Billy Gruff: $
Second format:
Billy Gruff                  : $12.
```

```
        cin.get(ch);
    }
    cout << "ct1 = " << ct1 << "; ct
    
    return 0;
}
```

What does it print, given the following

```
I see a q<Enter>
I see a q<Enter>
```

Here `<Enter>` signifies pressing the En

9. Both of the following statements read a end of a line. In what way does the beh

```
while (cin.get() != '\n')
    continue;
cin.ignore(80, '\n');
```

The resulting file would have these con

```
eggs kites donuts zero lassitude
balloons hammers finance drama
stones
```

5. Mat and Pat want to invite their friends
ming Exercise 8 in Chapter 16, except r
They have asked you to write a progran
   - Reads a list of Mat's friends' names
one friend per line. The names are
sorted order.
   - Reads a list of Pat's friends' names
one friend per line. The names are
sorted order.
   - Merges the two lists, eliminating
`matnpat.dat`, one friend per line

ate data from the user:

```
pc[i]->setall();  // invokes functio
```

To save the data to a file, devise a virtua

```
for (i = 0; i < index; i++)
    pc[i]->writeall(fout);// fout of
```

Note

Use text I/O, not binary I/O, for Programming
include pointers to tables of pointers to virtu
tion to a file. An object filled by using `read(`
pointers, which really messes up the behavio
to separate each data field from the next; th
you could still use binary I/O, but not write o
class methods that apply the `write()` and `r`
ually rather than to the object as a whole. Th
intended data to a file.

```
int main()
{
    using namespace std;
    vector<string> vostr;
    string temp;

// acquire strings
    cout << "Enter strings (empty li
    while (getline(cin,temp) && temp
        vostr.push_back(temp);
    cout << "Here is your input.\n";
    for_each(vostr.begin(), vostr.en

// store in a file
    ofstream fout("strings.dat", ios
    for_each(vostr.begin(), vostr.en
    fout.close();

// recover file contents
    vector<string> vistr;
    ifstream fin("strings.dat", ios_
    if (!fin.is_open())
    {
        cerr << "Could not open file
```

The `data()` member returns a p
the string. It's similar to the `c_st`
null character.

- Write a `GetStrs()` function tha
use `read()` to obtain the size of
many characters from the file, ap
rary string. Because a string's dat
get data into the string rather tha

- The function wrapper template
- Variadic templates

This chapter concentrates on the new C+
book already has covered several C++11 feat
Then we'll look at some additional features i
the C++11 additions that are beyond the sco
draft is over 80% longer than C++98, we wo
examine the BOOST library.

C++11 Features Revisite

By now you may have lost track of the many
tered. This section reviews them briefly.

New Types

C++11 adds the `long long` and `unsigned l`
(or wider) and the `char16_t` and `char32_t` t
representations, respectively. It also adds the "
discusses these additions.

```
Stump s1(3,15.6);      // old style
Stump s2{5, 43.4};     // C++11
Stump s3 = {4, 32.1};  // C++11
```

However, if a class has a constructor whose
template, then only that constructor can use t
list-initialization were discussed in Chapters 3

Narrowing

The initialization-list syntax provides protecti
assigning a numeric value to a numeric type n
initialization allows you to do things that may

```
char c1 = 1.57e27;   // double-to-char, un
char c2 = 459585821; // int-to-char, undef
```

If you use initialization-list syntax, howeve
that attempt to store values in a type "narrowe

```
char c1 {1.57e27};    // double-to-char, c
char c2 = {459585821};// int-to-char,out c
```

However, conversions to wider types are al
is allowed if the value is within the range allo

```
...
}
double sum(std::initializer_list<double>
{
    double tot = 0;
    for (auto p = il.begin(); p !=il.end(
        tot += *p;
    return tot;
}
```

Declarations

C++11 implements several features that simp
arising when templates are used.

auto

C++11 strips the keyword `auto` of its forme
(Chapter 9, "Memory Models and Namespac
ment automatic type deduction, provided th
sets the type of the variable to the type of th

```
auto maton = 112;  // maton is type int
auto pt = &maton;  // pt is type int *
double fm(double, int);
auto pf = fm;      // pf is type double (
```

```
template<typename T, typename U)
void ef(T t, U u)
{
    decltype(T*U) tu;
    ...
}
```

Here, `tu` is of whatever type results from th
defined. For example, if `T` is `char` and `U` is `sho`
automatic integer promotions that take place

The workings of `decltype` are more comp
types can be references and can be `const`-qua
Here are some more examples:

```
int j = 3;
int &k = j
const int &n = j;
decltype(n) i1;      // i1 type const int
decltype(j) i2;      // i2 type int
decltype((j)) i3;    // i3 type int &
decltype(k + 1) i4;  // i4 type int
```

See Chapter 8, "Adventures in Functions," had `typedef` for that purpose:

```
typedef std::vector<std::string>::iterato
```

C++11 provides a second syntax (discusse
for creating aliases:

```
using itType = std::vector<std::string>::
```

The difference is that the new syntax also
tions, whereas `typedef` can't:

```
template<typename T>
  using arr12 = std::array<T,12>;  // tem
```

This statement specializes the `array<T,i`
`12`. For instance, consider these declarations:

```
std::array<double, 12> a1;
std::array<std::string, 12> a2;
```

They can be replaced with the following:

```
arr12<double> a1;
arr12(std::string> a2;
```

unique_ptr, shared_ptr, and weak_ptr. Cha

All the new smart pointers have been desig
move semantics.

Exception Specification Changes

C++ provides a syntax for specifying what ex
to Chapter 15, "Friends, Exceptions, and Mor

```
void f501(int) throw(bad_dog);    // can t
void f733(long long) throw();     // doesn
```

As with auto_ptr, the collective experienc
that, in practice, exception specifications didn
C++11 standard deprecates exception specifi
felt that there is some value in documenting t
and it added the keyword noexcept for this p

```
void f875(short, short) noexcept; // doesn
```

Scoped Enumerations

Traditional C++ enumerations provide a way
come with a rather low level of type checking
the scope that encloses the enumeration defin

conversions for classes could be established. C
ming experience accumulated was that autom
lems in the form of unexpected conversions.
problem by introducing the keyword `explic`
invoked by one-argument constructors:

```
class Plebe
{
    Plebe(int);   // automatic int-to-ple
    explicit Plebe(double);  // requires
    ...
};
...
Plebe a, b;
a = 5;           // implicit conversion,
b = 0.5;         // not allowed
b = Plebe(0.5);  // explicit conversion
```

C++11 extends the use of `explicit` (disc
Classes") so that conversion functions can be

```
class Plebe
{
...
// conversion functions
```

```
    Session(int n, double d, short s) : me
...
};
```

You can use the equal sign or the brace fo
sized version. The result is the same as if you p
member initialization list entries for mem1 and

```
Session() : mem1(10), mem2(1966.54) {}
Session(short s) : mem1(10), mem2(1966.54)
```

Note how using in-class initialization avoi
tors, thus reducing work and the number of c
programmer.

These default values are overridden by a c
ber initialization list, so the third constructor

Template and STL Changes

C++11 makes several changes extending the
Standard Template Library in particular. Some
of use. In this chapter we've already mentione
smart pointers.

```
x = std::rand();
```

New STL Containers

C++11 adds forward_list, unordered_map
unordered_multiset to its collection of ST
forward_list container is a singly linked lis
it's simpler and more economical of space th
other four containers support implementing

C++11 also adds the array template (dis
Chapter 16), for which one specifies an elem

```
std::array<int,360> ar;  // array of 360
```

This template class does not satisfy all the
because the size is fixed, you can't use any m
size of a container. But array does have the
you to use many of the range-based STL alg

New STL Methods

C++11 adds cbegin() and cend() to the lis
the new methods return iterators to the first
container, thus specifying a range encompass

To avoid confusion with the >> operator, C+
nested template declarations:

```
std::vector<std::list<int> > vl; // >> not
```

C++11 removes that requirement:

```
std::vector<std::list<int>> vl; //  >> ok
```

The rvalue Reference

The traditional C++ reference, now called an lvalue. An lvalue is an expression, such as a var represents data for which the program can ob that could appear on the left side of an assignm modifier allowed for constructs that cannot be

```
int n;
int * pt = new int;
const int b = 101;  // can't assign to b,
int & rn = n;       // n identifies datum
int & rt = *pt;     // *pt identifies datu
const int & rb = b; // b identifies const
```

C++11 adds the rvalue reference (discusse can bind to rvalues—that is, values that can ap

```
#include <iostream>

inline double f(double tf) {return 5.0*(t
int main()
{
    using namespace std;
    double tc = 21.5;
    double && rd1 = 7.07;
    double && rd2 = 1.8 * tc + 32.0;
    double && rd3 = f(rd2);
    cout << " tc value and address: " <<
    cout << "rd1 value and address: " <<
    cout << "rd2 value and address: " <<
    cout << "rd3 value and address: " <<
    cin.get();
    return 0;
}
```

Here is a sample output:

```
 tc value and address: 21.5, 002FF744
rd1 value and address: 7.07, 002FF728
rd2 value and address: 70.7, 002FF70C
rd3 value and address: 21.5, 002FF6F0
```

constructor, which will use `new` to allocate m
20,000,000 characters will be copied from th
ory controlled by `vstr_copy1`. That's a lot of
to be done.

But does it "got to be done?" There are ti
suppose we have a function that returns a `vec`

```
vector<string> allcaps(const vector<strin
{
    vector<string> temp;
// code that stores an all-uppercase vers
    return temp;
}
```

Next, suppose we use it this way:

```
vector<string> vstr;
// build up a vector of 20,000 strings, e
vector<string> vstr_copy1(vstr);
vector<string> vstr_copy2(allcaps(vstr));
```

Superficially, statements #1 and #2 are sim
a new `vector<string>` object. If we take thi
`temp` object managing 20,000,000 characters,
go through the effort of creating a 20,000,00

A Move Example

Let's look at an example to see how move se
18.2 defines and uses the `Useless` class, whic
a regular copy constructor, and a move const
rvalue reference. In order to illustrate the pro
destructor are unusually verbose, and the clas
number of objects. Also some important met
omitted. (Despite these omissions, the `Usele`
eco-friendly `Use_Less` class.)

Listing 18.2 **useless.cpp**

```
// useless.cpp -- an otherwise useless cl
#include <iostream>
using namespace std;

// interface
class Useless
{
private:
    int n;          // number of elements
    char * pc;      // pointer to data
    static int ct;  // number of objects
```

```
}

Useless::Useless(int k) : n(k)
{
    ++ct;
    cout << "int constructor called; numbe
    pc = new char[n];
    ShowObject();
}

Useless::Useless(int k, char ch) : n(k)
{
    ++ct;
    cout << "int, char constructor called,
         << endl;
    pc = new char[n];
    for (int i = 0; i < n; i++)
        pc[i] = ch;
    ShowObject();
}

Useless::Useless(const Useless & f): n(f.n
{
```

```
    delete [] pc;
}

Useless Useless::operator+(const Useless
{
    cout << "Entering operator+()\n";
    Useless temp = Useless(n + f.n);
    for (int i = 0; i < n; i++)
        temp.pc[i] = pc[i];
    for (int i = n; i < temp.n; i++)
        temp.pc[i] = f.pc[i - n];
    cout << "temp object:\n";
    cout << "Leaving operator+()\n";
    return temp;
}

void Useless::ShowObject() const
{
    cout << "Number of elements: " << n;
    cout << " Data address: " << (void *)
}
```

```
        three.ShowData();
        cout << "object four: ";
        four.ShowData();
    }
}
```

The crucial definitions are those of the tw
the output statements, here is the copy constr

```
Useless::Useless(const Useless & f): n(f.
{
    ++ct;
    pc = new char[n];
    for (int i = 0; i < n; i++)
        pc[i] = f.pc[i];
}
```

It does the usual deep copy, and it is the c
statement:

```
Useless two = one;           // calls cop
```

The reference `f` refers to the lvalue object ence f binds to the rvalue temporary object

Here's the output when the program was

```
int, char constructor called; number of o
Number of elements: 10 Data address: 006F
copy const called; number of objects: 2
Number of elements: 10 Data address: 006F
int, char constructor called; number of o
Number of elements: 20 Data address: 006F
Entering operator+()
int constructor called; number of objects
Number of elements: 30 Data address: 006F
temp object:
Leaving operator+()
move constructor called; number of object
Number of elements: 30 Data address: 006F
destructor called; objects left: 4
deleted object:
Number of elements: 0 Data address: 00000
object one: xxxxxxxxxx
object two: xxxxxxxxxx
object three: oooooooooooooooooooo
object four: xxxxxxxxxxoooooooooooooooooo
destructor called; objects left: 3
```

```
Number of elements: 10 Data address: 0xa50
copy const called; number of objects: 2
Number of elements: 10 Data address: 0xa50
int, char constructor called; number of ob
Number of elements: 20 Data address: 0xa50
Entering operator+()
int constructor called; number of objects
Number of elements: 30 Data address: 0xa50
temp object:
Leaving operator+()
object one: xxxxxxxxxx
object two: xxxxxxxxxx
object three: oooooooooooooooooooo
object four: xxxxxxxxxxoooooooooooooooooooo
destructor called; objects left: 3
deleted object:
Number of elements: 30 Data address: 0xa50
destructor called; objects left: 2
deleted object:
Number of elements: 20 Data address: 0xa50
destructor called; objects left: 1
deleted object:
Number of elements: 10 Data address: 0xa50
```

directs initialization for object four to the m
move semantics is coding the move construc

In short, the presence of one constructor
structor with an rvalue reference sorts possib
tialized with an lvalue object use the copy co
rvalue object use the move constructor. The
tors with different behaviors.

This raises the question of what happened
language. If there is no move constructor and
need for the copy constructor, what should h
ment would invoke the copy constructor:

```
Useless four (one + three);
```

But an lvalue reference doesn't bind to an
from Chapter 8, a `const` reference parameter
if the actual argument is an rvalue:

```
int twice(const & rx) {return 2 * rx;}
...
int main()
{
    int m = 6;
    // below, rx refers to m
    int n = twice(m);
```

```
Number of elements: 30 Data address: 01C3
...
```

First, within the `Useless::operator+()` r
ing storage for 30 elements at location `01C33`
temporary copy to which `f` will refer, copyin
Next, `temp`, which uses location `01C337C4`, ge
structed, reusing the recently freed memory a
object, which used location `01C337E8`, gets de
structed, and two of them were destroyed. Th
tics are meant to eliminate.

As the g++ example shows, an optimizing
its own, but using an rvalue reference lets the
appropriate.

Assignment

The same considerations that make move sem
them appropriate for assignment. Here, for ex
assignment and the move assignment operato

```
Useless & Useless::operator=(const Useless
{
    if (this == &f)
```

object. It's important that only one pointer p
pointer in the source object to the null point

As with the move constructor, the move a
reference because the method alters the sour

Forcing a Move

Move constructors and move assignment ope
to use them with lvalues? For instance, a prog
candidate objects, select one object for furthe
venient if you could use a move constructor
the selected object. However, suppose you tr

```
Useless choices[10];
Useless best;
int pick;
... // select one object,  set pick to in
best = choices[pick];
```

The `choices[pick]` object is an lvalue, so
assignment operator, not the move assignmen
`choices[pick]` look like an rvalue, then the
This can be done by using the `static_cast`
`Useless &&`. C++11 provides a simpler way

```
    Useless(Useless && f);      // move co
    ~Useless();
    Useless operator+(const Useless & f)co
    Useless & operator=(const Useless & f)
    Useless & operator=(Useless && f);
    void ShowData() const;
};

// implementation
int Useless::ct = 0;

Useless::Useless()
{
    ++ct;
    n = 0;
    pc = nullptr;
 }

Useless::Useless(int k) : n(k)
{
    ++ct;
    pc = new char[n];
}
```

```
}

Useless::~Useless()
{
    delete [] pc;
}

Useless & Useless::operator=(const Useles
{
    std::cout << "copy assignment operato
    if (this == &f)
        return *this;
    delete [] pc;
    n = f.n;
    pc = new char[n];
    for (int i = 0; i < n; i++)
        pc[i] = f.pc[i];
    return *this;
}

Useless & Useless::operator=(Useless && f
{
    std::cout << "move assignment operato
    if (this == &f)
```

```
}

void Useless::ShowData() const
{
    if (n == 0)
        std::cout << "(object empty)";
    else
        for (int i = 0; i < n; i++)
            std::cout << pc[i];
    std::cout << std::endl;
}

// application
int main()
{
    using std::cout;
    {
        Useless one(10, 'x');
        Useless two = one +one;   // calls
        cout << "object one: ";
        one.ShowData();
        cout << "object two: ";
        two.ShowData();
```

```
object two: xxxxxxxxxxxxxxxxxxxx
three = one
copy assignment operator called:
now object three = xxxxxxxxxx
and object one = xxxxxxxxxx
four = one + two
move assignment operator called:
now object four = xxxxxxxxxxxxxxxxxxxxx
four = move(one)
move assignment operator called:
now object four = xxxxxxxxxx
and object one = (object empty)
```

As you can see, assigning `one` to `three` inv move(one) to four invokes move assignmen

You should realize that the `std::move()` operation. Suppose, for instance, that `Chunk` i the following code:

```
Chunk one;
...
Chunk two;
two = std::move(one);  // move semantics?
```

The default constructor, recall, is a construc
compiler provides one if you fail to define any
of a default constructor is termed the *defaulted*
constructor leaves members of the built-in typ
default constructors for members that are class

Also the compiler provides a defaulted cop
if your code requires its use, and it now provi
don't provide one and if your code requires it
two defaulted constructors have the following

```
Someclass::Someclass(const Someclass &);
Someclass::Someclass(Someclass &&);
```

In similar circumstances, the compiler prov
and a defaulted move assignment operator wi

```
Someclass & Someclass::operator(const Some
Someclass & Someclass::operator(Someclass
```

Finally, the compiler provides a destructor

There are various exceptions to this descri
copy constructor or a copy assignment operat
vide a move constructor or a move assignmen
structor or a move assignment operator, the c
copy constructor or a copy assignment operat

```
   Someclass & operator (const Someclass
...
};
```

The compiler provides the same construc
had you not provided the move constructor.

The `delete` keyword, on the other hand,
using a particular method. For example, to pr
disable the copy constructor and copy assign

```
class Someclass
{
public:
   Someclass() = default;      // use com
// disable copy constructor and copy assi
   Someclass(const Someclass &) = delete;
   Someclass & operator=(const Someclass
// use compiler-generated move constructo
   Someclass(Someclass &&) = default;
   Someclass & operator=(Someclass &&) =
   Someclass & operator+(const Someclass
...
};
```

Then suppose we have the following code

```
Someclass sc;
sc.redo(5);
```

The `int` value `5` will be promoted to `5.0`,

Now suppose the `Someclass` definition is

```
class Someclass
{
public:
...
    void redo(double);
    void redo(int) = delete;
...
};
```

In this case, the method call `sc.redo(5)` m
piler will detect that fact and also detect that
the call as a compile-time error. This illustrate
They do exist as far as function look-up is co

Delegating Constructors

If you provide a class with several constructor
code over and over. That is, some of the const

the data members and to also do whatever th
finishes up doing whatever its own body req

Inheriting Constructors

In another move to simplify coding, C++11
inherit constructors from the base class. C++
from a namespace available:

```
namespace Box
{
    int fn(int) { ... }
    int fn(double) { ... }
    int fn(const char *) { ... }
}
using Box::fn;
```

This makes all the overloaded `fn` function
making nonspecial member functions of a ba
ple, consider the following code:

```
class C1
{
...
public:
...
```

```
class BS
{
    int q;
    double w;
public:
    BS() : q(0), w(0) {}
    BS(int k) : q(k), w(100) {}
    BS(double x) : q(-1), w(x) {}
    B0(int k, double x) : q(k), w(x) {}
    void Show() const {std::cout << q <<",
};

class DR : public BS
{
    short j;
public:
    using BS::BS;
    DR() : j(-100) {}      // DR needs its
    DR(double x) : BS(2*x), j(int(x)) {}
    DR(int i) : j(-2), BS(i, 0.5* i) {}
    void Show() const {std::cout << j << "
};
int main()
```

signature, you hide rather than override the c

```
class Action
{
    int a;
public:
    Action(int i = 0) : a(i) {}
    int val() const {return a;};
    virtual void f(char ch) const { std::
};
class Bingo : public Action
{
public:
    Bingo(int i = 0) : Action(i) {}
    virtual void f(char * ch) const { std
};
```

Because class `Bingo` uses `f(char * ch)` in to a `Bingo` object. This prevents a program fr

```
Bingo b(10);
b.f('@');  // works for Action object, fa
```

With C++11, you can use the virtual spec to override a virtual function. Place it after th match a base method, the compiler objects. T would generate a compile-time error messag

grammer. You will have your suspicions seem
functions actually look—here's an example:

```
[&count](int x){count += (x % 13 == 0);}
```

But they aren't as arcane as they may look,
ularly with STL algorithms using function pr

The How of Function Pointers, Fu

Let's look at an example using three approach
rithm: function pointers, functors, and lambda
three forms as *function objects* so that we won't
functor or lambda.") Suppose you wish to ge
mine how many of them are divisible by 3 an
imagine that this is a quest you find absolutel

Generating the list is pretty straightforwar
to hold the numbers and use the STL genera
dom numbers:

```
#include <vector>
#include <algorithm>
#include <cmath>
...
std::vector<int> numbers(1000);
std::generate(vector.begin(), vector.end()
```

recall from Chapter 16, is a class object than
thanks to the class defining operator()() as
tor in our example is that you can use the sa
one possible definition:

```
class f_mod
{
private:
    int dv;
public:
    f_mod(int d = 1) : dv(d) {}
    bool operator()(int x) {return x % dv
};
```

Recall how this works. You can use the co
particular integer value:

```
f_mod obj(3);  // f_mod.dv set to 3
```

This object can use the operator() meth

```
bool is_div_by_3 = obj(7); // same as obj
```

lambda doesn't have a return statement, the ty
you would use this lambda as follows:

```
count3 = std::count_if(numbers.begin(), nu
         [](int x){return x % 3 == 0;});
```

That is, you use the entire lambda expressi
constructor.

The automatic type deduction for lambdas
return statement. Otherwise, you need to use

```
[](double x)->double{int y = x; return x -
```

Listing 18.4 illustrates the points just discu

Listing 18.4 **lambda0.cpp**

```
// lambda0.cpp -- using lambda expressions
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <ctime>
const long Size1 = 39L;
const long Size2 = 100*Size1;
const long Size3 = 100*Size2;
```

```
// using a functor
    class f_mod
    {
    private:
        int dv;
    public:
        f_mod(int d = 1) : dv(d) {}
        bool operator()(int x) {return x
    };

    count3 = std::count_if(numbers.begin(
    cout << "Count of numbers divisible b
    count13 = std::count_if(numbers.begin
    cout << "Count of numbers divisible b

// increase number of numbers again
    numbers.resize(Size3);
    std::generate(numbers.begin(), number
    cout << "Sample size = " << Size3 <<
// using lambdas
    count3 = std::count_if(numbers.begin(
             [](int x){return x % 3 == 0;
    cout << "Count of numbers divisible b
```

The Why of Lambdas

You may be wondering what need, other thar
lambda serves. Let's examine this question in t
efficiency, and capability.

Many programmers feel that it is useful to
used. That way, you don't have to scan throug
third argument to a `count_if()` function call
the code, all the components are close at hanc
elsewhere, again all the components are at har
because the definition is at the point of usage
cannot be defined inside other functions, so t
far from the point of usage. Functors can be p
functor class, can be defined inside a function
the point of use.

In terms of brevity, the functor code is mo
lambda code. Functions and lambdas are appro
exception would be if you had to use a lambd

```
count1 = std::count_if(n1.begin(), n1.end
         [](int x){return x % 3 == 0;});
count2 = std::count_if(n2.begin(), n2.end
         [](int x){return x % 3 == 0;});
```

ence. Using [&] provides access to all the au
vides access to all the automatic variables by
instance, [ted, &ed] would provide access t
ted] would provide access to ted by value a
ence, and [=, &ed] would provide access by
ing automatic variables. In Listing 18.4, you

```
int count13;
...
count13 = std::count_if(numbers.begin(),
          [](int x){return x % 13 == 0;})
```

with this:

```
int count13 = 0;
std::for_each(numbers.begin(), numbers.en
     [&count13](int x){count13 += x % 13
```

The [&count13] allows the lambda to us
captured by reference, any changes to count
count13. The expression x % 13 == 0 evalu
converts to 1 when added to count13. Simil
for_each() applies the lambda expression t
the number of elements divisible by 13.

```
    using std::cout;
    std::vector<int> numbers(Size);

    std::srand(std::time(0));
    std::generate(numbers.begin(), numbers
    cout << "Sample size = " << Size << '\
// using lambdas
    int count3 = std::count_if(numbers.beg
          [](int x){return x % 3 == 0;});
    cout << "Count of numbers divisible by
    int count13 = 0;
    std::for_each(numbers.begin(), numbers
         [&count13](int x){count13 += x %
    cout << "Count of numbers divisible by
// using a single lambda
    count3 = count13 = 0;
    std::for_each(numbers.begin(), numbers
         [&](int x){count3 += x % 3 == 0;
    cout << "Count of numbers divisible by
    cout << "Count of numbers divisible by
    return 0;
}
```

plied as an argument. C++11 provides additi
plate, which provides a more flexible alternat
template, which allows a member function to
`reference_wrapper` template allows you to c
which can be copied, and the `function` wra
function-like forms uniformly.

Let's look more closely at one example of
problem it addresses.

The `function` Wrapper and Tem

Consider the following line of code:

```
answer = ef(q);
```

What is `ef`? It could be the name of a fun
could be a function object. It could be a nam
are examples of *callable types*. The abundance
ciencies. To see this, let's examine a simple ca

First, let's define some templates in a head

```
class Fq
{
private:
    double z_;
public:
    Fq(double z = 1.0) : z_(z) {}
    double operator()(double q) { return z
};
```

The use_f() template uses the parameter

```
return f(v);
```

Next the program in Listing 18.7 calls the

Listing 18.7 **callable.cpp**

```
// callable.cpp -- callable types and temp
#include "somedefs.h"
#include <iostream>

double dub(double x) {return 2.0*x;}
double square(double x) {return x*x;}
```

parameter `F`? Each time the actual argument
ment and returns a type `double` value, so it m
all six calls to `use_f()` and that the template
following sample output shows, that belief is

```
Function pointer dub:
  use_f count = 1, &count = 0x402028
  2.42
Function pointer square:
  use_f count = 2, &count = 0x402028
  1.1
Function object Fp:
  use_f count = 1, &count = 0x402020
  6.05
Function object Fq:
  use_f count = 1, &count = 0x402024
  6.21
Lambda expression 1:
  use_f count = 1, &count = 0x405020
  1.4641
Lambda expression 2:
  use_f count = 1, &count = 0x40501c
  1.815
```

The function wrapper lets you rewrite the pr
use_f() instead of five. Note that the functio
expressions in Listing 18.7 share a common b
ment and each returns a type double value. W
signature, which is described by the return typ
parameter types enclosed in a pair of parenthe
double(double) as the call signature.

The function template, declared in the fu
terms of a call signature, and it can be used to
or lambda expression having the same call sig
tion creates a function object fdci that take
type double:

```
std::function<double(char, int)> fdci;
```

You can then assign to fdci any function p
sion that takes type char and int arguments a

The various callable arguments in Listing 1
double(double). So to fix Listing 18.7 and re
use function<double(double)> to create six
and lambdas. Then all six calls to use_f() can
(function<double(double)>) for F, resulting
the result.

```
    cout << "Function pointer dub:\n";
    cout << "  " << use_f(y, ef1) << endl
    cout << "Function pointer square:\n";
    cout << "  " << use_f(y, ef2) << endl
    cout << "Function object Fp:\n";
    cout << "  " << use_f(y, ef3) << endl
    cout << "Function object Fq:\n";
    cout << "  " << use_f(y, ef4) << endl
    cout << "Lambda expression 1:\n";
    cout << "  " << use_f(y, ef5) << endl
    cout << "Lambda expression 2:\n";
    cout << "  " << use_f(y,ef6) << endl;
    return 0;
}
```

Here is a sample output:

```
Function pointer dub:
  use_f count = 1, &count = 0x404020
  2.42
Function pointer sqrt:
  use_f count = 2, &count = 0x404020
  1.1
```

function:

```
typedef function<double(double)> fdd; // s
cout << use_f(y, fdd(dub)) << endl;   // c
cout << use_f(y, fdd(square)) << endl;
...
```

Second, Listing 18.8 adapts the second arg parameter `f`. Another approach is to adapt the the original arguments. This can be done by u ond parameter for the `use_f()` template defin

```
#include <functional>
template <typename T>
T use_f(T v,  std::function<T(T)> f)   //
{
    static int count = 0;
    count++;
    std::cout << "  use_f count = " << cou
              << ", &count = " << &count <
    return f(v);
}
```

Then the function calls can look like this:

```
cout << "  " << use_f<double>(y, dub) << e
...
```

The goal is to be able to define show_lis
pile and lead to this output:

```
14, 2.71828
7.38905, !, 7, Mr. String objects!
```

There are a few key points to understand

- Template parameter packs
- Function parameter packs
- Unpacking a pack
- Recursion

Template and Function Paramete

As a starting point to see how parameter pac
function, one that displays a list consisting of

```
template<typename T>
void show_list0(T value)
{
    std::cout << value << ", ";
}
```

Next, much as

```
void show_list0(T value)
```

states that `value` is of type `T`, the line

```
void show_list1(Args... args)  // args is
```

states that `args` is of type `Args`. More precisel
contains a list of values that matches the list o
type and in number. In this case, `args` contain

In this manner, the `show_list1()` variadic
function calls:

```
show_list1();
show_list1(99);
show_list1(88.5, "cat");
show_list1(2,4,6,8, "who do we", std::str
```

In the last case, the `Args` template paramet
`int`, `int`, `const char *`, and `std::string`, an
contain the matching values `2`, `4`, `6`, `8`, `"who d`

Unpacking the Packs

But how can the function access the contents
ture. That is, you can't use something like `Arg`

Using Recursion in Variadic Temp

Although recursion dooms show_list1() as
used recursion provides a solution to accessin
the function parameter pack, process the first
on to a recursive call, and so on, until the list
tant to make sure that there is a call that tern
involves changing the template heading to th

```
template<typename T, typename... Args>
void show_list3( T value, Args... args)
```

With this definition, the first argument to
is assigned to value. The remaining argumen
allows the function to do something with va
arguments, in the form args..., can be pass
recursive call then prints a value and passes o

Listing 18.9 presents an implementation t
technique.

Listing 18.9 **variadic1.cpp**

```
//variadic1.cpp -- using recursion to unp
#include <iostream>
#include <string>
```

Program Notes

Consider this function call:

```
show_list3(x*x, '!', 7, mr);
```

The first argument matches `T` to `double` a (`char`, `int`, and `std::string`) are placed in th (`'!'`, `7`, and `mr`) are placed in the `args` pack.

Next, the `show_list3()` function uses `co` and the string `", "`. That takes care of display

Next comes this call:

```
show_list3(args...);
```

This, given the expansion of `args...`, is th

```
show_list3('!', 7, mr);
```

As promised, the list is shortened by one it and `'!'`, and the remaining two types and val tively. The next recursive call processes these r the version of `show_list3()` with no argume

Here is the output for the two function ca

```
14, 2.71828, 7.38905, !, 7, Mr. String ob
```

you can use this:

```
show_list3(const Args&... args);
```

That will cause each function parameter t
instead of `std::string mr`, the final paring
`std::string& mr`.

Listing 18.10 incorporates these two chan

Listing 18.10 **variadic2.cpp**

```
// variadic2.cpp
#include <iostream>
#include <string>

// definition for 0 parameters
void show_list() {}

// definition for 1 parameter
template<typename T>
void show_list(const T& value)
{
    std::cout << value << '\n';
}
```

More C++11 Features

C++11 adds many more features than this bo
many of them are not widely implemented at
worth taking a quick look at the nature of so

Concurrent Programming

These days it's easier to improve computer pe
increase processor speed. So computers with c
with multiple multicore processors are the no
threads of execution simultaneously. One proc
another processor handles your spreadsheet.

Some activities can benefit from multiple t
searching for something in a singly linked list
follow the links, in order, to the end of the lis
could do to help. Now consider an unsorted a
arrays, you could start one thread from the be
dle, thus halving the search time.

Multiple threads do raise many problems. V
two threads try to access the same data simult

tor can be represented by the widest integer
for these numbers.

One of the most interesting additions is a
regex header file. A regular expression specif
tents in a text string. For example, a bracket e
the brackets. Thus, [cCkK] matches a single c
cat, Cat, kat, and Kat. Other patterns includ
and many, many others. The fact that a backsl
character in an escape sequence requires a pa
digit) to be written as the string literal "\\d\
one reason the raw string was introduced (se
same pattern as R"\d\t\w\t".

Unix utilities such as ed, grep, and awk us
language Perl extended their capabilities. The
to choose from several flavors of regular expr

Low-Level Programming

The "low level" in low-level programming re
quality of the programming. Low level mean
hardware and machine language. Low-level p
gramming and for increasing the efficiency o
to those who do low-level programming.

to allow `const` variables to be stored in read-
ful in embedded programming. (Variables, `co`
runtime, are stored in random-access memory

Miscellaneous

C++11 follows the lead of C99 in allowing f
integer types. Such types, for example, could
Extended types are supported in the C heade
`cstdint`.

C++11 provides a mechanism, the *literal o*
Using this mechanism, for instance, one can d
which the corresponding literal operator will

C++ has a debugging tool called `assert`.
an assertion is true and which displays a mess
The assertion would typically be about somet
at that point in the program. C++11 adds the
used to test assertions during compile time. T
debug templates for which instantiation takes

C++11 provides more support for metapr
create or modify other programs or even the
compile time using templates.

additional compiler support. And if they are f
distributed as text files in the form of header

One example of this sort of change is the
Stepanov, and made freely available by Hewle
ming community made it a candidate for the
influenced other aspects of the emerging stan

The Boost Project

More recently, the Boost library has become
has had a significant influence on C++11. Th
Dawes, the then-chairman of the C++ librar
other members of the group and developed a
confines of the standards committee. The bas
an open forum for people to post free C++
licensing and programming practices, and it r
The result is a group of highly praised and hi
environment in which the programming com
ideas and provide feedback.

Boost has over 100 libraries at the time of
a set from www.boost.org, as can documenta
including the appropriate header files.

types. The syntax is modeled after dynamic_c
a template parameter. Listing 18.11 shows a si

Listing 18.11 **lexcast.cpp**

```
// lexcast.cpp -- simple cast from float t
#include <iostream>
#include <string>
#include "boost/lexical_cast.hpp"
int main()
{
    using namespace std;
    cout << "Enter your weight: ";
    float weight;
    cin >> weight;
    string gain = "A 10% increase raises "
    string wt = boost::lexical_cast<string
    gain = gain + wt + " to ";  // string
    weight = 1.1 * weight;
    gain = gain + boost::lexical_cast<stri
    cout << gain << endl;
    return 0;
}
```

What Now?

If you have worked your way through this bo
rules of C++. However, that's just the beginn
stage is learning to use the language effective
situation is to be in a work or learning enviro
good C++ code and programmers. Also now
that concentrate on more advanced topics an
Appendix H, "Selected Readings and Interne

One promise of OOP is to facilitate the d
large projects. One of the essential activities o
that represent the situation (called the *problem*
problems are often complex, finding a suitabl
complex system from scratch usually doesn't
evolutionary approach. Toward this end, prac
techniques and strategies. In particular, it's im
evolution in the analysis and design stages as
actual code.

Two common techniques are *use-case analy*
development team lists the common ways, or
tem to be used, identifying elements, actions,

Summary

The new C++ standard adds many features to
the language easier to learn and easier to use.
tialization, automatic type deduction with `au`
range-based `for` loop. Other changes expand
include defaulted and deleted methods, deleg
and the `override` and `final` specifiers for cl

Several additions aim to make programs an
Lambda expressions provide advantages over f
`function` template can be used to reduce the
rvalue reference enables move semantics and
assignment operators.

Other changes deliver better ways of doin
ter control over the scope and underlying typ
`shared_ptr` templates provide better ways of

Template design has been enhanced with t
types, template aliases, and variadic templates.

Modified rules for unions, PODs, the `ali`
the `constexpr` mechanism support low-level

Several library additions, including the ne
`regex` library, provide solutions to many com

```
int k(99);
Z200 zip(200,'Z',0.675);
std::vector<int> ai(5);
int ar[5] = {3, 9, 4, 7, 1};
for (auto pt = ai.begin(), int i =
    *pt = ai[i];
```

2. For the following short program, whic
valid calls, what does the reference argu

```
#include <iostream>
using namespace std;

double up(double x) { return 2.0* x
void r1(const double &rx) {cout << 
void r2(double &rx) {cout << rx << 
void r3(double &&rx) {cout << rx <<

int main()
{
    double w = 10.0;
    r1(w);
    r1(w+1);
    r1(up(w));
```

```
    r1(up(w));
    return 0;
}
```

b. What does the following short pr

```
#include <iostream>
using namespace std;

double up(double x) { return 2
void r1(double &rx) {cout << "
void r1(double &&rx) {cout <<

int main()
{
    double w = 10.0;
    r1(w);
    r1(w+1);
    r1(up(w));
    return 0;
}
```

c. What does the following short pr

```
#include <iostream>
using namespace std;
```

Why would this class not be a good ca
What change in approach to storing th
good candidate for a move function?

6. Revise the following short program so
`f1()`. Don't change `show2()`.

```
#include <iostream>
template<typename T>
  void show2(double x, T& fp) {std:
double f1(double x) { return 1.8*x
int main()
{
    show2(18.0, f1);
    return 0;
}
```

7. Revise the following short and ugly pr
instead of the `Adder` functor. Don't ch

```
#include <iostream>
#include <array>
const int Size = 5;
template<typename T>
```

```
    {
        fp(*pt);
    }
}
```

Programming Exercises

1. Here is part of a short program:

```
int main()
{
    using namespace std;
 // list of double deduced from list
    auto q = average_list({15.4, 10.
    cout << q << endl;
// list of int deduced from list con
    cout << average_list({20, 30, 19
// forced list of double
    auto ad = average_list<double>({
    cout << ad << endl;
    return 0;
}
```

```
    Cpmv & operator=(const Cpmv & cp
    Cpmv & operator=(Cpmv && mv);
    Cpmv operator+(const Cpmv & obj
    void Display() const;
};
```

The `operator+()` function should cre
bers concatenate the corresponding m
implements move semantics for the m
operator. Write a program that uses all
various methods verbose so that you c

3. Write and test a variadic template func
ily long list of arguments with numeri
returns the sum as a `long double` valu

4. Redo Listing 16.5 using lambdas. In pa
a named lambda and replace the two u
expressions.

(binary).

Decimal Numbers (Base

The method we use for writing numbers is b
the number 2,468. The 2 represents 2 thousa
sents 6 tens, and the 8 represents 8 ones:

$$2,468 = 2 \quad 1,000 + 4 \quad 100 + 6 \quad 10 + 8 \quad 1$$

One thousand is 10 10 10, which can be
Using this notation, you can write the preced

$$2,468 = 2 \quad 10^3 + 4 \quad 10^2 + 6 \quad 10^1 + 8 \quad 10^0$$

Because this number notation is based on
decimal, notation. You can also use another n
use base 8 (octal) and base 16 (hexadecimal o
(Note that 10^0 is 1, as is any nonzero numbe

Octal Integers (Base 8)

Octal numbers are based on powers of 8, so b
numbers. C++ uses a 0 prefix to indicate oct
can use powers of 8 to find the equivalent ba

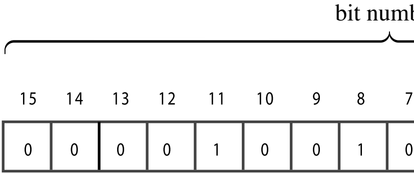
Hexadecimal Digit	Decimal Va
a or A	10
b or B	11
c or C	12
d or D	13
e or E	14
f or F	15

C++ uses 0x or 0X notation to indicate he
decimal value. To find the decimal equivalent

Hexadecimal	Decimal
0x2B3	$= 2\ \ 16^{2} + 11\ \ 16^{1} + 3\ \ 1$
	$= 2\ \ 256 + 11\ \ 16 + 3\ \ 1$
	$= 691$

Hardware documentation often uses hexac
memory locations and port numbers.

represents a 2-byte integer.




```
value = 1 x 2^11 +
      = 2048 + 256
      = 2338
```

Figure A.1 A 2-

Binary and Hex

Hex notation is often used to provide a more
memory addresses or integers holding bit-fla
hexadecimal digit corresponds to a 4-bit uni

To convert a hex value to binary, you just r
ding binary equivalent. For example, the hex
0100. Similarly, you can easily convert binary
4-bit unit into the equivalent hex digit. For e
becomes 0x95.

Big Endian and Little Endian

Oddly, two computing platforms that both use
represent the same number identically. Intel r
Little Endian architecture, whereas Motorola p
processors, and ARM processors employ the
tems can be configured to use either scheme

The terms *Big Endian* and *Little Endian* can be
tle End In"—a reference to the order of bytes
On an Intel computer (Little Endian), the low-o
value such as `0xABCD` would be stored in me
machine would store the same value in revers
`0xAB 0xCD`.

Jonathan Swift's book *Gulliver's Travels* is the
the irrationality of many political disputes by i

keywords. Keywords shown in boldface are a
ics are C++11 additions.

Table B.1 **C++ Keywords**

alignas	*alignof*	asm
break	**case**	cat
char32_t	class	**con**
continue	*decltype*	**def**
double	dynamic_cast	**els**
export	**extern**	fal
friend	**goto**	**if**
long	mutable	nam
nullptr	operator	pri
register	reinterpret_cast	**ret**
sizeof	**static**	*sta*
switch	template	thi
true	try	**typ**
union	**unsigned**	usi
volatile	wchar_t	**whi**

xor_eq	^=

C++ Library Reserved Na

The compiler won't let you use keywords anc
class of forbidden names for which the protec
which are names reserved for use by the C++
fier, the effect is undefined. That is, it might g
warning, it might cause a program to run inco

The C++ language reserves macro names
includes a particular header file, then you sho
that header (or in headers included by that he
example, if you include the header file <clim
CHAR_BIT as an identifier because that name i

The C++ language reserves names beginn
score followed by an uppercase letter for any
single underscore for use as a global variable. S
__Lynx in any case and names such as _lynx

keyword and provided more than one use fo
C++11 has implemented another way to avo
tifiers with special meanings. These identifier
they are used to implement language feature
mine whether they are used as ordinary iden

```
class F
{
   int final;                          /
public:
...
   virtual void unfold() {...} = final;
};
```

Here the `final` on line #1 is used as an c
is used to invoke a language feature. The two

Also C++ has many identifiers that comm
reserved. These include header file names, lib
the required function with which execution
flicts, you can use these identifiers for other p
so. That is, nothing except a lack of common

```
// allowable but silly
#include <iostream>
int iostream(int a);
```

prefix in the table, the ^ character denotes us

Table C.1 **ASCII Character Set Representati**

Decimal	Octal	Hex	Binary
0	0	0	00000
1	01	0x1	00000
2	02	0x2	00000
3	03	0x3	00000
4	04	0x4	00000
5	05	0x5	00000
6	06	0x6	00000
7	07	0x7	00000
8	010	0x8	00001
9	011	0x9	00001
10	012	0xa	00001
11	013	0xb	00001
12	014	0xc	00001
13	015	0xd	00001

30	036	0x1e	0001111
31	037	0x1f	0001111
32	040	0x20	0010000
33	041	0x21	0010000
34	042	0x22	0010000
35	043	0x23	0010000
36	044	0x24	0010001
37	045	0x25	0010001
38	046	0x26	0010001
39	047	0x27	0010001
40	050	0x28	0010100
41	051	0x29	0010100
42	052	0x2a	0010100
43	053	0x2b	0010100
44	054	0x2c	0010111
45	055	0x2d	0010111
46	056	0x2e	0010111

63	077	0x3f	00111
64	0100	0x40	01000
65	0101	0x41	01000
66	0102	0x42	01000
67	0103	0x43	01000
68	0104	0x44	01000
69	0105	0x45	01000
70	0106	0x46	01000
71	0107	0x47	01000
72	0110	0x48	01001
73	0111	0x49	01001
74	0112	0x4a	01001
75	0113	0x4b	01001
76	0114	0x4c	01001
77	0115	0x4d	01001
78	0116	0x4e	01001
79	0117	0x4f	01001

96	0140	0x60	011000
97	0141	0x61	011000
98	0142	0x62	011000
99	0143	0x63	011000
100	0144	0x64	011001
101	0145	0x65	011001
102	0146	0x66	011001
103	0147	0x67	011001
104	0150	0x68	011010
105	0151	0x69	011010
106	0152	0x6a	011010
107	0153	0x6b	011010
108	0154	0x6c	011011
109	0155	0x6d	011011
110	0156	0x6e	011011
111	0157	0x6f	011011
112	0160	0x70	011100

the table). Left-to-right associativity means to
right-to-left associativity means to apply the

Table D.1 **C++ Operator Precedence and As**

Operator	Assoc.	M
Precedence Group 1		
`::`		S
Precedence Group 2		
`(expression)`		G
`()`	L–R	Fu
`()`		Va
`[]`		A
`.`		D
`->`		In
`++`		In
`--`		D
`const_cast`		S
`dynamic_cast`		S
`reinterpret_cast`		S
`static_cast`		S
`typeid`		Ty

`noexcept`		Fa
Precedence Group 4		
`.*`	L–R	Mo
`->*`		Inc
Precedence Group 5 (All Binary)		
`*`	L–R	Mu
`/`		Di
`%`		Mo
Precedence Group 6 (All Binary)		
`+`	L–R	Ad
`-`		Su
Precedence Group 7		
`<<`	L–R	Le
`>>`		Rig
Precedence Group 8		
`<`	L–R	Le
`<=`		Le
`>=`		Gr
`>`		Gr

=	R–L	S
*=		M
/=		D
%=		Ta
+=		A
-=		S
&=		B
^=		B
|=		B
<<=		Lo
>>=		R
Precedence Group 17		
throw	L–R	Th
Precedence Group 18		
,	L–R	C

```
char ch = *str++;
```

The postfix ++ operator has higher preced
the increment operator operates on str and r
the pointer, making it point to the next chara
pointed to. However, because ++ is the postfix
original value of *str is assigned to ch. There
to ch and then moves str to point to the h c
Here's a similar example:

```
char * str = "Whoa";
char ch = *++str;
```

The prefix ++ operator and the unary * op
associate right-to-left. So, again, str and not
tor is in prefix form, first str is incremented,
enced. Thus, str moves to point to the h char

Note that Table D.1 uses *binary* or *unary*
between two operators that use the same sym
the binary bitwise AND operator.

Appendix B, "C++ Reserved words," lists
operators.

Bitwise Operators

The bitwise operators operate on the bits of
operator moves bits to the left, and the bitwi
each `0` to a `1`. Altogether, C++ has six such o

The Shift Operators

The left-shift operator has the following synt

```
value << shift
```

Here *`value`* is the integer value to be shif
For example, the following shifts all the bits i

```
13 << 3
```

The vacated places are filled with zeros, ar
Figure E.1).

Because each bit position represents a valu
Appendix A, "Number Bases"), shifting one b
value by 2. Similarly, shifting two bit position
shifting n positions is equivalent to multiplyi
13 2^3, or 104.

produce a new value, much as `x + 3` produce

If you want to use the left-shift operator to also use assignment. You can use regular assign shifting with assignment:

```
x = x << 4;           // regular assignmen
y <<= 2;              // shift and assign
```

The right-shift operator (`>>`), as you might following syntax:

```
value >> shift
```

Here *`value`* is the integer value to be shift For example, the following shifts all the bits i

```
17 >> 2
```

For unsigned integers, the vacated places a end are discarded. For signed integers, vacated the value of the original leftmost bit. The cho (Figure E.2 shows an example that illustrates

The right-shift operator. Shifting one place sion by 2. In general, shifting n places to the r

The Logical Bitwise Operators

The logical bitwise operators are analogous t
apply to a value on a bit-by-bit basis rather th
regular negation operator (`!`) and the bitwise
`!` operator converts a `true` (or nonzero) valu
operator converts each individual bit to its op
sider the `unsigned char` value of `3`:

```
unsigned char x = 3;
```

The expression `!x` has the value `0`. To see
`00000011`. Then you convert each `0` to `1` and
`11111100`, which in base 10 is the value `252`.
new value is termed the *complement* of the or

The bitwise OR operator (`|`) combines tw
value. Each bit in the new value is set to `1` if
ding bits in the original values is set to `1`. If b
bit is set to `0` (see Figure E.4).

bit in b is 1 bits in a and b are 0

Figure E.4 The b

Table E.1 summarizes how the | operator

Table E.1 **The Value of `b1 | b2`**

Bit Values	`b1 = 0`	`b1`
`b2 = 0`	`0`	`1`
`b2 = 1`	`1`	`1`

The `|=` operator combines the bitwise OR

```
a |= b;  // set a to a | b
```

The bitwise XOR operator (^) combines value. Each bit in the new value is set to `1` if o sponding bits in the original values is set to `1`. `1`, the final bit is set to `0` (see Figure E.5).

Table E.2 summarizes how the ^ operator

The ^= operator combines the bitwise XO

```
a ^= b;  // set a to a ^ b
```

The bitwise AND operator (&) combines
value. Each bit in the new value is set to 1 or
original values are set to 1. If either or both o
0 (see Figure E.6).

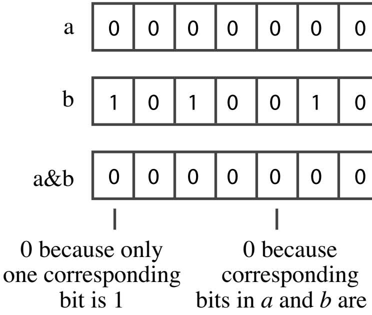


Figure E.6 The b

&	bitand
&=	and_eq
|	bitor
|=	or_eq
~	compl
^	xor
^=	xor_eq

These alternative forms let you write state

```
b = compl a bitand b;  // same as b = ~a &
c = a xor b;           // same as c = a ^
```

A Few Common Bitwise Operator

Often controlling hardware involves turning p
status. The bitwise operators provide the mean
the methods quickly.

In the following examples, `lottabits` repr
the value corresponding to a particular bit. Bi
ning with bit 0, so the value corresponding to

XORing 1 with 0 produces 1, turning an
0, turning an on bit off. All other bits in lott
XORing 0 with 0 produces 0, and XORing

Turning a Bit Off

The following operation turns off the bit in
sented by bit:

```
lottabits = lottabits & ~bit;
```

These statements turn the bit off, regardle
produces an integer with all its bits set to 1 *e*
bit becomes 0. ANDing a 0 with any bit resu
bits in lottabits are unchanged. That's beca
value that bit had before.

Here's a briefer way of doing the same thi

```
lottabits &= ~bit;
```

Testing a Bit Value

Suppose you want to determine whether the
lottabits. The following test does not nece

```
if (lottabits == bit)              // no g
```

```
    int feet;
    int inches;
public:
    Example();
    Example(int ft);
    ~Example();
    void show_in() const;
    void show_ft() const;
    void use_ptr() const;
};
```

Consider the `inches` member. Without a s
class defines `inches` as a member identifier, b
have memory allocated:

```
Example ob;  // now ob.inches exists
```

Thus, you specify an actual memory locati
junction with a specific object. (In a member
object, but then the object is understood to b

C++ lets you define a member pointer to

```
int Example::*pt = &Example::inches;
```

This pointer is a bit different from a regula
cific memory location. But the `pt` pointer do

```
pt = &Example::feet;      // reset pt
cout << ob1.*pt << endl;  // display feet
```

In essence, the combination *pt takes the
identify different member names (of the sam

You can also use member pointers to ider
relatively involved. Recall that declaring a po
no arguments looks like this:

```
void (*pf)();  // pf points to a function
```

Declaring a pointer to a member function
particular class. Here, for instance, is how to d

```
void (Example::*pf)() const;  // pf point
```

This indicates that pf can be used the sam
Note that the term Example::*pf has to be
particular member function to this pointer:

```
pf = &Example::show_inches;
```

Note that unlike in the case of ordinary f
must use the address operator. Having made
to invoke the member function:

```
Example ob3(20);
(ob3.*pf)();     // invoke show_inches()
```

```
    Example(int ft);
    ~Example();
    void show_in() const;
    void show_ft() const;
    void use_ptr() const;
};

Example::Example()
{
    feet = 0;
    inches = 0;
}

Example::Example(int ft)
{
    feet = ft;
    inches = 12 * feet;
}

Example::~Example()
{
}
```

```
    pf = &Example::show_in;
    cout << "Set pf to &Example::show_in:
    cout << "Using (this->*pf)(): ";
    (this->*pf)();
    cout << "Using (yard.*pf)(): ";
    (yard.*pf)();
}

int main()
{
    Example car(15);
    Example van(20);
    Example garage;

    cout << "car.use_ptr() output:\n";
    car.use_ptr();
    cout << "\nvan.use_ptr() output:\n";
    van.use_ptr();

    return 0;
}
```

This example assigned pointer values at co
you can use member pointers to data membe
associated with the pointer is determined at r

alignof (C++11)

Computer systems can have restrictions on ho
one system might require that a `double` value
location, whereas another might require the s
ple of eight. The `alignof` operator takes a typ
indicating the required alignment type. Align
mine how information is arranged within a s

Listing E.2 **align.cpp**

```
// align.cpp -- checking alignment
#include <iostream>
using namespace std;
struct things1
{
    char ch;
    int a;
    double x;
```

Here is the output on one system:

```
char alignment: 1
int alignment: 4
double alignment: 8
things1 alignment: 8
things2 alignment: 8
things1 size: 16
things2 size: 24
```

Both structures have an alignment require
structure size should be a multiple of eight so
each element adjacent to the next and also st
multiple of eight. The individual members of
of 13 bits, but the requirement of using a mu
needs some padding in it. Next, within each
on a multiple of eight. The different arranger
`things2` results in `things2` needing more in
out right.

class that defines necessary properties that a t
string. For example, it should have a `length(`
represented as an array of type `charT`. The en
`charT(0)`, the generalization of the null char
cast of `0` to type `charT`. It could be just `0`, as
be an object created by a `charT` constructor.)
ing values, and so on. The `Allocator` parame
cation for the string. The default `allocator<`
standard ways.

There are four predefined specializations:

```
typedef basic_string<char> string;
typedef basic_string<char16_t> u16string;
typedef basic_string<char32_t> u32string;
typedef basic_string<wchar_t> wstring;
```

These specializations, in turn, use the follo

```
char_traits<char>
allocator<char>
char_traits<char16_t>
allocator<char_16>
char_traits<char_32>
allocator<char_32>
char_traits<wchar_t>
allocator<wchar_t>
```

```
typedef typename traits::char_type
```

The keyword `typename` is used to tell the `traits::char_type` is a type. For the `strin` is `char`.

`size_type` is used like `size_of`, except th the stored type. For the `string` specialization case `size_type` is the same as `size_of`. It is a

`difference_type` is used to measure the c again in units corresponding to the size of a s version of the type underlying `size_type`.

For the `char` specialization, `pointer` is typ However, if you create a specialization for a ty (`pointer` and `reference`) could refer to class basic pointers and references.

To allow Standard Template Library (STL) plate defines some iterator types:

```
typedef (models random access iterator)
typedef (models random access iterator)
typedef std::reverse_iterator<iterator>
typedef std::reverse_iterator<const_iterat
```

`cbegin()`	A `const_ite`
`end()`	An iterator tha
`cend()`	A `const_ite`
`rbegin()`	A reverse itera
`crbegin()`	A reverse `con`
`rend()`	A reverse itera
`crend()`	A reverse `con` (C++11).
`size()`	The number o from `begin()`
`length()`	The same as
`capacity()`	The allocated greater than th `capacity() -` that can be ap be allocated.
`max_size()`	The maximum
`data()`	A pointer of ty ment of an arr responding ele pointer should object itself ha

methods that return iterators with the iterator
you can use the STL reverse() function to r

```
string word;
cin >> word;
reverse(word.begin(), word.end());
```

The data() and c_str() methods, on the
Furthermore, the returned pointers point to t
string characters. This array can but need not
the string object. (The internal representatio
array, but it doesn't have to be.) Because it is p
the original data, they are const, so they can't
are not guaranteed to be valid after the string
to the original data. The difference between d
c_str() points to is terminated with a null ch
guarantees that the actual string characters are
used, for example, as an argument to a functio

```
string file("tofu.man");
ofstream outFile(file.c_str());
```

Similarly, data() and size() could be use
pointer to an array element and a value that r
process:

```
    const Allocator& a = Allocator());
basic_string(initializer_list<charT>, con
~basic_string();
```

Some of the increase in constructors com
example, C++98 had this copy constructor:

```
basic_string(const basic_string& str, siz
    size_type n = npos, const Allocator&
```

C++11 replaces it with three constructor
preceding list. This allows the most common
more efficiently. The really new additions are
references, as discussed in Chapter 18, "Visiti
constructor with the `initializer_list` par

Note that most of the constructors have a

```
const Allocator& a = Allocator()
```

Recall that the term `Allocator` is the ten
class to manage memory. The term `Allocato`
Thus, the constructors, by default, use the de
they give you the option of using some othe
lowing sections examine the constructors inc

charT values:

```
basic_string(const charT* s, const Allocat
```

To determine how many characters to cop
traits::length() method to the array poin
null pointer.) For example, the following state
indicated character string:

```
string toast("Here's looking at you, kid."
```

The traits::length() method for type
how many characters to copy.

The following relationships hold after the

- The data() method returns a pointer t
- The size() method returns a value eq
- The capacity() method returns a valu

Constructors That Use Part of a C

Constructors that use part of a C-style string
of a C-style string; more generally, they let yo
of an array of charT values:

```
basic_string(const charT* s, size_type n,
```

```
string ida(mel);
```

Here, ida would get a copy of the string
The next constructor additionally require

```
basic_string(const basic_string& str, con
```

The following relationships hold after eith

- The data() method returns a pointer element is pointed to by str.data().
- The size() method returns the value
- The capacity() method returns a val

Moving along, the next constructor lets y

```
basic_string(const basic_string& str, siz
    const Allocator& a = Allocator());
```

The second argument pos specifies a loca begin the copying:

```
string att("Telephone home.");
string et(att, 4);
```

Position numbers begin with 0, so positio to "phone home.".

Constructors That Use an Rvalue

C++11 adds move semantics to the `string` c
involves adding a move constructor, which us
reference:

```
basic_string(basic_string&& str) noexcept;
```

This constructor is invoked when the actu

```
string one("din");     // C-style string c
string two(one);       // copy constructor
string three(one+two); // move constructor
```

As discussed in Chapter 18, the intent is th
object constructed by `operator+()` rather tha
original be destroyed.

The second rvalue constructor additionally

```
basic_string(const basic_string&& str, con
```

The following relationships hold after eith

- The `data()` method returns a pointer t
  element is pointed to by `str.data()`.
- The `size()` method returns the value o
- The `capacity()` method returns a valu

The begin iterator points to the element
end points to one past the last location to be

You can use this form with arrays, strings,

```
char cole[40] = "Old King Cole was a merr
string title(cole + 4, cole + 8);
vector<char> input;
char ch;
while (cin.get(ch) && ch != '\n')
    input.push_back(ch);
string str_input(input.begin(), input.end
```

In the first use, InputIterator is evaluate
InputIterator is evaluated to type vector<

The following relationships hold after the

- The data() method returns a pointer
  copying elements from the range [begi
- The size() method returns the distan
  measured in units equal to the size of
  dereferenced.)
- The capacity() method returns a val

```
void resize(size_type n)
```

```
void resize(size_type n, charT c)
```

```
void reserve(size_type res_arg = 0)
```

```
void shrink_to_fit()
```

```
void clear() noexcept
bool empty() const noexcept
```

```
cout << word.at(0);    // display the t
```

The difference (besides the syntax differen bounds checking and throw an out_of_rang pos is type size_type, which is unsigned; th pos. The operator[]() methods don't do bo if pos >= size(), except that the const ver pos == size().

Thus, you get a choice between safety (us cution speed (using array notation).

There is also a function that returns a new

```
basic_string substr(size_type pos = 0, si
```

It returns a string that's a copy of the strin characters or to the end of the string, whiche initializes pet to "donkey":

```
string message("Maybe the donkey will lea
string pet(message.substr(10, 6));
```

C++11 adds these four access methods:

```
const charT& front() const;
charT& front();
const charT& back() const;
charT& back();
```

```
source = "x";
join = name + source;   // now with move s
awkward = {'C','l','o','u','s','e','a','u
```

String Searching

The `string` class provides six search function
sections describe them briefly.

The `find()` Family

Here are the `find()` prototypes as provided b

```
size_type find (const basic_string& str, s
size_type find (const charT* s, size_type
size_type find (const charT* s, size_type
size_type find (charT c, size_type pos = 0
```

The first member returns the beginning po
in the invoking object, with the search beginn
found, the method returns `npos`.

Here's code for finding the location of the

```
string longer("That is a funny hat.");
string shorter("hat");
```

The `rfind()` Family

The `rfind()` methods have these prototypes

```
size_type rfind(const basic_string& str,
                size_type pos = npos) con
size_type rfind(const charT* s, size_type
size_type rfind(const charT* s, size_type
size_type rfind(charT c, size_type pos =
```

These methods work like the analogous f
occurrence of a string or character that starts
not found, the method returns `npos`.

Here's code for finding the location of the
at the end of the longer string:

```
string longer("That is a funny hat.");
string shorter("hat");
size_type loc1 = longer.rfind(shorter);
size_type loc2 = longer.rfind(shorter, lo
```

```
size_type find_last_of (const basic_string
                        size_type pos = np
size_type find_last_of (const charT* s, si
size_type find_last_of (const charT* s, si
size_type find_last_of (charT c, size_type
```

These methods work like the correspondin
looking for a match of the entire substring, th
character in the substring.

Here's code for finding the location of the
the letters in `"fluke"` in a longer string:

```
string longer("That is a funny hat.");
string shorter("hat");
size_type loc1 = longer.find_last_of(short
size_type loc2 = longer.find_last_of("any"
```

The last occurrence of any of the three let
last occurrence of any of the three characters

The `find_first_not_of()` Fan

The `find_first_not_of()` methods have th

```
size_type find_first_not_of(const basic_st
                            size_type pos =
```

```
                              size_type n)
size_type find_last_not_of (const charT*
                              size_type pos
size_type find_last_not_of (charT c, size
```

These methods work like the correspondi
they search for the last occurrence of any cha

Here's code for finding the location of the
not in "This" in a longer string:

```
string longer("That is a funny hat.");
string shorter("That.");
size_type loc1 = longer.find_last_not_of(
size_type loc2 = longer.find_last_not_of(
```

The last space in longer is the last charac
shorter. The f in the longer string is the la
through position 10.

Comparison Methods and

The string class offers methods and functio
the method prototypes:

```
int a13 = s1.compare(s3); // a13 is < 0
int a12 = s1.compare(s2); // a12 is > 0
```

The second method is like the first, except
position `pos1` in the first string for the compa

The following example compares the first

```
string s1("bellflower");
string s2("bell");
int a2 = s1.compare(0, 4, s2); // a2 is 0
```

The third method is like the first, except th
position `pos1` in the first string, and `n2` charac
ond string, for the comparison. For example,
the `out` in `about`:

```
string st1("stout boar");
string st2("mad about ewe");
int a3 = st1.compare(2, 3, st2, 6, 3);  //
```

The fourth method is like the first, except
`string` object for the second string.

The fifth and sixth methods essentially are
ter array instead of a `string` object for the se

append() method. All throw a length_erro
than the maximum string size. The += operat
array, or an individual character to another st

```
basic_string& operator+=(const basic_stri
basic_string& operator+=(const charT* s);
basic_string& operator+=(charT c);
```

The append() methods also let you appen
vidual character to another string. In additio
object by specifying an initial position and a
specifying a range. You can append part of a s
the string to use. The version for appending
instances of that character to copy. Here are t
methods:

```
basic_string& append(const basic_string&
basic_string& append(const basic_string&
                     size_type n);
template<class InputIterator>
  basic_string& append(InputIterator firs
basic_string& append(const charT* s);
basic_string& append(const charT* s, size
basic_string& append(size_type n, charT c
void push_back(charT c);
```

ods, which allow you to assign a whole string
cal characters to a `string` object. Here are th
methods:

```
basic_string& assign(const basic_string& s
basic_string& assign(basic_string&& str) n
basic_string& assign(const basic_string& s
                     size_type n);
basic_string& assign(const charT* s, size_
basic_string& assign(const charT* s);
basic_string& assign(size_type n, charT c)
template<class InputIterator>
  basic_string& assign(InputIterator first
basic_string& assign(initializer_list<char
```

Here are a couple examples:

```
string test;
string stuff("set tubs clones ducks");
test.assign(stuff, 1, 5);   // test is "et
test.assign(6, '#");        // test is "##
```

```
iterator insert(const_iterator p, charT c
iterator insert(const_iterator p, size_ty
template<class InputIterator>
  void insert(iterator p, InputIterator f
iterator insert(const_iterator p, initial
```

For example, the following code inserts th
`banker.":`

```
string st3("The banker.");
st3.insert(4, "former ");
```

Then the following code inserts the string
would be the ninth character) just before the
`banker.":`

```
st3.insert(st3.size() - 1, " waltzed!", 8
```

Erase Methods

The `erase()` methods remove characters fro

```
basic_string& erase(size_type pos = 0, si
iterator erase(const_iterator position);
iterator erase(const_iterator first, iter
void pop_back();
```

```
basic_string& replace(size_type pos, size_
                      size_type n2);
basic_string& replace(size_type pos, size_
basic_string& replace(size_type pos, size_
basic_string& replace(const_iterator i1, c
                      const basic_string&
basic_string& replace(const_iterator i1, c
                      const charT* s, size
basic_string& replace(const_iterator i1, c
                      const charT* s);
basic_string& replace(const_iterator i1, c
                      size_type n, charT c
template<class InputIterator>
  basic_string& replace(const_iterator i1,
                        InputIterator j1,
basic_string& replace(const_iteraor i1, co
                      initializer)list<cha
```

Here is an example:

```
string test("Take a right turn at Main Str
test.replace(7,5,"left"); // replace right
```

Caution

The `copy()` method does not append a null
nation array is large enough.

The `swap()` method swaps the contents o
time algorithm:

```
void swap(basic_string& str);
```

Output and Input

The `string` class overloads the `<<` operator t
ence to the `istream` object so that output ca

```
string claim("The string class has many f
cout << claim << endl;
```

The `string` class overloads the `>>` operato

```
string who;
cin >> who;
```

Input terminates on the end-of-file, on re
allowed in a string, or on reaching a white-sp
depends on the character set and on the type

chapter assumes that you know about iterato

The STL and C++11

Just as the changes brought by C++11 to the completely in this book, so are the changes t in this appendix. However, we can summariz

C++11 brings several new elements to th Second, it adds a few new features to the old functions to its family of algorithms. All these dix, but you may find an overview of the firs

New Containers

C++11 adds the following containers: array unordered associative containers unordered_ unordered_multimap.

An array container, once declared, is fixe rather than dynamically allocated memory. It array type; it's more limited than vector, but

The list container is a bidirectional link ends, linked to the item before it and the one emplacement is a means to increase efficiency

```
class Items
{
    double x;
    double y;
    int m;
public:
    Items();                                     /
    Items (double xx, double yy, int mm);
    ...
};
...
vector<Items> vt(10);
...
vt.push_back(Items(8.2, 2.8, 3));  //
```

The call to `insert()` causes the memory a
object at the end of `vt`. Next, the `Items()` co
this object is copied to a location at the front
object is deleted. With C++11, you can do th

```
vi.emplace_back(8.2, 2.8, 3);
```

`X::value_type`	`T`, the element ty
`X::reference`	`T &`.
`X::const_reference`	`const T &`.
`X::iterator`	Iterator type poin
`X::const_iterator`	Iterator type poin
`X::difference_type`	Signed integral ty iterator to anothe pointers).
`X::size_type`	Unsigned integra objects, number

The class definition uses a `typedef` to def
to declare suitable variables. For example, the
replace the first occurrence of `"bonus"` in a v
order to show how you can use member typ

```
using namespace std;
vector<string> input;
string temp;
while (cin >> temp && temp != "quit")
    input.push_back(temp);
vector<string>::iterator want=
```

the smallest item in the container. This assum
value type used to instantiate the template an
`min_element()` algorithm, which uses an iter
be `vector<int>` or `list<string>` or `deque<`
plate parameter, such as `Bag`, to represent the
that might be instantiated as `vector<int>`, `li`
So the argument type for the function is `cons`
should be the value type for the container—t
point, `Bag` is just a template parameter, and th
`value_type` member is actually a type. But y
that a class member is a `typedef`:

```
vector<string>::value_type st;    // vecto
typename Bag::value_type m;       // Bag a
```

For the first definition here, the compiler h
which states that `value_type` is a `typedef`. Fo
word promises that, whatever `Bag` may turn o
is the name of a type. These considerations le

```
template<typename Bag>
typename Bag::value_type min(const Bag & b
{
    typename Bag::const_iterator it;
    typename Bag::value_type m = *b.begin
```

Operation	Description
`X u;`	Constructs an empty obje
`X()`	Constructs an empty obje
`X(a)`	Constructs a copy of obje
`X u(a)`	`u` is a copy of `a` (copy cor
`X u = a;`	`u` is a copy of `a` (copy cor
`r = a`	`r` equals the value of `a` (
`X u(rv)`	`u` equals the value that `r`
`X u = rv;`	`u` equals the value that `r`
`a = rv`	`a` equals the value that `r`
`(&a)->~X()`	Destructor applied to eve
`begin()`	Returns an iterator to the
`end()`	Returns an iterator to pas
`cbegin()`	Returns a `const` iterator
`cend()`	Returns a `const` iterator
`size()`	Returns the number of el
`maxsize()`	Returns the size of the la
`empty()`	Returns `true` if the conta

a.crend()	Returns a

The unordered set and unordered map con
optional container operations in Table G.4, bu

Table G.4 **Optional Container Operations**

Operation	Description
<	a < b returns true if a lexicogra
>	a > b returns b < a.
<=	a <= b returns !(a > b).
>=	a >= b returns !(a < b).

The > operator for a container assumes tha
type. A lexicographic comparison is a generali
two containers, element-by-element, until it e
doesn't equal the corresponding element in th
ers are considered to be in the same order as t
example, if two containers are identical throu
ment in the first container is less than the 11t

valid dereferenceable `const` iterator to `a`. [q1
an integer of `X::size_type`. `Args` is a templa
parameter pack with the pattern `Args&&`.

Table G.5 **Additional Operations Defined fo**

Operation	Description
`X(n, t)`	Constructs a sequ
`X a(n, t)`	Constructs a sequ
`X(i, j)`	Constructs a sequ
`X a(i, j)`	Constructs a sequ
`X(il);`	Constructs a sequ
`a = il;`	Copies the values
`a.emplace(p, args);`	Inserts an object the arguments pa
`a.insert(p,t)`	Inserts a copy of inserted copy of t value used for typ
`a.insert(p,rv)`	Inserts a copy of inserted copy of t

	ment that had follo
`a.erase(q1,q2)`	Erases the elemen pointing to the ele
`a.clear()`	Does the same thi
`a.front()`	Returns `*a.begin`

Table G.6 lists methods common to some forward_list, list, and deque).

Table G.6 **Operations Defined for Some Seq**

Operation	Description
`a.back()`	Returns `*—a.end()`
`a.push_back(t)`	Inserts t before a.e
`a.push_back(rv)`	Inserts t before a.e move semantics.
`a.pop_back()`	Erases the last elem

The `vector` template additionally has the
container and `n` is an integer of `X::size_typ`

Table G.7 **Additional Operations for Vectors**

Operation	Description
`a.capacity()`	Returns the total nu requiring reallocatio
`a.reserve(n)`	Alerts object `a` that the method call, the Reallocation occurs greater than `a.max_` exception.

The `list` template additionally has the m
containers, and `T` is the type stored in the list
are input iterators, `q2` and `p` are iterators, `q` ar
integer of `X::size_type`. The table uses the
range from `i` up to, but not including, `j`.

`a.merge(b)`	Merges the tor defined lent to an e List `b` is em
`a.merge(b, Compare comp)`	Merges the function or to an eleme `b` is empty
`a.sort()`	Sorts list `a`
`a.sort(Compare comp)`	Sorts list `a`
`a.reverse()`	Reverses th

The `forward-list` operations are similar.
class iterator can't go backwards, some metho
`erase()`, and `splice()` methods are replaced
and `splice_after()` methods, all of which o
position rather than preceding it.

Associative containers provide the methoc
parison object need not require that values w
equivalent keys means that two values, which r
In the table, `X` is a container class, and `a` is an
is, is `set` or `map`), `a_uniq` is an object of type
or `multimap`), `a_eq` is an object of type `X`. As
to elements of `value_type`, [`i`, `j`) is a valid ra
dereferenceable iterators to `a`, [`q1`, `q2`) is a val
(which may be a pair), and `k` is a value of `X:`
`initializer_list<value_type>` object.

	component points
	key of `t`.
`a_eq.insert(t)`	Inserts `t` and retu
`a.insert(p,t)`	Inserts `t`, using `p` search. If `a` is a c if and only if `a` do key; otherwise, ins takes place, the m location with an e
`a.insert(i,j)`	Inserts elements
`a.insert(il)`	Inserts elements
`a_uniq.emplace(args)`	Like `a_uniq.ins` parameter list ma pack.
`a_eq.emplace(args)`	Like `a_eq.inser` eter list matches
`a.emplace_hint(args)`	Like `a.insert(p` ter list matches th
`a.erase(k)`	Erases all elemen returns the numb

Unordered Associative C

As mentioned earlier, the unordered associati
`unordered_multiset`, `unordered_map`, and `u`
tables to provide rapid access to data. Let's tal
function converts a key to an index value. For
function could sum the numeric codes for th
modulus 13, thus giving an index in the rang
13 *buckets* to store `strings`. Any `string` with
bucket 4. If you wished to search the contain
tion to the key and just search the bucket wi
would have enough buckets that each one w

The C++11 library provides a `hash<Key>`
containers use by default. Specializations are
point types, for pointers, and for some templ

Table G.11 lists types used for these conta

The interface for the unordered associativ
tive containers. In particular, Table G.10 also
ers with the following exceptions: The `lowe`
not required, nor is the `X(i,j,c)` constructo
tainers are ordered allows them to use a com

container class, a is an object of type X, b is a
an object of type unordered_set or unorder
unordered_multiset or unordered_multima
of type key_equal, n is a value of type size_
before, i and j are input iterators referring to
range, p and q2 are iterators to a, q and q1 are
valid range, t is a value of X::value_type (w
X::key_type. Also il is an initializer_lis

Table G.12 **Additional Operations Defined fo**
Multimaps

Operation	Description
X(n,hf,eq)	Constructs a using hf as predicate. If key equality used as the
X a(n,hf,eq)	Constructs a using hf as predicate. If key equality used as the

`b.max_bucket_count()`	Returns an contain.
`b.bucket(k)`	Returns the with a key e
`b.bucket_size(n)`	Returns the index `n`.
`b.begin(n)`	Returns an index `n`.
`b.end(n)`	Returns an bucket havi
`b.cbegin(n)`	Returns a c bucket havi
`b.cbegin(n)`	Returns a c the bucket
`b.load_factor()`	Returns the
`b.max_load_factor()`	Returns the tor; the con load factor
`b.max_load_factor(z)`	May change

related operators, and numeric operations. (C
STL to the numeric library, but that doesn't a
operation indicates that the function takes a pa
or sequence, to be operated on. The term *mut*
the container.

Nonmodifying Sequence Operatio

Table G.13 summarizes the nonmodifying se
shown, and overloaded functions are listed jus
prototypes, follows the table. Thus, you can sc
tion does and then look up the details if you

`for_each()`	Applies a nonmodifying
`find()`	Finds the first occurren
`find_if()`	Finds the first value th
`find_if_not()`	Finds the first value th range. (C++11)
`find_end()`	Finds the last occurren values of a second sec applying a binary predi
`find_first_of()`	Finds the first occurren matches a value in the may be evaluated with
`adjacent_find()`	Finds the first element ing it. Matching may be predicate.
`count()`	Returns the number of
`count_if()`	Returns the number of with a match determin

all_of() (C++11)

```
template<class InputIterator, class Predic
bool all_of(InputIterator first, InputIter
            Predicate pred);
```

The `all_of()` function returns `true` if `pred(`
`[first,last)` or if the range is empty. Other

any_of() (C++11)

```
template<class InputIterator, class Predic
bool any_of(InputIterator first, InputIter
            Predicate pred);
```

The `any_of()` function returns `false` if `pred`
`[first,last)` or if the range is empty. Other

none_of() (C++11)

```
template<class InputIterator, class Predic
bool none_of(InputIterator first, InputIte
             Predicate pred);
```

The `none_of()` function returns `true` if `pred`
`[first,last)` or if the range is empty. Other

last) for which the function object can pre
not found.

find_if_not()

```
template<class InputIterator, class Predi
InputIterator find_if_not(InputIterator f
                      Predicate pred);
```

The find_if_not() function returns an iter
[first, last) for which the function object
item is not found.

find_end()

```
template<class ForwardIterator1, class Fo
ForwardIterator1 find_end(
         ForwardIterator1 first1, Forward
         ForwardIterator2 first2, Forward

template<class ForwardIterator1, class Fo
    class BinaryPredicate>
ForwardIterator1 find_end(
         ForwardIterator1 first1, Forward
         ForwardIterator2 first2, Forward
         BinaryPredicate pred);
```

uses the `==` operator for the value type to cor
binary predicate function object `pred` to com
by `it1` and `it2` match if `pred(*it1, *it2)`
not found.

adjacent_find()

```
template<class ForwardIterator>
ForwardIterator adjacent_find(ForwardItera
        ForwardIterator last);

template<class ForwardIterator, class Bina
ForwardIterator adjacent_find(ForwardItera
        ForwardIterator last, BinaryPredi
```

The `adjacent_find()` function returns an it
[`first1`, `last1`) such that the element matche
returns `last` if no such pair is found. The first
type to compare elements. The second versior
`pred` to compare elements. That is, elements p
`pred(*it1, *it2)` is `true`.

```
   mismatch(InputIterator1 first1,
            InputIterator1 last1, InputIt

template<class InputIterator1, class Inpu
                                class Bina
  pair<InputIterator1, InputIterator2>
    mismatch(InputIterator1 first1,
             InputIterator1 last1, InputI
             BinaryPredicate pred);
```

Each of the `mismatch()` functions finds the f
that doesn't match the corresponding elemen
returns a pair holding iterators to the two mi
found, the return value is `pair<last1, firs`
uses the `==` operator to test matching. The se
tion object `pred` to compare elements. That i
match if `pred(*it1, *it2)` is `false`.

`equal()`

```
template<class InputIterator1, class Inpu
bool equal(InputIterator1 first1, InputIt
           InputIterator2 first2);
```

last1) matches the corresponding element in
sequence beginning at first2; it returns fal
operator for the value type to compare eleme
predicate function object pred to compare el
and it2 match if pred(*it1, *it2) is true.

search()

```
template<class ForwardIterator1, class For
ForwardIterator1 search(
        ForwardIterator1 first1, ForwardI
        ForwardIterator2 first2, ForwardI

template<class ForwardIterator1, class For
        class BinaryPredicate>
ForwardIterator1 search(
        ForwardIterator1 first1, ForwardI
        ForwardIterator2 first2, ForwardI
        BinaryPredicate pred);
```

The search() function finds the first occurre
matches the corresponding sequence found in
last1 if no such sequence is found. The first
type to compare elements. The second version

overloaded functions are listed just once. A fu
follows the table. Thus, you can scan the table
then look up the details if you find the funct

Now let's take a more detailed look at the
function, the discussion shows the prototype
saw earlier, pairs of iterators indicate ranges,
indicating the type of iterator. As usual, a ran
`first` up to, but not including, `last`. Functi
which can be function pointers or objects fo
Chapter 16, a predicate is a Boolean function
is a Boolean function with two arguments. (T
as they return `0` for `false` and a nonzero val
function object is one that takes a single argu
that takes two arguments.

copy()

```
template<class InputIterator, class Outpu
OutputIterator copy(InputIterator first,
                    OutputIterator result
```

The `copy()` function copies the elements in
[`result, result + (last - first)`). It retu

`swap()`	Exchanges two values
`swap_ranges()`	Exchanges correspondi
`iter_swap()`	Exchanges two values
`transform()`	Applies a function obje elements in a pair of ra sponding location of ar
`replace()`	Replaces each occurre
`replace_if()`	Replaces each occurre predicate function obje
`replace_copy()`	Copies one range to ar fied value with another
`replace_copy_if()`	Copies one range to ar icate function object is
`fill()`	Sets each value in a ra
`fill_n()`	Sets `n` consecutive ele
`generate()`	Sets each value in a ra a function object that t
`generate_n()`	Sets the first `n` values which is a function obje

rotate()	Treats a range as a c
rotate_copy()	Copies one range to a
random_shuffle()	Randomly rearranges
shuffle()	Randomly rearranges type satisfying C++11 (C++11)
is_partitioned()	Returns true if a rang
partition()	Places all the element all elements that don'
stable_partition()	Places all the element all elements that don' preserved.
partition_copy()	Copies all the elemen output range and the (C++11)
partition_point()	For a range partitioned first element not satis

copy_backward()

```
template<class BidirectionalIterator1,
         class BidirectionalIterator2>
BidirectionalIterator2 copy_backward(Bidi

BidirectionalIterator1 last, Bidirectional
```

The copy_backward() function copies the el
range [result -(last - first), result). C
being copied to location result - 1 and pro
returns result - (last - first)—that is,
to location. The function requires that result
ever, because copying is done backward, it is p

move() (C++11)

```
template<class InputIterator, class Output
OutputIterator copy(InputIterator first, I
                    OutputIterator result)
```

The move() function uses std::move() to m
into the range [result, result + (last - f
first)—that is, an iterator pointing one past

The swap() function exchanges values stored
(C++ moves this function to the utility he

swap_ranges()

```
template<class ForwardIterator1, class Fo
ForwardIterator2 swap_ranges(
                          ForwardIterator
                          ForwardIterator
```

The swap_ranges() function exchanges valu
responding values in the range beginning at f

iter_swap()

```
template<class ForwardIterator1, class Fo
void iter_swap(ForwardIterator1 a, Forwar
```

The iter_swap() function exchanges values

transform()

```
template<class InputIterator, class Outpu
OutputIterator transform(InputIterator fi
OutputIterator result, UnaryOperation op)
```

[first, last) with the value new_value.

replace_if()

```
template<class ForwardIterator, class Pred
void replace_if(ForwardIterator first, For
                Predicate pred, const T& n
```

The replace()_if function replaces each val which pred(old) is true with the value new_

replace_copy()

```
template<class InputIterator, class Output
OutputIterator replace_copy(InputIterator
     OutputIterator result,const T& old_ v
```

The replace_copy() function copies the ele beginning at result but substituting new_val returns result + (last - first), the past-

replace_copy_if()

```
template<class Iterator, class OutputItera
OutputIterator replace_copy_if(Iterator fi
    OutputIterator result, Predicate pred,
```

The generate() function sets each element
gen is a generator function object—that is, o
can be a pointer to rand().

generate_n()

```
template<class OutputIterator, class Size
void generate_n(OutputIterator first, Siz
```

The generate_n() function sets each of the
first to gen(), where gen is a generator fun
arguments. For example, gen can be a pointe

remove()

```
template<class ForwardIterator, class T>
ForwardIterator remove(ForwardIterator fi
                       const T& value);
```

The remove() function removes all occurren
and returns a past-the-end iterator for the res
that the order of the unremoved elements is

The `remove_copy()` function copies values fr
beginning at `result`, skipping instances of `va`
iterator for the resulting range. The function i
moved elements is unaltered.

remove_copy_if()

```
template<class InputIterator, class Output
OutputIterator remove_copy_if(InputIterato
                              OutputIterat
```

The `remove_copy_if()` function copies value
beginning at `result`, skipping instances of `va`
It returns a past-the-end iterator for the result
that the order of the unremoved elements is u

unique()

```
template<class ForwardIterator>
ForwardIterator unique(ForwardIterator fir

template<class ForwardIterator, class Bina
ForwardIterator unique(ForwardIterator fir
                       BinaryPredicate pre
```

reverse()

```
template<class BidirectionalIterator>
void reverse(BidirectionalIterator first,
```

The `reverse()` function reverses the elemen
`swap(first, last - 1)`, and so on.

reverse_copy()

```
template<class BidirectionalIterator, cla
OutputIterator reverse_copy(Bidirectional
                            Bidirectional
                            OutputIterato
```

The `reverse copy()` function copies the ele
range beginning at `result` in reverse order. T

rotate()

```
template<class ForwardIterator>
void rotate(ForwardIterator first, Forwar
            ForwardIterator last);
```

The `rotate()` function performs a left rotatio
`last`). The element at `middle` is moved to `fi`

```
                    RandomNumberGenerator&
```

This version of the `random_shuffle()` functi
[`first`, `last`). The function object `random` de
the expression `random(n)` should return a valu
argument was an lvalue reference; in C++11,

shuffle()

```
template<class RandomAccessIterator, class
void shuffle(RandomAccessIterator first, R
                    UniformRandomNumberGen
```

This version of the `random_shuffle()` functi
[`first`, `last`). The type of the function objec
uniform random number generator as definec
mines the distribution. Given `n` elements, the
the range [`0`,`n`).

is_partitioned() (C++11)

```
template<class InputIterator, class Predic
bool is_partitioned(InputIterator first,
                    InputIterator last, Pr
```

relative ordering within each of the two grou
lowing the last position holding a value for w

`partition_copy()` (C++11)

```
template<class InputIterator, class Outpu
        classs OutputIterator2, class Pr
pair<OutputIterator1, OutputIterator2> pa
     InputIterator first, InputIterator l
     OutputIterator1 out_true, OutputIter
                               Predicate
```

The `partition_copy()` function copies each `pred(val)` is `true` to the range beginning w the range beginning with `out_false`. It retur the end of the range that begins with `out_tr` that begins with `out_false`.

`partition_point()` (C++11)

```
template<class ForwardIterator, class Pre
ForewardIterator partition_point(ForwardI
                                 ForwardI
                                 Predicat
```

	full sort.
`partial_sort_copy()`	Copies a p
`is_sorted()`	Returns `tr`
`is_sorted_until()`	Returns th (C++11)
`nth_element()`	Given an it would be t element th
`lower_bound()`	Given a va before whi the orderir
`upper_bound()`	Given a va before whi the orderir
`equal_range()`	Given a va range such ment in th
`binary_search()`	Returns `tr` equivalent

`sort_heap()`	Sorts a h
`is_heap()`	Returns t
`is_heap_until()`	Returns t (C++11)
`min()`	Returns t an initi
`max()`	Returns t initial
`minmax()`	Returns a ments in largest ite
`min_element()`	Finds the range.
`max_element()`	Finds the
`minmax_element()`	Returns a occurrenc tor to the (C++11)

- If a is equivalent to b and b is equivalen
equivalency is a transitive relationship).

If you think of applying < to integers, then
doesn't have to hold for more general cases. F
with several members describing a mailing ad
that orders the structures according to zip cod
code would be equivalent but not equal.

Now let's take a more detailed look at thes
function, the discussion shows the prototype(s
tion is divided into several subsections. As you
ranges, with the chosen template parameter n
a range in the form [first, last) goes from f
tions passed as arguments are function objects
the () operation is defined. As you learned in
tion with one argument, and a binary predica
ments. (The functions need not be type bool,
nonzero value for true.) Also as in Chapter 1
single argument, and a binary function object

The stable_sort() function sorts the range
of equivalent elements. The first version uses
son object comp to determine the order.

partial_sort()

```
template<class RandomAccessIterator>
void partial_sort(RandomAccessIterator fi
                  RandomAccessIterator la

template<class RandomAccessIterator, clas
void partial_sort(RandomAccessIterator fi
RandomAccessIterator last, Compare comp);
```

The partial_sort() function partially sorts
- first elements of the sorted range are pla
remaining elements are unsorted. The first ve
comparison object comp to determine the or

partial_sort_copy()

```
template<class InputIterator, class Rando
RandomAccessIterator partial_sort_copy(In
                        InputIterator las
```

```
                 Compare comp);
```

The `is_sorted()` function returns `true` if th
otherwise. The first version uses `<`, and the sec
`comp` to determine the order.

`is_sorted_until()` (C++11)

```
template<class ForwardIterator>
ForwardIterator is_sorted_until(ForwardIte

template<class ForwardIterator, class Comp
ForwardIterator is_sorted_until(ForwardIte
               Compare comp);
```

The `is_sorted_until()` function returns `la`
two elements. Otherwise, it returns the last ite
sorted. The first version uses `<`, and the second
to determine the order.

`nth_element()`

```
template<class RandomAccessIterator>
void nth_element(RandomAccessIterator firs
                 RandomAccessIterator last
```

The `lower_bound()` function finds the first p
front of which `value` can be inserted withou
that points to this position. The first version u
parison object `comp` to determine the order.

upper_bound()

```
template<class ForwardIterator, class T>
ForwardIterator upper_bound(ForwardIterat
                            const T& valu

template<class ForwardIterator, class T,
ForwardIterator upper_bound(ForwardIterat
const T& value, Compare comp);
```

The `upper_bound()` function finds the last p
front of which `value` can be inserted withou
that points to this position. The first version u
parison object `comp` to determine the order.

equal_range()

```
template<class ForwardIterator, class T>
pair<ForwardIterator, ForwardIterator> eq
    ForwardIterator first, ForwardIterato
```

> **Note**
>
> Recall that if < is used for ordering, the value
> a are false. For ordinary numbers, equivaler
> structures sorted on the basis of just one me
> tion where a new value can be inserted and s
> parison object comp is used for ordering, equ
> comp(b,a) are false. (This is a generalizati
> lent if a is not less than b and b is not less th

Merging

The merging functions assume that ranges are

merge()

```
template<class InputIterator1, class Input
         class OutputIterator>
OutputIterator merge(InputIterator1 first1
                     InputIterator2 first2
                     OutputIterator result

template<class InputIterator1, class Input
         class OutputIterator,
```

and [middle, last)—into a single sorted seq
Elements from the first range precede equiva
first version uses <, and the second version u
mine the order.

Working with Sets

Set operations work with all sorted sequence
ers that hold more than one instance of a val
ized. A union of two multisets contains the la
element, and an intersection contains the less
For example, suppose Multiset A contains the
contains the same string four times. Then the
instances of "apple", and the intersection wi

includes()

```
template<class InputIterator1, class Inpu
bool includes(InputIterator1 first1, Inpu
              InputIterator2 first2, Inpu

template<class InputIterator1, class Inpu
bool includes(InputIterator1 first1, Inpu
              InputIterator2 first2, Inpu
```

the comparison object comp to determine the

set_intersection()

```
template<class InputIterator1, class Input
        class OutputIterator>
OutputIterator set_intersection(InputItera
      InputIterator1 last1,InputIterator2
      InputIterator2 last2, OutputIterator

template<class InputIterator1, class Input
        class OutputIterator, class Compa
OutputIterator set_intersection(InputItera
      InputIterator1 last1, InputIterator2
      InputIterator2 last2, OutputIterator
      Compare comp);
```

The `set_intersection()` function construct ranges [`first1`, `last1`) and [`first2`, `last2`) a to by `result`. The resulting range should not function returns a past-the-end iterator for th set that contains the elements that are commo the second version uses the comparison objec

```
OutputIterator set_symmetric_difference(
               InputIterator1 first1, Inp
               InputIterator2 first2, Inp
               OutputIterator result);

template<class InputIterator1, class Inpu
         class OutputIterator, class Comp
OutputIterator set_symmetric_difference(
                InputIterator1 first1, In
                InputIterator2 first2, In
                  OutputIterator result,
```

The `set_symmetric_difference()` func difference between the ranges [`first1`, `last` to the location pointed to by `result`. The re original ranges. The function returns a past-t The symmetric difference is the set that cont not in the second and the elements found in as the difference between the union and the second version uses the comparison object c

```
void push_heap(RandomAccessIterator first,
               Compare comp);
```

The `push_heap()` function assumes that the r
it adds the value at location `last - 1` (that is,
to be valid) into the heap, making [`first`, `las`
determine the ordering, and the second versio

`pop_heap()`

```
template<class RandomAccessIterator>
void pop_heap(RandomAccessIterator first,

template<class RandomAccessIterator, class
void pop_heap(RandomAccessIterator first,
              Compare comp);
```

The `pop_heap()` function assumes that the ra
the value at location `last - 1` with the value
`- 1`) a valid heap. The first version uses `<` to d
sion uses the `comp` comparison object.

comp to determine the order.

is_heap_until() (C++11)

```
template<class RandomAccessIterator>
RandomAccessIterator is_heap_until(Random
                                  Rand

template<class RandomAccessIterator, clas
RandomAccessIterator is_heap_until(
            RandomAccessIterator first,
            Compare comp);
```

The `is_heap_until()` function returns las
two elements. Otherwise, it returns the last i
a heap. The first version uses <, and the secon
to determine the order.

Finding Minimum and Maximum Values

The minimum and maximum functions retu
pairs of values and of sequences of values.

```
template<class T, class Compare>
const T& max(const T& a, const T& b, Compa
```

These versions of the `max()` function return t
are equivalent, they return the first value. The
ing, and the second version uses the `comp` con

```
template<class T> T max(initializer_list<T

template<class T, class Compare>
T max(initializer_list<T> t), Compare com
```

These versions of `max()` function (C++11
list `t`. If the two or more values are equivalent
item having that value. The first version uses <
version uses the `comp` comparison object.

`minmax()` (C++11)

```
template<class T>
  pair<const T&,const T&> minmax(const T&

template<class T, class Compare>
  pair<const T&,const T&> minmax(const T&
                              Compare c
```

The min_element() function returns the firs
that no element in the range is less than *it.
ordering, and the second version uses the co

max_element()

```
template<class ForwardIterator>
ForwardIterator max_element(ForwardIterat

template<class ForwardIterator, class Com
ForwardIterator max_element(ForwardIterat
Compare comp);
```

The max_element() function returns the firs
that there is no element that *it is less than.
ordering, and the second version uses the co

minmax_element() (C++11)

```
template<class ForwardIterator>
  pair<ForwardIterator,ForwardIterator>
     minmax_element(ForwardIterator first

template<class ForwardIterator, class Com
  pair<ForwardIterator,ForwardIterator>
    minmax_element(ForwardIterator first,
                Compare comp);
```

to `*first2`. If `*first1` is less than `*first2`, t
than `*first1`, the function returns `false`. If t
to the next element in each sequence. This pr
elements are not equivalent or until the end o
equivalent until the end of one is reached, the
sequences are equivalent and of the same leng
`false`. The first version of the function uses <
sion uses the `comp` comparison object. The le
of an alphabetic comparison.

Working with Permutations

A *permutation* of a sequence is a reordering of
three elements has six possible orderings beca
the first element. Choosing a particular eleme
two for the second, and one for the third. For
1, 2, 3 are as follows:

123 132 213 232 312 321

In general, a sequence of *n* elements has *n*
The permutation functions assume that th
arranged in lexicographic order, as in the prev

```
template<class BidirectionalIterator>
bool prev_permutation(BidirectionalIterat
                      BidirectionalIterat

template<class BidirectionalIterator, cla
bool prev_permutation(BidirectionalIterat
                      BidirectionalIterat
```

The `previous_permutation()` function tra
`last`) to the previous permutation in lexicog
exists, the function returns `true`. If it doesn't
mutation in lexicographic order), the functio
the last permutation in lexicographic order. T
ordering, and the second version uses the co

Numeric Operations

Table G.16 summarizes the numeric operatio
header file. Arguments are not shown, and ov
function has a version that uses `<` for orderin
ison function object for ordering elements. A
follows the table. Thus, you can scan the tabl
then look up the details if you find the funct

```
template <class InputIterator, class T, cl
T accumulate(InputIterator first, InputIte
             BinaryOperation binary_op);
```

The `accumulate()` function initializes a value
`acc = acc + *i` (first version) or `acc = bin`
iterator `i` in the range [`first`, `last`), in order.

inner_product()

```
template <class InputIterator1, class Inpu
T inner_product(InputIterator1 first1, Inp
                InputIterator2 first2, T i

template <class InputIterator1, class Inpu
class BinaryOperation1, class BinaryOperat
T inner_product(InputIterator1 first1, Inp
                InputIterator2 first2, T i
                BinaryOperation1 binary_op
```

The `inner_product()` function initializes a v
ation `acc = *i * *j` (first version) or `acc =`
each iterator `i` in the range [`first1`, `last1`), i

copied over the original sequence, if desired.

adjacent_difference()

```
template <class InputIterator, class Outp
OutputIterator adjacent_difference(InputI
                                   Output

template <class InputIterator, class Outp
OutputIterator adjacent_difference(InputI
                                   Output
                                   Binary
```

The `adjacent_difference()` function assig `= *first`). Subsequent locations in the targe `binary_op` equivalent) of adjacent locations in the target range (`result + 1`) is assigned `binary_op(*(first + 1), *first)` (secon the past-the-end iterator for the result. The a it allows the result to be copied over the orig

Selected Readings

- Becker, Pete. *The C++ Standard Library*
son-Wesley, 2007.

  This book discusses the first C++ Libr
an optional library for C++98, but mo
C++11. Topics include the unordered
expression library, the random number

- Booch, Grady, Robert A. Maksimchuk
Object-Oriented Analysis and Design, Thi
Wesley, 2007.

  This book presents the concepts behin
cusses OOP methods, and presents sam

- Cline, Marshall, Greg Lomow, and Mik
ing, MA: Addison-Wesley, 1998.

  This book addresses a great number of
C++ language.

- Josuttis, Nicolai M. *The C++ Standard*
MA: Addison-Wesley, 1999.

  This book describes the Standard Temp
library features, such as complex numb

- Musser, David R, Gillmer J. Derge, and
C++ Programming with the Standard Tem
Addison-Wesley, 2001.

  The STL merits a complete book to de
such a book.

- Stroustrup, Bjarne. *The C++ Programmi*
Addison-Wesley, 1997.

  Stroustrup created C++, so this is the d
digested if you already have some know
guage, but it also provides many exampl
OOP methodology. Successive editions
and this edition includes a discussion of
and strings.

- Stroustrup, Bjarne. *The Design and Evol*
1994.

  If you're interested in learning how C+
this book.

- Vandevoorde, David and Nocolai M. Jo
Reading, MA: Addison-Wesley, 2003.

  A lot can be said about templates, as thi

Use Alternatives for Som
Directives

The C/C++ preprocessor provides an array
use those directives that are designed to man
using directives as a substitute for code. For e
component for managing program files. Oth
you control whether particular blocks of cod
you control compiler-specific compilation o
sary, tools. You should exercise caution, howe

Use `const` Instead of `#define`

Symbolic constants make code more readabl
cates its meaning, and if you need to change
once, in the definition, and then recompile. C
names for a constant:

```
#define MAX_LENGTH 100
```

The preprocessor then does a text substitu
rences of `MAX_LENGTH` with `100` prior to com
The C++ approach is to apply the `const`

```
const int MAX_LENGTH = 100;
```

}

The preprocessor will replace

```
int n;
```

with

```
int 5;
```

and induce a compilation error. The `dz` de
variable. Also if necessary, `fizzle()` can use th
the constant as `::dz`.

C++ has borrowed the `const` keyword fro
For example, the C++ version has internal lin
the default external linkage used by variables
file in a program using a `const` needs that co
sound like extra work, but, in fact, it makes lif
`const` definitions in a header file used by vari
error for external linkage but not for internal
defined in the file that uses it (being in a head
ment), you can use `const` values as array size

```
const int MAX_LENGTH = 100;
...
double loads[MAX_LENGTH];
```

```
enum {LEVEL1 = 1, LEVEL2 = 2, LEVEL3 = 4,
```

Use inline Instead of #define

The traditional C way to create the near-equ
#define macro definition:

```
#define Cube(X) X*X*X
```

This leads the preprocessor to do text sub
sponding argument to Cube():

```
y = Cube(x);      // replaced with y = x*
y = Cube(x + z++);  // replaced with x +
```

Because the preprocessor uses text substitu
using such macros can lead to unexpected an
by using lots of parentheses in the macro to e

```
#define Cube(X) ((X)*(X)*(X))
```

Even this, however, doesn't deal with case serves as its own prototype.

You should use const in function prototyp
ular, you should use const with pointer param
data that is not to be altered. Not only does th
change data, it also makes a function more ge
pointer or reference can process both const a
fails to use const with a pointer or reference

Use Type Casts

One of Stroustrup's pet peeves about C is its u
casts are often necessary, but the standard type
sider the following code:

```
struct Doof
{
    double feeb;
    double steeb;
    char sgif[10];
};
Doof leam;
short * ps = (short *) & leam;  // old syn
int * pi = int * (&leam);       // new syn
```

If you've been using malloc() and free(), y
instead. If you've been using setjmp() and l
try, throw, and catch instead. You should try
true and false.

Use the New Header Org

The C++ Standard specifies new names for t
"Setting Out to C++." If you've been using
over to using the new-style names. This is no
versions sometimes add new features. For exa
port for wide-character input and output. It
boolalpha and fixed (as described in Chapt
offer a simpler interface than using setf() o
formatting options. If you do use setf(), yo
specifying constants; that is, you should use i
Also the new header files incorporate names

Use Namespaces

Namespaces help organize identifiers used in
Because the standard library, as implemented

expression that uses the operator:

```
cout << std::fixed << x << endl;    //usi
```

This could get wearisome, but you could c
header file:

```
// mynames — a header file
using std::cin;                          // a u
using std::cout;
using std::endl;
```

Going a step further, you could collect usi

```
// mynames — a header file
#include <iostream>

namespace io
{
    using std::cin;
    using std::cout;
    using std::endl;
}

namespace formats
{
```

assignment operator to assign a string to a ch
use `strcpy()` or `strncpy()`. You can't use th
strings; instead, you must remember to use st
> operator, you don't get a syntax error; inste
instead of string contents.)

The `string` class (see Chapter 16, "The s
Library," and Appendix F, "The `string` Temp
objects to represent strings. Assignment opera
operator (for concatenation) are all defined. I
matic memory management so that you norr
entering a string that either overruns an array

The `string` class provides many convenie
one `string` object to another, but you can al
value to a `string` object. For functions that r
use the `c_str()` method to return a suitable

Not only does the `string` class provide a
string-related tasks, such as finding substrings
ble with the Standard Template Library (STL
`string` objects.

your part. And the iterator concept used to in
algorithms aren't limited to being used with S
applied to traditional arrays, too.

final compilation.

3. It makes definitions made in the `std` n

4. `cout << "Hello, world\n";`

   or

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

5. `int cheeses;`

6. `cheeses = 32;`

7. `cin >> cheeses;`

8. `cout << "We have " << cheeses <<`

9. The function `froop()` expects to be ca `double`, and that the function will retu used as follows:

   `int gval = froop(3.14159);`

   The function `rattle()` has no return instance, it could be used as follows:

   `rattle(37);`

Note: Don't count on int being large e[...]
system supports universal list-initializati[...]

```
short rbis = {80};          // = is
unsigned int q {42110};     // could
long long ants {3000000000};
```

3. C++ provides no automatic safeguards [...]
you can use the `climits` header file to [...]

4. The constant `33L` is type `long`, whereas [...]

5. The two statements are not really equiv[...]
some systems. Most importantly, the firs[...]
only on a system using the ASCII code, [...]
other codes. Second, 65 is a type `int` co[...]

6. Here are four ways:

```
char c = 88;
cout << c << endl;          // char

cout.put(char(88));         // put()

cout << char(88) << endl;    // new-

cout << (char)88 << endl;    // old-
```

10. a. int
    b. `float`
    c. `char`
    d. `char32_t`
    e. `double`

Answers to Chapter Revi

1. a. `char actors[30];`
   b. `short betsie[100];`
   c. `float chuck[13];`
   d. `long double dipsea[64];`

2. a. `array<char,30> actors;`
   b. `array<short, 100> betsie;`
   c. `array<float, 13> chuck;`
   d. `array<long double, 64> dips`

```
{
    "trout",
    12,
    26.25
};
```

10. `enum Response {No, Yes, Maybe};`

11.
```
double * pd = &ted;
cout << *pd << "\n";
```

12.
```
float * pf = treacle;   // or = &tre
cout << pf[0] << " " << pf[9] << "\n
           // or use *pf and *(pf +
```

13. This assumes that the `iostream` and ve that there is a `using` directive:
```
unsigned int size;
cout << "Enter a positive integer: "
cin >> size;
int * dyn = new int [size];
vector<int> dv(size);
```

17.
```
#include <string>
#include <vector>
#include <array>
const int Str_num {10};  // or = 10
...
std::vector<std::string> vstr(Str_n
std::array<std::string, Str_num> as
```

Answers to Chapter Revi

1. An entry-condition loop evaluates a te loop. If the condition is initially `false` condition loop evaluates a test expressi Thus, the loop body is executed once, The `for` and `while` loops are entry-co exit-condition loop.

2. It would print the following:

```
01234
```

Note that `cout << endl;` is not part o braces).

to x. The second statement is also valid.
be evaluated as follows:

```
(y = 1), 024;
```

That is, the left expression sets y to 1, ar
isn't used, is 024, or 20.

9. The cin >> ch form skips over spaces,
them. The other two forms read those c

Answers to Chapter Revi

1. Both versions give the same answers, bu
Consider what happens, for example, w
ing spaces, tests whether the character i
program has already established that ch
line. Version 2, in the same situation, ski

2. Both ++ch and ch + 1 have the same n
prints as a character, while ch + 1, beca
prints as a number.

6. `(x < 0)? -x : x`

   or

   `(x >= 0)? x : -x;`

7.
```
switch (ch)
{
    case 'A': a_grade++;
              break;
    case 'B': b_grade++;
              break;
    case 'C': c_grade++;
              break;
    case 'D': d_grade++;
              break;
    default:  f_grade++;
              break;
}
```

8. If you use integer labels and the user ty
hangs because integer input can't proc
and the user types an integer such as **5**
Then the default part of the switch ca

g. `char * plot(map *pmap);`

3.
```
void set_array(int arr[], int size,
{
    for (int i = 0; i < size; i++)
        arr[i] = value;
}
```

4.
```
void set_array(int * begin, int * en
{
    for (int * pt = begin; pt != end
        pt* = value;
}
```

5.
```
double biggest (const double foot[],
{
    double max;
    if (size < 1)
    {
        cout << "Invalid array size
        cout << "Returning a value o
        return 0;
```

```
    while (*str)      // while not
    {
        if (*str == c1)
        {
            *str = c2;
            count++;
        }
        str++;        // advance to
    }
    return count;
}
```

9. Because C++ interprets `"pizza"` as th
operator yields the value of that first el
C++ interprets `"taco"` as the address o
the value of the element two positions
other words, the string constant acts th

10. To pass it by value, you just pass the str
use the address operator `&glitz`. Passin
original data, but it takes time and mem
ory but doesn't protect the original dat
function parameter. Also passing by val

13.
```
typedef void (*p_f1)(applicant *);
p_f1 p1 = f1;
typedef const char * (*p_f2)(const a
p_f2 p2 = f2;
p_f1 ap[5];
p_f2 (*pa)[10];
```

Answers to Chapter Revi

1. Short, nonrecursive functions that can f for inline status.

2. a. `void song(const char * name`
   b. None. Only prototypes contain th
   c. Yes, provided that you retain the
      `void song(char * name = "O, My`

3. You can use either the string "\"" or th The following functions show both me

```
#include <iostream.h>
void iquote(int n)
```

```
      cout << "Volume = " << co
   }
```

b.
```
   void set_volume(box & crate
   {
       crate.volume = crate.heigh
   }
```

5. First, change the prototypes to the foll

```
// function to modify array object
void fill(std::array<double, Season
// function that uses array object
void show(const std::array<double,
```

Note that `show()` should use `const` to
Next, within `main()`, change the `fill`

```
fill(expenses);
```

There's no change to the `show()` call.

b. You can't use a default for the rep
default values from right to left. Y

```
void repeat(int times, const ch
void repeat(const char * str);
```

c. You can use function overloading

```
int average(int a, int b);
double average(double x, double
```

d. You can't do this because both ve

7.
```
template<class T>
T max(T t1, T t2)  // or T max(const
{
    return t1 > t2? t1 : t2;
}
```

8.
```
template<> box max(box b1, box b2)
{
    return b1.volume > b2.volume? b1
}
```

3.
```
#include <iostream>
int main()
{
    double x;
    std::cout << "Enter value: ";
    while (! (std::cin >> x) )
    {
        std::cout << "Bad input. Pl
        std::cin.clear();
       while (std::cin.get() != '\n
           continue;
    }
    std::cout << "Value = " << x <<
    return 0;
}
```

4. Here is the revised code:
```
#include <iostream>
int main()
{
    using std::cin;
    using std::cout;
    using std::endl;
```

```
4, 1, 2
2
2
4, 1, 2
2
```

Answers to Chapter Revi

1. A class is a definition of a user-defined to be stored, and it specifies the method to access and manipulate that data.

2. A class represents the operations you ca interface of class methods; this is abstrac default) for data members, meaning tha member functions; this is data hiding. D representation and method code, are hid

3. A class defines a type, including how it another data object, such as that produc according to the class definition. The re the same as that between a standard typ

```
    void deposit(double cash);
    void withdraw(double cash);
};
```

6. A class constructor is called when you
explicitly call the constructor. A class d

7. These are two possible solutions (note
in order to use `strncpy()` or else you
class):

```
BankAccount::BankAccount(const char
{
    strncpy(name, client, 39);
    name[39] = '\0';
    strncpy(acctnum, num, 24);
    acctnum[24] = '\0';
    balance = bal;
}
```

or

```
BankAccount::BankAccount(const std:
                         const std:
{
    name = client;
```

```
    void set_tot() { total_val = sha
public:
    Stock();              // default c
    Stock(const std::string & co, lo
    ~Stock() {}           // do-nothin
    void buy(long num, double price)
    void sell(long num, double price
    void update(double price);
    void show() const;
    const Stock & topval(const Stock
    int numshares() const { return s
    double shareval() const { return
    double totalval() const { return
    const string & co_name() const {
};
```

10. The `this` pointer is available to class m invoke the method. Thus, `this` is the ac object itself.

access public members.

4. Here's a prototype for the class definiti
   methods file:

```
// prototype
friend Stonewt operator*(double mul

// definition — let constructor do
Stonewt operator*(double mult, cons
{
    return Stonewt(mult * s.pounds)
}
```

5. The following five operators cannot be

```
sizeof
.
.*
::
? :
```

6. These operators must be defined by us

ory allocated by `new` in the constructor
deletes such memory, it might end up t
izes one such object to another. That's b
object to another copies pointer values
this produces two pointers to the same
constructor that causes initialization to
one object to another can produce the
the same data. The solution is to overlo
the data, not the pointers.

3. C++ automatically provides the follow
   - A default constructor if you defin
   - A copy constructor if you don't d
   - An assignment operator if you do
   - A default destructor if you don't
   - An address operator if you don't

   The default constructor does nothing, b
   tialized objects. The default copy constr
   use memberwise assignment. The defau
   address operator returns the address of t
   `this` pointer).

```
    talents = 0;
}

nifty::nifty(const char * s)
{
    strcpy(personality, s);
    talents = 0;
}

ostream & operator<<(ostream & os, c
{
    os << n.personality << '\n';
    os << n.talent << '\n';
    return os;
}
```

Here is another possible solution:

```
#include <iostream>
#include <cstring>
using namespace std;
class nifty
{
private: // optional
    char * personality;    // create
```

```
}

ostream & operator<<(ostream & os, c
{
    os << n.personality << '\n';
    os << n.talent << '\n';
    return os;
}
```

5. a.
```
Golfer nancy; // default co
Golfer lulu("Little Lulu"); //
Golfer roy("Roy Hobbs", 12); /
Golfer * par = new Golfer; //
Golfer next = lulu; // Golfer(
Golfer hazard = "Weed Thwacker
*par = nancy; // default assig
nancy = "Nancy Putter";// Golf
                  // the defau
```

Note that some compilers additio
for Statements 5 and 6.

b. The class should define an assign
addresses.

4. Constructors are called in the order of
tor called first. Destructors are called in

5. Yes, every class requires its own constru
bers, the constructor can have an empt

6. Only the derived-class method is called
base-class method is called only if the c
if you use the scope-resolution operato
tual any functions that will be redefine

7. The derived class should define an assi
structors use the `new` or `new []` operat
that class. More generally, the derived c
the default assignment is incorrect for

8. Yes, you can assign the address of an ob
base class. You can assign the address of
class (downcasting) only by making an
safe to use such a pointer.

9. Yes, you can assign an object of a deriv
data members that are new to the deri
however. The program uses the base-cl

PublicCorporation::head().

14. First, the situation does not fit the *is-a* r
ate. Second, the definition of area() in
because the two methods have different

Answers to Chapter Revi

1. class Bear	class PolarBe
class Kitchen	class Home
class Person	class Program
class Person	class HorseAndJocke
class Person, class Automobile	class Driver

```
private:
    enum {MAX = 10};          // cons
    Worker * items[MAX];      // hold
    int top;                  // inde
public:
    Stack();
    Boolean isempty();
    Boolean isfull();
    Boolean push(const Worker * & i
    Boolean pop(Worker * & item);
};
```

5. `ArrayTP<string> sa;`
   `StackTP< ArrayTP<double> > stck_arr`
   `ArrayTP< StackTP<Worker *> > arr_st`

   Listing 14.18 generates four templates:
   `ArrayTP<int,5>`, and `Array< ArrayT`

6. If two lines of inheritance for a class sh
   having two copies of the ancestor's me
   base class to its immediate descendants

```
      class cuff {
      public:
          void snip(muff &) { ... }
          ...
      };
      class muff {
          friend void cuff::snip(muf
          ...
};
```

2. No. For Class `A` to have a friend that's a
   tion must precede the `A` declaration. A f
   would tell `A` that `B` is a class, but it woul
   Similarly, if `B` has a friend that's a memb
   must precede the `B` declaration. These t

3. The only access to a class is through its
   thing you can do with a `Sauce` object i
   other members (`soy` and `sugar`) are pri

4. Suppose the function `f1()` calls the fun
   causes program execution to resume at
   tion call in function `f1()`. A `throw` stat

```
class RQ1
{
private:
    string st;       // a string ob
public:
    RQ1() : st("") {}
    RQ1(const char * s) : st(s) {}
    ~RQ1() {};
// more stuff
};
```

The explicit copy constructor, destruct
needed because the string object prov

2. You can assign one string object to a
memory management so that you norm
exceeding the capacity of its holder.

3.
```
#include <string>
#include <cctype>
using namespace std;
void ToUpper(string & str)
```

ordinary arrays as well as with iterators
generality.

9. You can assign one `vector` object to an
so you can insert items into a vector an
the `at()` method, you can get automati

10. The two `sort()` functions and the `ran`
access iterator, whereas a `list` object ju
the list template class `sort()` member f
Methods and Functions") instead of the
ing, but there is no member function e
you could copy the list to a vector, shuf
the list.

Answers to Chapter Revi

1. The `iostream` file defines the classes, c
input and output. These objects manage
The file also creates standard objects (`c`
character equivalents) used to handle th
nected to every program.

then invoke the next method in a sequ

6.
```
//rq17-6.cpp
#include <iostream>
#include <iomanip>

int main()
{
    using namespace std;
    cout << "Enter an integer: ";
    int n;
    cin >> n;
    cout << setw(15) << "base ten" 
           << "base sixteen" << setw
    cout.setf(ios::showbase);    //
    cout << setw(15) << n << hex <<
           << oct << setw(15) << n 

    return 0;
}
```

```
    cout << "First format:\n";
    cout << setw(30) << name << ": $
         << setw(10) << hourly << ":
         << setw(5) << hours << "\n"
    cout << "Second format:\n";
    cout.setf(ios::left, ios::adjust
    cout << setw(30) << name << ": $
         << setw(10) << hourly << ":
         << setw(5) << hours << "\n"

    return 0;
}
```

8. Here is the output:

```
ct1 = 5; ct2 = 9
```

The first part of the program ignores sp
doesn't. Note that the second part of th
character following the first `q`, and it co
total.

9. The `ignore()` form falters if the input
skips only the first 80 characters.

of w+1.

r1(up(w)) is valid, and the argument r
return value of up(w).

In general, if an lvalue is passed to a co
ter is initialized to the lvalue. If an rval
reference parameter refers to a tempora

r2(w) is valid, and the argument rx ref

r2(w+1) is an error because w+1 is an r

r2(up(w)) is an error because the retu

In general, if an lvalue is passed to a no
parameter is initialized to the lvalue. Bu
can't accept an rvalue function argume

r3(w) is an error because an rvalue ref

r3(w+1) is valid, and rx refers to the te

r3(up(w)) is valid, and rx refers to the

3. a. `double & rx`
   `const double & rx`
   `const double & rx`

these functions, depending on the conte

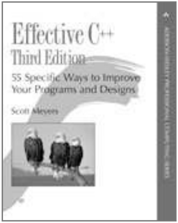
5. A move constructor can be used when instead of copying it, but there is no me standard array. If the `Fizzle` class used a then one can transfer ownership by reas pointer.

6.
```
#include <iostream>
#include <algorithm>
template<typename T>
  void show2(double x, T fp) {std::c
int main()
{
    show2(18.0, [](double x){return
    return 0;
}
```

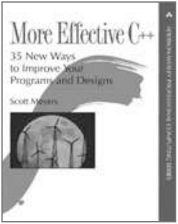
C

I

P

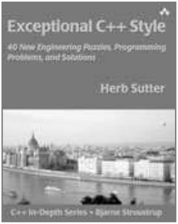


978-0-321-33487-9

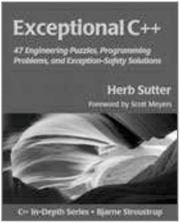


978-0-201-63371-9

978



978-0-201-76042-2



978-0-201-61562-3

978

Addison Wesley
PRENTICE HALL
informIT
Safari Books Online

The **Developer's Library Series** fro
practicing programmers with unique
tutorials on the latest programming langu
use in their daily work. All books in the D
expert technology practitioners who are
and presenting information in a way that

Developer's Library books cover a wide
source programming languages and dat
Microsoft, and Java, to Web developme
Mac/iPhone programming, and Android

Addison-Wesley Cisco Press EXAM/CRAM IBM Press