Instructor's Manual

for

C++ How to Program 3rd

Deitel, Deitel & Nieto

C++ How to Program: Third Edition Instructor’s Manual Contents

Preface

Thank you for considering and/or adopting our text *C++ How to Program: Third Edition*. If you have not read the preface to *C++ How to Program: Third Edition*, please do so. The preface contains a careful walkthrough of the book's key features, including our new Unified Modeling Language ™ (UML™) case study, which carefully introduces the reader to the UML and object-oriented design (OOD). Students are presented with a detailed problem statement and guided through a simplified, UML-based object-oriented design process. The complete 1000-line C++ program solution for the case study is presented in the book and provided on the CD-ROM in the back of the textbook.

We have worked hard to produce a textbook and ancillaries that we hope you and your students will find valuable. The following ancillary resources are available:

- *C++ How to Program: Third Edition's 250 program examples* are included on the CD-ROM in the back of the textbook. This helps instructors prepare lectures faster and helps students master C++. The examples are also available for download at **`www.deitel.com`**. When extracting the source code from the ZIP file, you must use a ZIP-file reader such as WinZip (**`www.winzip.com`**) or PKZIP (**`www.pkware.com`**) that understands directories. The file should be extracted into a separate directory (e.g., **`cpphtp3e_examples`**).
- *Microsoft Visual C++ 6 Introductory Edition software* is provided on the textbook's CD-ROM. This software allows students to edit, compile and debug C++ programs. We have provided at no charge a short Visual C++ 6 tutorial (in Adobe PDF format) on our Web site (**`www.deitel.com`**).
- This *C++ How to Program: Third Edition Instructor's Manual* on CD contains answers to most of the exercises in the textbook. The programs are separated into directories by chapter and exercise number.
- The optional *C++ Multimedia Cyber Classroom: Third Edition* is an interactive multimedia CD version of the book for Windows. Its features include audio walkthroughs of programs, section review questions (which are available only on the *C++ Multimedia Cyber Classroom: Third Edition*), a text-search engine, the ability to execute example programs, and more. The *Cyber Classroom* helps students get more out of their courses. The *Cyber Classroom* is also useful for students who miss a lecture and have to catch up quickly. The *Cyber Classroom* is available as a stand-alone product (see the last few pages of the textbook for the ISBN number) or bundled with the textbook (at a discount) in a product called *The Complete C++ Training Course: Third Edition* (ISBN# 0-13-089563-6).
- *Companion Web site* (**`www.prenhall.com/deitel`**) provides instructor and student resources. Instructor resources include textbook appendices (e.g., Appendix D, "C++ Internet and Web Resources") and a syllabus manager for lesson planning. Student resources include chapter objectives, true/false questions, chapter highlights, reference materials and a message board.
- Customizable *Powerpoint Instructor Lecture Notes*, with many complete features including source code, and key discussion points for each program and major illustration. These lecture notes are available for instructors and students at no charge at our Web site **`www.deitel.com`**.
- *Lab Manual* (available Spring 2001)—a for-sale item containing closed-lab sessions.

We would sincerely appreciate your questions, comments, criticisms and corrections addressed to us at:

`deitel@deitel.com`

We will respond immediately. Please read the latest copy of the *Deitel Buzz* (published every April and November) for information on forthcoming Deitel publications, ancillaries, product options and ordering information. To receive the *Deitel Buzz*, please contact Jennie Burger (**`jennie_burger@prenhall.com`**).

Watch our Deitel & Associates, Inc. Web site (**`www.deitel.com`**) and our Prentice Hall Web site (**`www.prenhall.com/deitel`**) for the latest publication updates.

We would like to thank the extraordinary team of publishing professionals at Prentice Hall who made *C++ How to Program: Third Edition* and its ancillaries possible. Our Computer Science editor, Petra Recter, worked closely with us to ensure the timely availability and professional quality of these ancillaries.

We would also like to thank two of our student interns—Aftab Bukhari (a Computer Science major at Boston University) and Jason Rosenfeld (a Computer Science major at Northwestern University) for their assistance in preparing this Instructor's Manual.

Harvey M. Deitel
Paul J. Deitel

1

Introduction to Computers and C++ Programming Solutions

SOLUTIONS

1.10 Categorize each of the following items as either hardware or software:

a) CPU
ANS: hardware.
b) C++ compiler
ANS: software.
c) ALU
ANS: hardware.
d) C++ preprocessor
ANS: software.
e) input unit
ANS: hardware.
f) an editor program
ANS: software.

1.11 Why might you want to write a program in a machine-independent language instead of a machine-dependent language? Why might a machine-dependent language be more appropriate for writing certain types of programs?

ANS: Machine independent languages are useful for writing programs to be executed on multiple computer platforms. Machine dependent languages are appropriate for writing programs to be executed on a single platform. Machine dependent languages tend to exploit the efficiencies of a particular machine.

1.12 Fill in the blanks in each of the following statements:

a) Which logical unit of the computer receives information from outside the computer for use by the computer? __________.
ANS: input unit.
b) The process of instructing the computer to solve specific problems is called __________.
ANS: computer programming.
c) What type of computer language uses English-like abbreviations for machine language instructions? __________.
ANS: high-level language.
d) Which logical unit of the computer sends information that has already been processed by the computer to various devices so that the information may be used outside the computer? __________.
ANS: output unit.
e) Which logical unit of the computer retains information? __________.
ANS: memory unit and secondary storage unit.
f) Which logical unit of the computer performs calculations? __________.

ANS: arithmetic and logical unit.

g) Which logical unit of the computer makes logical decisions? __________.

ANS: arithmetic and logical unit.

h) The level of computer language most convenient to the programmer for writing programs quickly and easily is __________.

ANS: high-level language.

i) The only language that a computer can directly understand is called that computer's __________.

ANS: machine language.

j) Which logical unit of the computer coordinates the activities of all the other logical units? __________.

ANS: central processing unit.

1.13 Discuss the meaning of each of the following objects:

a) `cin`

ANS: This object refers to the standard input device that is normally connected to the keyboard.

b) `cout`

ANS: This object refers to the standard output device that is normally connected to the computer screen.

c) `cerr`

ANS: This object refers to the standard error device that is normally connected to the computer screen.

1.14 Why is so much attention today focused on object-oriented programming in general and C++ in particular?

ANS: Object-oriented programming enables the programmer to build reusable software components that model items in the real world. Building software quickly, correctly, and economically has been an elusive goal in the software industry. The modular, object-oriented design and implementation approach has been found to increase productivity 10 to 100 times over conventional programming languages while reducing development time, errors, and cost. C++ is extremely popular because it is a superset of the widely used C programming language. Programmers already familiar with C have an easier time learing C++.

1.15 Fill in the blanks in each of the following:

a) __________ are used to document a program and improve its readability.

ANS: comments

b) The object used to print information on the screen is __________.

ANS: `cout`

c) A C++ statement that makes a decision is __________.

ANS: `if`

d) Calculations are normally performed by __________ statements.

ANS: assignment

e) The __________ object inputs values from the keyboard.

ANS: `cin`

1.16 Write a single C++ statement or line that accomplishes each of the following:

a) Print the message `"Enter two numbers"`.

ANS: `cout << "Enter two numbers";`

b) Assign the product of variables `b` and `c` to variable `a`.

ANS: `a = b * c;`

c) State that a program performs a sample payroll calculation (i.e., use text that helps to document a program).

ANS: `// Sample Payroll Calculation Program`

d) Input three integer values from the keyboard and into integer variables `a`, `b` and `c`.

ANS: `cin >> a >> b >> c;`

1.17 State which of the following are *true* and which are *false*. If *false*, explain your answers.

a) C++ operators are evaluated from left to right.

ANS: False. Some operators are evaluated from left to right, while other operators are evaluated right to left.

b) The following are all valid variable names: `_under_bar_`, `m928134`, `t5`, `j7`, `her_sales`, `his_account_total`, `a`, `b`, `c`, `z`, `z2`.

ANS: True. All variables begin with an underscore or letter.

c) The statement `cout << "a = 5;";` is a typical example of an assignment statement.

ANS: False. The statement is an output statement. `a = 5;` is output to the screen.

d) A valid C++ arithmetic expression with no parentheses is evaluated from left to right.

ANS: False. Arithmetic operators can appear in any order in an expression. Since multiplication, division, and modulus have higher precendence than addition and subtraction the statement cannot be true.

e) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22`, `2h`.

ANS: False. `h22` is a valid variable name.

1.18 Fill in the blanks in each of the following:

a) What arithmetic operations are on the same level of precedence as multiplication? __________.

ANS: division and modulus.

b) When parentheses are nested, which set of parentheses is evaluated first in an arithmetic expression? __________.

ANS: innermost.

c) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a __________.

ANS: variable.

1.19 What, if anything, prints when each of the following C++ statements is performed? If nothing prints, then answer “nothing.” Assume `x = 2` and `y = 3`.

a) `cout << x;`

ANS: `2`

b) `cout << x + x;`

ANS: `4`

c) `cout << "x=";`

ANS: `x=`

d) `cout << "x = " << x;`

ANS: `x = 2`

e) `cout << x + y << " = " << y + x;`

ANS: `5 = 5`

f) `z = x + y;`

ANS: nothing.

g) `cin >> x >> y;`

ANS: 23.

h) `// cout << "x + y = " << x + y;`

ANS: nothing.

i) `cout << "\n";`

ANS: A newline is output which positions the cursor at the beginning of the next line on the screen.

1.20 Which of the following C++ statements contain variables whose values are replaced?

a) `cin >> b >> c >> d >> e >> f;`

b) `p = i + j + k + 7;`

c) `cout << "variables whose values are destroyed";`

d) `cout << "a = 5";`

ANS: Parts (a) and (b).

1.21 Given the algebraic equation $y = ax^3 + 7$, which of the following, if any, are correct C++ statements for this equation?

a) `y = a * x * x * x + 7;`

b) `y = a * x * x * ( x + 7 );`

c) `y = ( a * x ) * x * ( x + 7 );`

d) `y = (a * x) * x * x + 7;`

e) `y = a * ( x * x * x ) + 7;`

f) `y = a * x * ( x * x + 7 );`

ANS: Parts (a), (d) and (e).

1.22 State the order of evaluation of the operators in each of the following C++ statements and show the value of `x` after each statement is performed.

a) `x = 7 + 3 * 6 / 2 - 1;`

ANS: `*, /, +, -, =, 15`

b) `x = 2 % 2 + 2 * 2 - 2 / 2;`

ANS: `%, *, /, +, -, =, 3`

c) `x = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) );`

ANS: `*`, `/`, `+`, `*`, `*, 324`

1.23 Write a program that asks the user to enter two numbers, obtains the two numbers from the user and prints the sum, product, difference, and quotient of the two numbers.

```
// Exercise 1.23 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num1, num2;  // declare variables

   cout << "Enter two integers: ";  // prompt user
   cin >> num1 >> num2;             // read values from keyboard

   // output the results
   cout << "The sum is " << num1 + num2
        << "\nThe product is " << num1 * num2
        << "\nThe difference is " << num1 - num2
        << "\nThe quotient is " << num1 / num2 << endl;

   return 0;  // indicate successful termination
}
```

```
Enter two integers: 8 22
The sum is 30
The product is 176
The difference is -14
The quotient is 0
```

1.24 Write a program that prints the numbers 1 to 4 on the same line with each pair of adjacent numbers separated by one space. Write the program using the following methods:

a) Using one output statement with one stream insertion operator.
b) Using one output statement with four stream insertion operators.
c) Using four output statements.

```
// Exercise 1.24 Solution
#include <iostream>

using std::cout;
using std::endl;

int main ()
{
   // Part A
   cout << "1 2 3 4\n";

   // Part B
   cout << "1 " << "2 " << "3 " << "4\n";

   // Part C
   cout << "1 ";
   cout << "2 ";
   cout << "3 ";
   cout << "4" << endl;

   return 0;
}
```

```
1 2 3 4
1 2 3 4
1 2 3 4
```

1.25 Write a program that asks the user to enter two integers, obtains the numbers from the user, then prints the larger number followed by the words "**`is larger`**." If the numbers are equal, print the message "**`These numbers are equal`**."

```
// Exercise 1.25 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num1, num2;   // declaration

   cout << "Enter two integers: ";  // prompt
   cin >> num1 >> num2;             // input to numbers

   if ( num1 == num2 )
      cout << "These numbers are equal." << endl;

   if ( num1 > num2 )
      cout << num1 << " is larger." << endl;

   if ( num2 > num1 )
      cout << num2 << " is larger." << endl;

   return 0;
}
```

```
Enter two integers: 22 8
22 is larger.
```

1.26 Write a program that inputs three integers from the keyboard and prints the sum, average, product, smallest and largest of these numbers. The screen dialogue should appear as follows:

```
Input three different integers: 13 27 14
Sum is 54
Average is 18
Product is 4914
Smallest is 13
Largest is 27
```

```
// Exercise 1.26 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num1, num2, num3, smallest, largest;  // declaration

   cout << "Input three different integers: ";  // prompt
   cin >> num1 >> num2 >> num3;                 // input

   largest = num1;  // assume first number is largest

   if ( num2 > largest )  // is num2 larger?
      largest = num2;

   if ( num3 > largest )  // is num3 larger?
      largest = num3;

   smallest = num1;  // assume first number is smallest

   if ( num2 < smallest )
      smallest = num2;

   if ( num3 < smallest )
      smallest = num3;

   cout << "Sum is " << num1 + num2 + num3
        << "\nAverage is " << (num1 + num2 + num3) / 3
        << "\nProduct is " << num1 * num2 * num3
        << "\nSmallest is " << smallest
        << "\nLargest is " << largest << endl;

   return 0;
}
```

```
Input three different integers: 13 27 14
Sum is 54
Average is 18
Product is 4914
Smallest is 13
Largest is 27
```

1.27 Write a program that reads in the radius of a circle and prints the circle's diameter, circumference and area. Use the constant value 3.14159 for π. Do these calculations in output statements. (Note: In this chapter, we have discussed only integer constants and variables. In Chapter 3 we will discuss floating-point numbers, i.e., values that can have decimal points.)

```
// Exercise 1.27 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int radius;  // declaration

   cout << "Enter the circle radius: ";  // prompt
   cin >> radius;                        // input

   cout << "Diameter is " << radius * 2.0
        << "\nCircumference is " << 2 * 3.14159 * radius
        << "\nArea is " << 3.14159 * radius * radius << endl;

   return 0;
}
```

```
Enter the circle radius: 8
Diameter is 16
Circumference is 50.2654
Area is 201.062
```

1.28 Write a program that prints a box, an oval, an arrow and a diamond as follows:

```
*********          ***              *              *
*       *        *     *           ***            * *
*       *       *       *         *****          *   *
*       *       *       *           *           *     *
*       *       *       *           *          *       *
*       *       *       *           *           *     *
*       *       *       *           *            *   *
*       *        *     *            *             * *
*********          ***              *              *
```

```
// Exercise 1.28 Solution
#include <iostream>

using std::cout;
using std::endl;

main()
{
   cout << "*********      ***         *         *\n"
        << "*       *    *     *      ***       * *\n"
        << "*       *   *       *    *****     *   *\n"
        << "*       *   *       *      *      *     *\n"
        << "*       *   *       *      *     *       *\n"
        << "*       *   *       *      *      *     *\n"
        << "*       *   *       *      *       *   *\n"
        << "*       *    *     *       *        * *\n"
        << "*********      ***         *         *" << endl;

   return 0;
}
```

1.29 What does the following code print?

```
cout << "*\n**\n***\n****\n*****\n";
```

ANS:

```
*
**
***
****
*****
```

1.30 Write a program that reads in five integers and determines and prints the largest and the smallest integers in the group. Use only the programming techniques you learned in this chapter.

```
// Exercise 1.30 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num1, num2, num3, num4, num5, largest, smallest;

   cout << "Enter five integers: ";
   cin >> num1 >> num2 >> num3 >> num4 >> num5;

   largest = num1;
   smallest = num1;

   if ( num1 > largest )
      largest = num1;

   if ( num2 > largest )
      largest = num2;

   if ( num3 > largest )
      largest = num3;

   if ( num4 > largest )
      largest = num4;

   if ( num5 > largest )
      largest = num5;

   if ( num1 < smallest )
      smallest = num1;

   if ( num2 < smallest )
      smallest = num2;

   if ( num3 < smallest )
      smallest = num3;

   if ( num4 < smallest )
      smallest = num4;

   if ( num5 < smallest )
      smallest = num5;

   cout << "Largest is " << largest
```

```
          << "\nSmallest is " << smallest << endl;

     return 0;
  }
```

```
Enter five integers: 88 22 8 78 21
Largest is 88
Smallest is 8
```

1.31 Write a program that reads an integer and determines and prints whether it is odd or even. (Hint: Use the modulus operator. An even number is a multiple of two. Any multiple of two leaves a remainder of zero when divided by 2.)

```
// Exercise 1.31 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num;

   cout << "Enter a number: ";
   cin >> num;

   if ( num % 2 == 0 )
      cout << "The number " << num << " is even." << endl;

   if ( num % 2 != 0 )
      cout << "The number " << num << " is odd." << endl;

   return 0;
}
```

```
Enter a number: 73
The number 73 is odd.
```

1.32 Write a program that reads in two integers and determines and prints if the first is a multiple of the second. (Hint: Use the modulus operator.)

```
// Exercise 1.32 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num1, num2;

   cout << "Enter two integers: ";
   cin >> num1 >> num2;

   if ( num1 % num2 == 0 )
      cout << num1 << " is a multiple of " << num2 << endl;

   if ( num1 % num2 != 0 )
      cout << num1 << " is not a multiple of " << num2 << endl;

   return 0;
}
```

```
Enter two integers: 22 8
22 is not a multiple of 8
```

1.33 Display a checkerboard pattern with eight output statements, then display the same pattern with as few output statements as possible.

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

```
// Exercise 1.33 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   // Eight output statements
   cout << "* * * * * * * *\n";
   cout << " * * * * * * * *\n";
   cout << "* * * * * * * *\n";
   cout << " * * * * * * * *\n";
   cout << "* * * * * * * *\n";
   cout << " * * * * * * * *\n";
   cout << "* * * * * * * *\n";
   cout << " * * * * * * * *\n\n";

   // One output statement; 3 parts
   cout << "* * * * * * * *\n * * * * * * * *\n* * * * * * * *\n"
        << " * * * * * * * *\n* * * * * * * *\n  * * * * * * * *\n"
        << "* * * * * * * *\n * * * * * * * *\n";

   cout << endl;   // ensure everything is displayed

   return 0;
}
```

```
 * * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *

* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
  * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

1.34 Distinguish between the terms fatal error and non–fatal error. Why might you prefer to experience a fatal error rather than a non–fatal error?

ANS: A fatal error causes a program to terminate prematurely. A nonfatal error occurs when the logic of the program is incorrect, and the program does not work properly. A fatal error is preferred for debugging purposes. A fatal error immediately lets you know there is a problem with the program, whereas a nonfatal error can be subtle and possibly go undetected.

1.35 Here is a peek ahead. In this chapter you learned about integers and the type **`int`**. C++ can also represent uppercase letters, lowercase letters and a considerable variety of special symbols. C++ uses small integers internally to represent each different character. The set of characters a computer uses and the corresponding integer representations for those characters is called that computer's *character set*. You can print a character by simply enclosing that character in single quotes as with

```
cout << 'A';
```

You can print the integer equivalent of a character using **`static_cast`** as follows:

```
cout << static_cast< int >( 'A' );
```

This is called a *cast* operation (we formally introduce casts in Chapter 2). When the preceding statement executes, it prints the value 65 (on systems that use the *ASCII character set*). Write a program that prints the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. At a minimum, determine the integer equivalents of the following: **`A B C a b c 0 1 2 $ * + /`** and the blank character.

```
// Exercise 1.35 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   char symbol;

   cout << "Enter a character: ";
   cin >> symbol;

   cout << symbol << "'s integer equivalent is "
        << static_cast< int > ( symbol ) << endl;

   return 0;
}
```

```
Enter a character: A
A's integer equivalent is 65
```

1.36 Write a program that inputs a five-digit number, separates the number into its individual digits and prints the digits separated from one another by three spaces each. (Hint: Use the integer division and modulus operators.) For example, if the user types in **42339** the program should print

```
4   2   3   3   9
```

```
// Exercise 1.36 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int num;

   cout << "Enter a five-digit number: ";
   cin >> num;

   cout << num / 10000 << "   ";
   num = num % 10000;
   cout << num / 1000 << "   ";
   num = num % 1000;
   cout << num / 100 << "   ";
   num = num % 100;
   cout << num / 10 << "   ";
   num = num % 10;
   cout << num << endl;

   return 0;
}
```

```
Enter a five-digit number: 42339
4   2   3   3   9
```

1.37 Using only the techniques you learned in this chapter, write a program that calculates the squares and cubes of the numbers from 0 to 10 and uses tabs to print the following table of values:

```
number  square  cube
0       0       0
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
```

```cpp
// Exercise 1.37 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int num;

   num = 0;
   cout << "\nnumber\tsquare\tcube\n"
        << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << "\n";

   num = num + 1;
   cout << num << '\t' << num * num << '\t' << num * num * num << endl;

   return 0;
}
```

1.38 Give a brief answer to each of the following “object think” questions:

a) Why does this text choose to discuss structured programming in detail before proceeding with an in-depth treatment of object-oriented programming?

ANS: Objects are composed in part by structured program pieces.

b) What are the typical steps (mentioned in the text) of an object-oriented design process?

ANS: (1) Determine which objects are needed to implement the system. (2) Determine’s each object’s attributes. (3) Determine each object’s behaviors. (4) Determine the interaction between the objects.

c) How is multiple inheritance exhibited by human beings?

ANS: Children. A child receives genes from both parents.

d) What kinds of messages do people send to one another?

ANS: People send messages through body language, speech, writings, email, telephones, etc.

e) Objects send messages to one another across well-defined interfaces. What interfaces does a car radio (object) present to its user (a person object)?

ANS: Dials and buttons that allow the user to select a station, adjust the volume, adjust bass and treble, play a CD or tape, etc.

1.39 You are probably wearing on your wrist one of the world’s most common types of objects—a watch. Discuss how each of the following terms and concepts applies to the notion of a watch: object, attributes, behaviors, class, inheritance (consider, for example, an alarm clock), abstraction, modeling, messages, encapsulation, interface, information hiding, data members and member functions.

ANS: The entire watch is an object that is composed of many other objects (such as the moving parts, the band, the face, etc.) Watch attributes are time, color, band, style (digital or analog), etc. The behaviors of the watch include setting the tim and getting the time. A watch can be considered a specific type of clock (as can an alarm clock). With that in mind, it is possible that a class called`Clock` could exist from which other classes such as watch and alarm clock can inherit the basic features in the clock. The watch is an abstraction of the mechanics needed to keep track of the time. The user of the watch does not need to know the mechanics of the watch in order to use it; the user only needs to know that the watch keeps the proper time. In this sense, the mechanics of the watch are encapsulated (hidden) inside the watch. The interface to the watch (its face and controls for setting the time) allows the user to set and get the time. The user is not allowed to directly touch the internal mechanics of the watch. All interaction with the internal mechanics is controlled by the interface to the watch. The data members stored in the watch are hidden inside the watch and the member functions (looking at the face to get the time and setting the time) provide the interface to the data.

2

Control Structures Solutions

Solutions

Exercises 2.14 through 2.38 correspond to Sections 2.1 through 2.12.
Exercises 2.39 through 2.63 correspond to Sections 2.13 through 2.21.

2.14 Identify and correct the error(s) in each of the following:

a)
```
if ( age >= 65 );
   cout << "Age is greater than or equal to 65" << endl;
else
   cout << "Age is less than 65 << endl";
```
ANS: The semicolon at the end of the **`if`** should be removed. The closing double quote after the second **`endl`** should be placed after **`65`**.

b)
```
if ( age >= 65 )
   cout << "Age is greater than or equal to 65" << endl;
else;
   cout << "Age is less than 65 << endl";
```
ANS: The semicolon after the **`else`** should be removed. The closing double quote after the second **`endl`** should be placed after **`65`**.

c)
```
int x = 1, total;
while ( x <= 10 ) {
   total += x;
   ++x;
}
```
ANS: Variable **`total`** should be initialized to **`0`**.

d)
```
While ( x <= 100 )
   total += x;
   ++x;
```
ANS: The **`W`** in **`while`** should be lowercase. The **`while`**'s body should be enclosed in braces **`{}`**.

e)
```
while ( y > 0 ) {
   cout << y << endl;
   ++y;
}
```
ANS: The variable **`y`** should be decremented (i.e., **`--y;`**) not incremented (**`++y;`**).

2.15 What does the following program print?

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int y, x = 1, total = 0;

   while ( x <= 10 ) {
      y = x * x;
      cout << y << endl;
      total += y;
      ++x;
   }

   cout << "Total is " << total << endl;
   return 0;
}
```

```
1
4
9
16
25
36
49
64
81
100
Total is 385
```

For Exercises 2.16 to 2.19, perform each of these steps:

a) Read the problem statement.
b) Formulate the algorithm using pseudocode and top-down, stepwise refinement.
c) Write a C++ program.
d) Test, debug and execute the C++ program.

2.16 Drivers are concerned with the mileage obtained by their automobiles. One driver has kept track of several tankfuls of gasoline by recording miles driven and gallons used for each tankful. Develop a C++ program that will input the miles driven and gallons used for each tankful. The program should calculate and display the miles per gallon obtained for each tankful. After processing all input information, the program should calculate and print the combined miles per gallon obtained for all tankfuls.

```
Enter the gallons used (-1 to end): 12.8
Enter the miles driven: 287
The miles / gallon for this tank was 22.421875

Enter the gallons used (-1 to end): 10.3
Enter the miles driven: 200
The miles / gallon for this tank was 19.417475

Enter the gallons used (-1 to end): 5
Enter the miles driven: 120
The miles / gallon for this tank was 24.000000

Enter the gallons used (-1 to end): -1

The overall average miles/gallon was 21.601423
```

ANS:

Top:

```
Determine the average miles/gallon for each tank of gas, and the overall miles/gallons for an arbitrary number
      of tanks of gas.
```

First refinement:

```
Initialize variables.
Input the gallons used and the miles driven, and calculate and print the miles/gallon for each tank of gas. Keep
      track of the total miles and total gallons.
Calculate and print the overall average miles/gallon.
```

Second refinement:

```
Initialize totalGallons to zero
Initialize totalMiles to zero

Input the gallons used for the first tank

While the sentinel value (-1) has not been entered for the gallons
      Add gallons to the running total in totalGallons
      Input the miles driven for the current tank
      Add miles to the running total in totalMiles
      Calculate and print the miles/gallon
      Input the gallons used for thenext tank

Print the average miles/gallon
```

```
// Exercise 2.16 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

```

```
int main()
{
   double gallons, miles, totalGallons = 0.0,
         totalMiles = 0.0, average;

   cout << "Enter the gallons used (-1 to end): ";
   cin >> gallons;

   while ( gallons != -1.0 ) {
      totalGallons += gallons;

      cout << "Enter the miles driven: ";
      cin >> miles;
      totalMiles += miles;

      cout << "The Miles / Gallon for this tank was "
           << miles / gallons
           << "\n\nEnter the gallons used (-1 to end): ";
      cin >> gallons;
   }

   average = totalMiles / totalGallons;
   cout << "\nThe overall average Miles/Gallon was "
        << average << endl;

   return 0;
}
```

```
Enter the gallons used (-1 to end): 16
Enter the miles driven: 220
The Miles / Gallon for this tank was 13.75

Enter the gallons used (-1 to end): 16.5
Enter the miles driven: 272
The Miles / Gallon for this tank was 16.4848

Enter the gallons used (-1 to end): -1

The overall average Miles/Gallon was 15.1385
```

2.17 Develop a C++ program that will determine if a department store customer has exceeded the credit limit on a charge account. For each customer, the following facts are available:

a) Account number (an integer)
b) Balance at the beginning of the month
c) Total of all items charged by this customer this month
d) Total of all credits applied to this customer's account this month
e) Allowed credit limit

The program should input each of these facts, calculate the new balance (= beginning balance + charges – credits) and determine if the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the program should display the customer's account number, credit limit, new balance and the message "Credit limit exceeded".

```
Enter account number (-1 to end): 100
Enter beginning balance: 5394.78
Enter total charges: 1000.00
Enter total credits: 500.00
Enter credit limit: 5500.00
Account:      100
Credit limit: 5500.00
Balance:      5894.78
Credit Limit Exceeded.

Enter account number (-1 to end): 200
Enter beginning balance: 1000.00
Enter total charges: 123.45
Enter total credits: 321.00
Enter credit limit: 1500.00

Enter account number (-1 to end): 300
Enter beginning balance: 500.00
Enter total charges: 274.73
Enter total credits: 100.00
Enter credit limit: 800.00

Enter account number (-1 to end): -1
```

ANS:

Top:

Determine if each of an arbitrary number of department store customers has exceeded the credit limit on a charge account.

First refinement:

Input the account number, beginning balance, total charges, total credits, and credit limit for a customer, calculate the customer's new balance and determine if the balance exceeds the credit limit. Then process the next customer.

Second refinement:

```
Input the first customer's account number

While the sentinel value (-1) has not been entered for the account number
    Input the customer's beginning balance
    Input the customer's total charges
    Input the customer's total credits
    Input the customer's credit limit
    Calculate the customer's new balance

    If the balance exceeds the credit limit
        Print the account number
        Print the credit limit
        Print the balance
        Print "Credit Limit Exceeded"

    Input the next customer's account number
```

```
// Exercise 2.17 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   int accountNumber;
   double balance, charges, credits, limit;

   cout << "Enter account number (-1 to end): "
        << setiosflags(ios::fixed | ios::showpoint);
   cin >> accountNumber;

   while ( accountNumber != -1 ) {
      cout << "Enter beginning balance: ";
      cin >> balance;
      cout << "Enter total charges: ";
      cin >> charges;
      cout << "Enter total credits: ";
      cin >> credits;
      cout << "Enter credit limit: ";
      cin >> limit;
      balance += charges - credits;

      if ( balance > limit )
         cout << "Account:      " << accountNumber
              << "\nCredit limit: " << setprecision( 2 ) << limit
              << "\nBalance:      " << setprecision( 2 ) << balance
              << "\nCredit Limit Exceeded.\n";

      cout << "\nEnter account number (-1 to end): ";
      cin >> accountNumber;
   }

   cout << endl;  // ensure all output is displayed

   return 0;
}
```

```
Enter account number (-1 to end): 88
Enter beginning balance: 900.57
Enter total charges: 324.78
Enter total credits: 100
Enter credit limit: 1500

Enter account number (-1 to end): 89
Enter beginning balance: 2789.65
Enter total charges: 1540.55
Enter total credits: 25
Enter credit limit: 1500
Account:      89
Credit limit: 1500.00
Balance:      4305.20
Credit Limit Exceeded.

Enter account number (-1 to end): -1
```

2.18 One large chemical company pays its salespeople on a commission basis. The salespeople receive $200 per week plus 9 percent of their gross sales for that week. For example, a salesperson who sells $5000 worth of chemicals in a week receives $200 plus 9 percent of $5000, or a total of $650. Develop a C++ program that will input each salesperson's gross sales for last week and calculate and display that salesperson's earnings. Process one salesperson's figures at a time.

```
Enter sales in dollars (-1 to end): 5000.00
Salary is: $650.00

Enter sales in dollars (-1 to end): 6000.00
Salary is: $740.00

Enter sales in dollars (-1 to end): 7000.00
Salary is: $830.00

Enter sales in dollars (-1 to end): -1
```

ANS:

Top:

```
For an arbitrary number of salespeople, determine each salesperson’s earnings for the last week.
```

First refinement:

```
Initialize variables

Input the salesperson’s sales for the week, calculate and print the salesperson’s wages for the week then
    process the next salesperson.
```

Second refinement:

```
Input the first salesperson’s sales in dollars

While the sentinel value (-1) has not been entered for the sales
    Calculate the salesperson’s wages for the week
    Print the salesperson’s wages for the week
    Input the next salesperson’s sales in dollars
```

```
// Exercise 2.18 Solution
#include <iostream>

using std::cout;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   double sales, wage;

   cout << "Enter sales in dollars (-1 to end): "
        << setiosflags( ios::fixed | ios::showpoint );
   cin >> sales;

   while ( sales != -1.0 ) {
      wage = 200.0 + 0.09 * sales;
      cout << "Salary is: $" << setprecision( 2 ) << wage
           << "\n\nEnter sales in dollars (-1 to end): ";
      cin >> sales;
   }

   return 0;
}
```

```
Enter sales in dollars (-1 to end): 7000
Salary is: $830.00

Enter sales in dollars (-1 to end): 500
Salary is: $245.00

Enter sales in dollars (-1 to end): -1
```

2.19 Develop a C++ program that will determine the gross pay for each of several employees. The company pays “straight-time” for the first 40 hours worked by each employee and pays “time-and-a-half” for all hours worked in excess of 40 hours. You are given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your program should input this information for each employee and should determine and display the employee's gross pay.

```
Enter hours worked (-1 to end): 39
Enter hourly rate of the worker ($00.00): 10.00
Salary is $390.00

Enter hours worked (-1 to end): 40
Enter hourly rate of the worker ($00.00): 10.00
Salary is $400.00

Enter hours worked (-1 to end): 41
Enter hourly rate of the worker ($00.00): 10.00
Salary is $415.00

Enter hours worked (-1 to end): -1
```

ANS:

Top:

```
Determine the gross pay for an arbitrary number of employees
```

Second refinement:

```
Input the hours worked for the first employee

While the sentinel value (-1) has not been entered for the hours
    Input the employee’s hourly rate

    If the hours input is less than or equal to 40
        Calculate gross pay by multiplying hours worked by hourly rate
    Else
        Calculate gross pay by multiplying hours worked above forty by 1.5 times the hourly rate and
    adding 40 hours worked by the hourly rate.
    Display the employee’s gross pay.
    Input the hours worked for thenext employee

Print the average miles/gallon
```

```
// Exercise 2.19 Solution
#include <iostream>

using std::cout;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   double hours, rate, salary;

   cout << "Enter hours worked (-1 to end): "
        << setiosflags( ios::fixed | ios::showpoint );
   cin >> hours;

   while ( hours != -1.0 ) {
      cout << "Enter hourly rate of the worker ($00.00): ";
      cin >> rate;

      if ( hours <= 40 )
         salary = hours * rate;
      else
         salary = 40.0 * rate + ( hours - 40.0 ) * rate * 1.5;

      cout << "Salary is $" << setprecision( 2 ) << salary
           << "\n\nEnter hours worked (-1 to end): ";
      cin >> hours;
   }

   return 0;
}
```

```
Enter hours worked (-1 to end): 40
Enter hourly rate of the worker ($00.00): 10
Salary is $400.00

Enter hours worked (-1 to end): 50
Enter hourly rate of the worker ($00.00): 10
Salary is $550.00

Enter hours worked (-1 to end): -1
```

2.20 The process of finding the largest number (i.e., the maximum of a group of numbers) is used frequently in computer applications. For example, a program that determines the winner of a sales contest would input the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a pseudocode program, then a C++ program that inputs a series of 10 numbers, and determines and prints the largest of the numbers. Hint: Your program should use three variables, as follows:

`counter:`	A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed).
`number:`	The current number input to the program.
`largest:`	The largest number found so far.

```
// Exercise 2.20 solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int counter = 0, number, largest;

   cout << "Enter the first number: ";
   cin >> largest;

   while ( ++counter < 10 ) {
      cout << "Enter the next number : ";
      cin >> number;

      if ( number > largest )
         largest = number;
   }

   cout << "Largest is " << largest << endl;

   return 0;
}
```

```
Enter the first number: 78
Enter the next number : 19
Enter the next number : 99
Enter the next number : 33
Enter the next number : 22
Enter the next number : 10
Enter the next number : 8
Enter the next number : 88
Enter the next number : 22
Enter the next number : 34
Largest is 99
```

2.21 Write a C++ program that utilizes looping and the tab escape sequence `\t` to print the following table of values:

```
N       10*N    100*N   1000*N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

```
// Exercise 2.21 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int n = 0;

   cout << "N\t10 * N\t100 * N\t1000 * N\n\n";

   while ( ++n <= 5 )
      cout << n << '\t' << 10 * n << '\t' << 100 * n
           << '\t' << 1000 * n << '\n';

   cout << endl;

   return 0;
}
```

```
N       10 * N  100 * N 1000 * N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

2.22 Using an approach similar to Exercise 2.20, find the *two* largest values among the 10 numbers. Note: You must input each number only once.

```cpp
// Exercise 2.22 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int counter = 0, number, largest, secondLargest = 0;

   cout << "Enter the first number: ";
   cin >> largest;

   while ( ++counter < 10 ) {
      cout << "Enter next number: ";
      cin >> number;

      if ( number > largest ) {
         secondLargest = largest;
         largest = number;
      }
      else if ( number > secondLargest )
         secondLargest = number;
   }

   cout << "\nLargest is " <<  largest
        << "\nSecond largest is " << secondLargest << endl;

   return 0;
}
```

```
Enter the first number: 77
Enter next number: 33
Enter next number: 44
Enter next number: 73
Enter next number: 79
Enter next number: 45
Enter next number: 2
Enter next number: 22
Enter next number: 21
Enter next number: 8

Largest is 79
Second largest is 77
```

2.23 Modify the program in Fig. 2.11 to validate its inputs. On any input, if the value entered is other than 1 or 2, keep looping until the user enters a correct value.

```cpp
// Exercise 2.23 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int passes = 0, failures = 0, student = 0, result;

   while ( ++student <= 10 ) {
      cout << "Enter result (1=pass, 2=fail): ";
      cin >> result;

      while ( result != 1 && result != 2 ) {
         cout << "Invalid result"
              << "\nEnter result (1=pass, 2=fail): ";
         cin >> result;
      }

      if ( result == 1 )
         ++passes;
      else
         ++failures;
   }

   cout << "Passed: " << passes
        << "\nFailed: " << failures;

   if ( passes >= 8 )
      cout << "\nRaise tuition\n";

   cout << endl;

   return 0;
}
```

```
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 4
Invalid result
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 5
Invalid result
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 1
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 2
Enter result (1=pass, 2=fail): 2
Passed: 6
Failed: 4
```

2.24 What does the following program print?

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int count = 1;

   while ( count <= 10 ) {
      cout << (count % 2 ? "****" : "++++++++")
           << endl;
      ++count;
   }

   return 0;
}
```

```
****
++++++++
****
++++++++
****
++++++++
****
++++++++
****
++++++++
```

2.25 What does the following program print?

```
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int row = 10, column;

   while ( row >= 1 ) {
      column = 1;

      while ( column <= 10 ) {
         cout << ( row % 2 ? "<" : ">" );
         ++column;
      }

      --row;
      cout << endl;
   }

   return 0;
}
```

```
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
```

2.26 *(Dangling Else Problem)* Determine the output for each of the following when **x** is **9** and **y** is **11** and when **x** is **11** and **y** is **9**. Note that the compiler ignores the indentation in a C++ program. Also, the C++ compiler always associates an **else** with the previous **if** unless told to do otherwise by the placement of braces **{}**. Because, on first glance, the programmer may not be sure which **if** an **else** matches, this is referred to as the "dangling else" problem. We have eliminated the indentation from the following code to make the problem more challenging. (Hint: Apply indentation conventions you have learned.)

a)
```
if ( x < 10 )
if ( y > 10 )
cout << "*****" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
```
ANS: x = 9, y = 11

```
*****
$$$$$
```

b)
```
if ( x < 10 ) {
if ( y > 10 )
cout << "*****" << endl;
}
else {
cout << "#####" << endl;
cout << "$$$$$" << endl;
}
```
ANS: x = 11, y = 9

```
#####
$$$$$
```

2.27 *(Another Dangling Else Problem)* Modify the following code to produce the output shown. Use proper indentation techniques. You must not make any changes other than inserting braces. The compiler ignores indentation in a C++ program. We have eliminated the indentation from the following code to make the problem more challenging. Note: It is possible that no modification is necessary.

```
if ( y == 8 )
if ( x == 5 )
cout << "@@@@@" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
cout << "&&&&&" << endl;
```

a) Assuming **x = 5** and **y = 8**, the following output is produced.

```
@@@@@
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      cout << "@@@@@\n";
   else
      cout << "#####\n";

cout << "$$$$$\n";
cout << "&&&&&\n";
```

b) Assuming **x = 5** and **y = 8**, the following output is produced.

```
@@@@@
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      cout << "@@@@@\n";
   else {
      cout << "#####\n";
      cout << "$$$$$\n";
      cout << "&&&&&\n";
   }
```

c) Assuming **x = 5** and **y = 8**, the following output is produced.

```
@@@@@
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      cout << "@@@@@\n";
   else {
      cout << "#####\n";
      cout << "$$$$$\n";
   }

cout << "&&&&&\n";
```

d) Assuming **x = 5** and **y = 7**, the following output is produced. Note: The last three output statements after the **else** are all part of a compound statement.

```
#####
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 ) {
   if ( x == 5 )
      cout << "@@@@@\n";
}
else {
   cout << "#####\n";
   cout << "$$$$$\n";
   cout << "&&&&&\n";
}
```

2.28 Write a program that reads in the size of the side of a square and then prints a hollow square of that size out of asterisks and blanks. Your program should work for squares of all side sizes between 1 and 20. For example, if your program reads a size of 5,it should print

```
*****
*   *
*   *
*   *
*****
```

```
// Exercise 2.28 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int side, rowPosition, size;

   cout << "Enter the square side: ";
   cin >> side;

   size = side;

   while ( side > 0 ) {
      rowPosition = size;

      while ( rowPosition > 0 ) {

         if ( size == side || side == 1 || rowPosition == 1 ||
             rowPosition == size )

            cout << '*';
         else
```

```
            cout << ' ';

         --rowPosition;
      }

      cout << '\n';
      --side;
   }

   cout << endl;

   return 0;
}
```

```
Enter the square side: 8
********
*      *
*      *
*      *
*      *
*      *
*      *
********
```

2.29 A palindrome is a number or a text phrase that reads the same backwards as forwards. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a program that reads in a five-digit integer and determines whether it is a palindrome. (Hint: Use the division and modulus operators to separate the number into its individual digits.)

```
// Exercise 2.29 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int number, firstDigit, secondDigit, fourthDigit, fifthDigit;

   cout << "Enter a five-digit number: ";
   cin >> number;

   firstDigit = number / 10000;
   secondDigit = number % 10000 / 1000;
   fourthDigit = number % 10000 % 1000 % 100 / 10;
   fifthDigit = number % 10000 % 1000 % 10;

   if ( firstDigit == fifthDigit && secondDigit == fourthDigit )
      cout << number << " is a palindrome" << endl;
   else
      cout << number << " is not a palindrome" << endl;

   return 0;
}
```

```
Enter a five-digit number: 57475
57475 is a palindrome
```

2.30 Input an integer containing only 0s and 1s (i.e., a “binary” integer) and print its decimal equivalent. (Hint: Use the modulus and division operators to pick off the “binary” number’s digits one at a time from right to left. Just as in the decimal number system where the rightmost digit has a positional value of 1 and the next digit left has a positional value of 10, then 100, then 1000,etc., in the binary number system, the rightmost digit has a positional value of 1, the next digit left has a positional value of 2, then 4, then 8, etc. Thus the decimal number 234 can be interpreted as 4 * 1 + 3 * 10 + 2 * 100. The decimal equivalent of binary 1101 is 1 * 1 + 0 * 2 + 1 * 4 + 1 * 8 or 1 + 0 + 4 + 8, or 13.)

```cpp
// Exercise 2.30 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int binary, number, decimal = 0, highBit = 16, factor = 10000;

   cout << "Enter a binary number (5 digits maximum): ";
   cin >> binary;

   number = binary;

   while ( highBit >= 1 ) {
      decimal += binary / factor * highBit;
      highBit /= 2;
      binary %= factor;
      factor /= 10;
   }

   cout << "The decimal equivalent of "
        << number << " is " << decimal << endl;

   return 0;
}
```

```
Enter a binary number (5 digits maximum): 10010
The decimal equivalent of 10010 is 18
```

2.31 Write a program that displays the following checkerboard pattern

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

Your program must use only three output statements, one of each of the following forms:

```
cout << "* ";
cout << ' ';
cout << endl;
```

```
// Exercise 2.31 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int side = 8, row;

   while ( side-- > 0 ) {
      row = 8;

      if ( side % 2 == 0 )
         cout << ' ';

      while ( row-- > 0 )
         cout << "* ";

      cout << '\n';
   }

   cout << endl;

   return 0;
}
```

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

2.32 Write a program that keeps printing the multiples of the integer 2, namely 2, 4, 8, 16, 32, 64, etc. Your loop should not terminate (i.e., you should create an infinite loop). What happens when you run this program?

```
// Exercise 2.32 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int multiple = 1;

   while ( multiple *= 2 )
      cout << multiple << "\n";

   cout << endl;

   return 0;
}
```

```
2
4
...
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
```

2.33 Write a program that reads the radius of a circle (as a `double` value) and computes and prints the diameter, the circumference and the area. Use the value 3.14159 for π.

```
// Exercise 2.33 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   double radius, pi = 3.14159;

   cout << "Enter the radius: ";
   cin >> radius;

   cout << "The diameter is " << radius * 2.0
        << "\nThe circumference is " << 2.0 * pi * radius
        << "\nThe area is " << pi * radius * radius << endl;

   return 0;
}
```

```
Enter the radius: 5
The diameter is 10
The circumference is 31.4159
The area is 78.5397
```

2.34 What's wrong with the following statement? Provide the correct statement to accomplish what the programmer was probably trying to do.

```
cout << ++( x + y );
```

ANS: The `++` operator must be used in conjuction with variables. The programmer probably intended to write the statement: `cout << x + y + 1;`.

2.35 Write a program that reads three nonzero `double` values and determines and prints if they could represent the sides of a right triangle.

```
// Exercise 2.35 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   double a, b, c;

   cout << "Enter three floating point numbers: ";
   cin >> a >> b >> c;

   if ( c * c == a * a + b * b )
      cout << "The three numbers could"
           << " be sides of a right triangle" << endl;
   else
      cout << "The three numbers probably"
           << " are not the sides of a right triangle" << endl;

   return 0;
}
```

```
Enter three floating point numbers: 4.5 7.7 6.6
The three numbers probably are not the sides of a right triangle
```

2.36 Write a program that reads three nonzero integers and determines and prints if they could be the sides of a right triangle.

```
// Exercise 2.36 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
  int a, b, c;

  do {
     cout << "Enter three integers: ";
     cin >> a >> b >> c;
  } while ( a <= 0 || b <= 0 || c <= 0 );

  if ( c * c == a * a + b * b )
     cout << "The three integers are the"
          << " sides of a right triangle\n";
  else
     cout << "The three integers are not the"
          << " sides of a right triangle\n";

```

```
   cout << endl;

   return 0;
}
```

```
Enter three integers: 3 4 5
The three integers are the sides of a right triangle
```

2.37 A company wants to transmit data over the telephone, but they are concerned that their phones may be tapped. All of their data are transmitted as four-digit integers. They have asked you to write a program that encrypts their data so that it can be t rans-mitted more securely. Your program should read a four-digit integer and encrypt it as follows: Replace each digit by *(the sum of that digit plus 7) modulus 10*. Then, swap the first digit with the third, swap the second digit with the fourth and print the encrypted integer. Write a separate program that inputs an encrypted four-digit integer and decrypts it to form the original number.

```
// Exercise 2.37 Part A Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int first, second, third, fourth, digit, temp;
   int encryptedNumber;

   cout << "Enter a four digit number to be encrypted: ";
   cin >> digit;

   first = ( digit / 1000 + 7 ) % 10;
   second = ( digit % 1000 / 100 + 7 ) % 10;
   third = ( digit % 1000 % 100 / 10 + 7 ) % 10;
   fourth = ( digit % 1000 % 100 % 10 + 7 ) % 10;

   temp = first;
   first = third * 1000;
   third = temp * 10;

   temp = second;
   second = fourth * 100;
   fourth = temp * 1;

   encryptedNumber = first + second + third + fourth;
   cout << "Encrypted number is " << encryptedNumber << endl;

   return 0;
}
```

```
Enter a four digit number to be encrypted: 1009
Encrypted number is 7687
```

```
// Exercise 2.37 Part B Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
```

```

int main()
{
   int first, second, third, fourth, decrypted, temp, num;

   cout << "Enter a four digit encrypted number: ";
   cin >> num;

   first = num / 1000;
   second = num % 1000 / 100;
   third = num % 1000 % 100 / 10;
   fourth = num % 1000 % 100 % 10;

   temp = ( first + 3 ) % 10;
   first = ( third + 3 ) % 10;
   third = temp;

   temp = ( second + 3 ) % 10;
   second = ( fourth + 3 ) % 10;
   fourth = temp;

   decrypted = first * 1000 + second * 100 + third * 10 + fourth;
   cout << "Decrypted number is " << decrypted << endl;

   return 0;
}
```

```
Enter a four digit encrypted number: 7687
Decrypted number is 1009
```

2.38 The factorial of a nonnegative integer n is written $n!$ (pronounced "n factorial") and is defined as follows:

$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$ (for values of n greater than or equal to 1)

and

$n! = 1$ (for $n = 0$).

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120.

a) Write a program that reads a nonnegative integer and computes and prints its factorial.

b) Write a program that estimates the value of the mathematical constant e by using the formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

c) Write a program that computes the value of e^x by using the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

```
// Exercise 2.38 Part A Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int n = 0, number;
   unsigned factorial = 1;

   do {
      cout << "Enter a positive integer: ";
      cin >> number;
   } while ( number < 0 );

   while ( n++ < number )
      factorial *= n == 0 ? 1 : n;

   cout << number << "! is " << factorial << endl;

   return 0;
}
```

```
Enter a positive integer: 8
8! is 40320
```

```
// Exercise 2.38 Part B Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int n = 0, fact = 1, accuracy = 10;
   double e = 1;

   while ( ++n < accuracy ) {
      fact *= n;
      e += 1.0 / fact;
   }

   cout << "e is " << e << endl;

   return 0;
}
```

```
e is 2.71828
```

```
// Exercise 2.38 Part C Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>
using std::setw;
using std::setprecision;
using std::setiosflags;

int main()
{
   int n = 0, accuracy = 15, x = 3.0, times = 0, count;
   double e = 1.0, exp = 0.0, fact = 1.0;

   while ( n++ <= accuracy ) {
      count = n;
      fact *= n == 0 ? 1.0 : n;

      while ( times < count ) {

         if ( times == 0 )
            exp = 1.0;

         exp *= x;
         ++times;
      }

      e += exp / fact;
    }

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "e raised to the " << x << " power is "
        << setprecision( 4 ) << e << endl;

   return 0;
}
```

```
e raised to the 3 power is 20.0855
```

2.39 Find the error(s) in each of the following:

a) **`For ( x = 100, x >= 1, x++ )`**
```
   cout << x << endl;
```
ANS: `For` should be `for`. The commas should be semicolons. The `++` should be a decrement such as `--`.

b) The following code should print whether integer `value` is odd or even:
```
switch ( value % 2 ) {
   case 0:
      cout << "Even integer" << endl;
   case 1:
      cout << "Odd integer" << endl;
}
```
ANS: `case 0` needs a `break` statement.

c) The following code should output the odd integers from 19 to 1:
```
for ( x = 19; x >= 1; x += 2 )
   cout << x << endl;
```
ANS: `+=` should be `-=`.

d) The following code should output the even integers from 2 to 100:
```
counter = 2;
do {
   cout << counter << endl;
   counter += 2;
} While ( counter < 100 );
```
ANS: `While` should be `while`. Operator `<` should be `<=`.

2.40 Write a program that sums a sequence of integers. Assume that the first integer read specifies the number of values remaining to be entered. Your program should read only one value per input statement. A typical input sequence might be

```
5 100 200 300 400 500
```

where the **5** indicates that the subsequent **5** values are to be summed.

```
// Exercise 2.40 Solution
#include <iostream>


using std::cout;
using std::endl;
using std::cin;

int main()
{
   int sum = 0, number, value;

   cout << "Enter the number of values to be processed: ";
   cin >> number;

   for ( int i = 1; i <= number; i++ ) {
      cout << "Enter a value: ";
      cin >> value;
      sum += value;
   }

   cout << "Sum of the " << number << " values is "
        << sum << endl;

   return 0;
}
```

```
Enter the number of values to be processed: 3
Enter a value: 7
Enter a value: 8
Enter a value: 9
Sum of the 3 values is 24
```

2.41 Write a program that calculates and prints the average of several integers. Assume the last value read is the sentinel **9999**. A typical input sequence might be

```
10 8 11 7 9 9999
```

indicating that the average of all the values preceding **9999** is to be calculated.

```
// Exercise 2.41 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int value, count = 0, total = 0;

   cout << "Enter an integer (9999 to end): ";
   cin >> value;

   while ( value != 9999 ) {
      total += value;
      ++count;
      cout << "Enter next integer (9999 to end): ";
      cin >> value;
   }

   if ( count != 0 )
      cout << "\nThe average is: "
           << static_cast< double > ( total ) / count << endl;
   else
      cout << "\nNo values were entered." << endl;

   return 0;
}
```

```
Enter an integer (9999 to end): 88
Enter next integer (9999 to end): 65
Enter next integer (9999 to end): 77
Enter next integer (9999 to end): 43
Enter next integer (9999 to end): 90
Enter next integer (9999 to end): 45
Enter next integer (9999 to end): 76
Enter next integer (9999 to end): 2
Enter next integer (9999 to end): 9999

The average is: 60.75
```

2.42 What does the following program do?

```
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
   int x, y;

   cout << "Enter two integers in the range 1-20: ";
   cin >> x >> y;

   for ( int i = 1; i <= y; i++ ) {

      for ( int j = 1; j <= x; j++ )
         cout << '@';

      cout << endl;
   }

   return 0;
}
```

```
Enter two integers in the range of 1-20: 3 4
@@@
@@@
@@@
@@@
```

2.43 Write a program that finds the smallest of several integers. Assume that the first value read specifies the number of values remaining and that the first number is not one of the integers to compare.

```
// Exercise 2.43 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int number, value, smallest;

   cout << "Enter the number of integers to be processed: ";
   cin >> number;
   cout << "Enter an integer: ";
   cin >> smallest;

   for ( int i = 2; i <= number; i++ ) {
      cout << "Enter next integer: ";
      cin >> value;

      if ( value < smallest )
         smallest = value;
   }

   cout << "\nThe smallest integer is: " << smallest << endl;

   return 0;
}
```

```
Enter the number of integers to be processed: 5
Enter an integer: 5
Enter next integer: 4
Enter next integer: 3
Enter next integer: 1
Enter next integer: 8

The smallest integer is: 1
```

2.44 Write a program that calculates and prints the product of the odd integers from 1 to 15.

```
// Exercise 2.44 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   long product = 1;

   for ( long i = 3; i <= 15; i += 2 )
      product *= i;

   cout << "Product of the odd integers from 1 to 15 is: "
        << product << endl;

   return 0;
}
```

```
Product of the odd integers from 1 to 15 is: 2027025
```

2.45 The *factorial* function is used frequently in probability problems. The factorial of a positive integer *n* (written *n!* and pronounced "n factorial") is equal to the product of the positive integers from 1 to *n*. Write a program that evaluates the factorials of the integers from 1 to 5. Print the results in tabular format. What difficulty might prevent you from calculating the factorial of 20?

```
// Exercise 2.45 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int factorial;

   cout << "X\tFactorial of X\n";

   for ( int i = 1; i <= 5; ++i ) {
      factorial = 1;

      for ( int j = 1; j <= i; ++j )
         factorial *= j;

      cout << i << '\t' << factorial << '\n';
   }

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

```
X       Factorial of X
1       1
2       2
3       6
4       24
5       120
```

2.46 Modify the compound interest program of Section 2.15 to repeat its steps for interest rates of 5 percent, 6 percent, 7 percent, 8 percent, 9 percent and 10 percent. Use a **for** loop to vary the interest rate.

```
// Exercise 2.46 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
```

```cpp
using std::setiosflags;

#include <cmath>

int main()
{
   double amount, principal = 1000.00;

   cout << setiosflags( ios::fixed | ios::showpoint );

   for ( int rate = 5; rate <= 10; rate++ ) {
      cout << "Interest Rate: " << setprecision( 2 ) << rate / 100.0
           << "\nYear\tAmount on deposit\n";

      for ( int year = 1; year <= 10; year++ ) {
         amount = principal * pow( 1 + ( rate / 100.0 ), year );
         cout << year << '\t' << setprecision( 2 ) << amount << '\n';
      }

      cout << '\n';
   }

   cout << endl;

   return 0;
}
```

```
...
Interest Rate: 0.10
Year    Amount on deposit
1       1100.00
2       1210.00
3       1331.00
4       1464.10
5       1610.51
6       1771.56
7       1948.72
8       2143.59
9       2357.95
10      2593.74
```

2.47 Write a program that prints the following patterns separately one below the other. Use **`for`** loops to generate the patterns. All asterisks (**`*`**) should be printed by a single statement of the form **`cout << '*';`** (this causes the asterisks to print side by side). Hint: The last two patterns require that each line begin with an appropriate number of blanks. Extra credit: Combine your code from the four separate problems into a single program that prints all four patterns side by side by making clever use of nested **`for`** loops.

```
(A)           (B)           (C)           (D)
*             **********    **********             *
**            *********      *********            **
***           ********        ********           ***
****          *******          *******          ****
*****         ******            ******         *****
******        *****              *****        ******
*******       ****                ****       *******
********      ***                  ***      ********
*********     **                    **     *********
**********    *                      *    **********
```

```
// Exercise 2.47 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   // Pattern A
   for ( int row = 1; row <= 10; ++row ) {

      for ( int col = 1; col <= row; ++col )
         cout << '*';

      cout << '\n';
   }

   cout << '\n';

   // Pattern B
   for ( row = 10; row >= 1; --row ) {

      for ( int col = 1; col <= row; ++col )
         cout << '*';

      cout << '\n';
   }

   cout << '\n';

   // Pattern C
   for ( row = 10; row >= 1; --row ) {

      for ( int space = 1; space <= 10 - row; ++space )
         cout << ' ';

      for ( int col = 1; col <= row; ++col )
         cout << '*';

      cout << '\n';
   }

   cout << '\n';

   // Pattern D
   for ( row = 1; row <= 10; ++row ) {

      for ( int space = 1; space <= 10 - row; ++space )
         cout << ' ';

      for ( int col = 1; col <= row; ++col )
         cout << '*';

      cout << '\n';
   }

   cout << endl;

   return 0;
}
```

```
*
**
***
****
*****
******
*******
********
*********
**********

**********
*********
********
*******
******
*****
****
***
**
*

**********
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *

         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

```
// Exercise 2.47 Extra Credit Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   int row, column, space;

   for ( row = 1; row <= 10; ++row ) {

      // part a
      for ( column = 1; column <= row; ++column )
```

```
         cout << '*';

      for ( space = 1; space <= 10 - row; ++space )
         cout << ' ';

      cout << '\t';

      // part b
      for ( column = 10; column >= row; --column )
         cout << '*';

      for ( space = 1; space < row; ++space )
         cout << ' ';

      cout << '\t';

      // part c
      for ( space = 10; space > row; --space )
         cout << ' ';

      for ( column = 1; column <= row; ++column )
         cout << '*';

      cout << '\t';

      // part d
      for ( space = 1; space < row; ++space )
         cout << ' ';

      for ( column = 10; column >= row; --column )
         cout << '*';

      cout << endl;
   }

   return 0;
}
```

```
*               **********               *      **********
**              *********               **       *********
***             ********               ***        ********
****            *******               ****         *******
*****           ******               *****          ******
******          *****               ******           *****
*******         ****               *******            ****
********        ***               ********             ***
*********       **               *********              **
**********      *               **********               *
```

2.48 One interesting application of computers is drawing graphs and bar charts (sometimes called "histograms"). Write a program that reads five numbers (each between 1 and 30). For each number read, your program should print a line containing that number of adjacent asterisks. For example, if your program reads the number seven, it should print `*******`.

```
// Exercise 2.48 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
```

```

int main()
{
   int number;

   cout << "Enter 5 numbers between 1 and 30: ";

   for ( int i = 1; i <= 5; ++i ) {
      cin >> number;

      for ( int j = 1; j <= number; ++j )
         cout << '*';

      cout << '\n';
   }

   cout << endl;

   return 0;
}
```

```
Enter 5 numbers between 1 and 30: 16 12 8 27 9
****************
************
********
***************************
*********
```

2.49 A mail order house sells five different products whose retail prices are product 1 — $2.98, product 2—$4.50, product 3—$9.98, product 4—$4.49 and product 5—$6.87. Write a program that reads a series of pairs of numbers as follows:

a) Product number
b) Quantity sold for one day

Your program should use a `switch` statement to help determine the retail price for each product. Your program should calculate and display the total retail value of all products sold last week.

```
// Exercise 2.49 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   int product, quantity;
   double total = 0.0;

   cout << "Enter pairs of item numbers and quantities."
        << "\nEnter -1 for the item number to end input: ";
   cin >> product;

   while ( product != -1 ) {
      cin >> quantity;

      switch ( product ) {
         case 1:
            total += quantity * 2.98;
            break;
         case 2:
            total += quantity * 4.50;
            break;
         case 3:
            total += quantity * 9.98;
            break;
         case 4:
            total += quantity * 4.49;
            break;
         case 5:
            total += quantity * 6.87;
            break;
         default:
            cout << "Invalid product code: " << product
                 << "\n            Quantity: " << quantity << '\n';
            break;
      }

      cout << "Enter pairs of item numbers and quantities.\n"
           << "Enter -1 for the item number to end input: ";
      cin >> product;
   }

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "The total retail value was: " << setprecision( 2 )
        << total << endl;

   return 0;
}
```

```
Enter pairs of item numbers and quantities.
Enter -1 for the item number to end input: 2 10
Enter pairs of item numbers and quantities.
Enter -1 for the item number to end input: 1 5
Enter pairs of item numbers and quantities.
Enter -1 for the item number to end input: -1
The total retail value was: 59.90
```

2.50 Modify the program of Fig. 2.22 so that it calculates the grade-point average for the class. A grade of ‘A’ is worth 4 points, ‘B’ is worth 3 points, etc.

```
// Exercise 2.50 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

```

```
int main()
{
   int grade, gradeTotal = 0, gradeCount, aCount = 0,
       bCount = 0, cCount = 0, dCount = 0, fCount = 0;

   cout << "Enter the letter grades."
        << "\nEnter the EOF character to end input.\n"
        << setiosflags(ios::fixed | ios::showpoint);

   while ( ( grade = cin.get() ) != EOF ) {

      switch ( grade ) {
         case 'A': case 'a':
            gradeTotal += 4;
            ++aCount;
            break;
         case 'B': case 'b':
            gradeTotal += 3;
            ++bCount;
            break;
         case 'C': case 'c':
            gradeTotal += 2;
            ++cCount;
            break;
         case 'D': case 'd':
            ++gradeTotal;
            ++dCount;
            break;
         case 'F': case 'f':
            ++fCount;
            break;
         case ' ': case '\n':
            break;
         default:
            cout << "Incorrect letter grade entered."
                 << " Enter a new grade.\n";
            break;
      }
   }

   gradeCount = aCount + bCount + cCount + dCount + fCount;

   if ( gradeCount != 0 )
      cout << "\nThe class average is: " << setprecision( 1 )
           << static_cast< double > ( gradeTotal ) / gradeCount
           << endl;

   return 0;
}
```

```
Enter the letter grades.
Enter the EOF character to end input.
aABcccBbDf
E
Incorrect letter grade entered. Enter a new grade.
F
The class average is: 2.2
```

2.51 Modify the program in Fig. 2.21 so it uses only integers to calculate the compound interest. (Hint: Treat all monetary amounts as integral numbers of pennies. Then "break" the result into its dollar portion and cents portion by using the division and modulus operations. Insert a period.)

```
// Exercise 2.51 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

#include <cmath>

int main()
{
   int amount, principal = 1000, dollars, cents;
   double rate = .05;

   cout << "Year" << setw( 24 ) << "Amount on deposit\n";

   for ( int year = 1; year <= 10; ++year ) {
      amount = principal * pow( 1.0 + rate, year );
      cents = amount % 100;
      dollars = amount;  // assignment truncates decimal places
      cout << setw( 4 ) << year << setw( 21 ) << dollars << '.';

      if ( cents < 10 )
         cout << '0' << cents << endl;
      else
         cout << cents << endl;
   }

   return 0;
}
```

```
Year      Amount on deposit
   1               1050.50
   2               1102.02
   3               1157.57
   4               1215.15
   5               1276.76
   6               1340.40
   7               1407.07
   8               1477.77
   9               1551.51
  10               1628.28
```

2.52 Assume **i = 1**, **j = 2**, **k = 3** and **m = 2**. What does each of the following statements print? Are the parentheses necessary in each case?

a) `cout << ( i == 1 ) << endl;`
ANS: 1, no.
b) `cout << ( j == 3 ) << endl;`
ANS: 1, no.
c) `cout << ( i >= 1 && j < 4 ) << endl;`
ANS: 1, no.
d) `cout << ( m <= 99 && k < m ) << endl;`
ANS: 0, no.

e) `cout << ( j >= i || k == m ) << endl;`
ANS: 1, no.
f) `cout << ( k + m < j || 3 - j >= k ) << endl;`
ANS: 0, no.
g) `cout << ( !m ) << endl;`
ANS: 0, no.
h) `cout << ( !( j - m ) ) << endl;`
ANS: 1, yes. The inner pair of parentheses are required.
i) `cout << ( !( k > m ) ) << endl;`
ANS: 0, yes. The inner pair of parentheses are required.

2.53 Write a program that prints a table of the binary, octal and hexadecimal equivalents of the decimal numbers in the range 1 through 256. If you are not familiar with these number systems, read Appendix C first.

```
// Exercise 2.53 Solution
// The oct, hex, and dec identifiers are stream manipulators
// like endl that are defined in Chapter 11. The manipulator
// oct causes integers to be output in octal, the manipulator
// hex causes integers to be output in hexadecimal, and the manipulator
// dec causes integers to be output in decimal.
#include <iostream>

using std::cout;
using std::endl;
using std::oct;
using std::hex;
using std::dec;

int main()
{
   cout << "Decimal\t\tBinary\t\t\tOctal\tHexadecimal\n";

   for ( int loop = 1; loop <= 256; ++loop ) {
      cout << dec << loop << "\t\t";

      // Output binary number
      int number = loop;
      cout << ( number == 256 ? '1' : '0' );
      int factor = 256;

      do {
         cout << ( number < factor && number >= ( factor / 2 ) ? '1' : '0' );
         factor /= 2;
         number %= factor;
      } while ( factor > 2 );

      // Output octal and hexadecimal numbers
      cout << '\t' << oct << loop << '\t' << hex << loop << endl;
   }

   return 0;
}
```

```
Decimal          Binary           Octal    Hexadecimal
1                00000000         1        1
2                00000001         2        2
...
245              01111010         365      f5
246              01111011         366      f6
247              01111011         367      f7
248              01111100         370      f8
249              01111100         371      f9
250              01111101         372      fa
251              01111101         373      fb
252              01111110         374      fc
253              01111110         375      fd
254              01111111         376      fe
255              01111111         377      ff
256              10000000         400      100
```

2.54 Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Print a table that shows the value of π approximated by 1 term of this series, by two terms, by three terms, etc. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?

```cpp
// Exercise 2.54 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   long double pi = 0.0, num = 4.0, denom = 1.0;
   long accuracy = 4;    // set decimal accuracy

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "Accuracy set at: " << accuracy
        << "\nterm\t\t  pi\n";

   for ( long loop = 1; loop <= accuracy; ++loop ) {

      if ( loop % 2 != 0 )
         pi += num / denom;
      else
         pi -= num / denom;

      cout << loop << "\t\t" << setprecision( 8 ) << pi << '\n';
      denom += 2.0;
   }

   cout << endl;

   return 0;
}
```

```
Accuracy set at: 400000
term               pi
1               4.00000000
2               2.66666667
3               3.46666667
4               2.89523810
...
236138          3.14158999
236139          3.14159531
236140          3.14159000
...
```

2.55 *(Pythagorean Triples)* A right triangle can have sides that are all integers. The set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Find all Pythagorean triples for `side1`, `side2` and `hypotenuse` all no larger than 500. Use a triple-nested `for`-loop that tries all possibilities. This is an example of "brute force" computing. You will learn in more advanced computer science courses that there are many interesting problems for which there is no known algorithmic approach other than using sheer brute force.

```cpp
// Exercise 2.55 Solution
#include<iostream>

using std::cout;
using std::endl;

int main()
{
   int count = 0;
   long int hyptSquared, sidesSquared;

   for ( long side1 = 1; side1 <= 500; ++side1 ) {

       for ( long side2 = 1; side2 <= 500; ++side2 ) {

          for ( long hypt = 1; hypt <= 500; ++hypt ) {
             hyptSquared = hypt * hypt;
             sidesSquared = side1 * side1 + side2 * side2;

             if ( hyptSquared == sidesSquared ) {
                cout << side1 << '\t' << side2 << '\t'
                     << hypt << '\n';
                ++count;
             }
          }
       }
   }

   cout << "A total of " << count << " triples were found." << endl;

   return 0;
}
```

```
...
476     93      485
480     31      481
480     88      488
480     108     492
480     140     500
483     44      485
A total of 772 triples were found.
```

2.56 A company pays its employees as managers (who receive a fixed weekly salary), hourly workers (who receive a fixed hourly wage for up to the first 40 hours they work and "time-and-a-half," i.e., 1.5 times their hourly wage, for overtime hours worked), commission workers (who receive $250 plus 5.7% of their gross weekly sales), or pieceworkers (who receive a fixed amount of money per item for each of the items they produce—each pieceworker in this company works on only one type of item). Write a program to compute the weekly pay for each employee. You do not know the number of employees in advance. Each type of employee has its own pay code: Managers have paycode 1, hourly workers have code 2, commission workers have code 3 and pieceworkers have code 4. Use a `switch` to compute each employee's pay based on that employee's paycode. Within the `switch`, prompt the user (i.e., the payroll clerk) to enter the appropriate facts your program needs to calculate each employee's pay basd on that employee's paycode.

```cpp
// Exercise 2.56 Solution
#include<iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include<iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   int payCode, managers = 0, hWorkers = 0, cWorkers = 0;
   int pWorkers = 0, pieces;
   double mSalary, hSalary, cSalary, pSalary, hours;
   double pay;

   cout << "Enter paycode (-1 to end): "
        << setiosflags( ios::fixed | ios::showpoint );
   cin >> payCode;

   while ( payCode != -1 ) {
      switch ( payCode ) {
         case 1:
            cout << "Manager selected."
                 << "\nEnter weekly salary: ";
            cin >> mSalary;
            cout << "The manager's pay is $ "
                 << setprecision( 2 ) << mSalary;
            ++managers;
            break;
         case 2:
            cout << "Hourly worker selected.\n"
                 << "Enter the hourly salary: ";
            cin >> hSalary;
            cout << "Enter the total hours worked: ";
            cin >> hours;

            pay = hours > 40.0 ? ( hours - 40 ) * 1.5 * hSalary + hSalary * 40.0:
                                  hSalary * hours;

            cout << "Worker's pay is $ " << setprecision( 2 ) << pay << '\n';
            ++hWorkers;
            break;
         case 3:
            cout << "Commission worker selected.\n"
                 << "Enter gross weekly sales: ";
            cin >> cSalary;
            pay = 250.0 + 0.057 * cSalary;
            cout << "Commission Worker's pay is $ " << setprecision( 2 )
                 << pay << '\n';
            ++cWorkers;
            break;
         case 4:
            cout << "Piece worker selected.\n"
                 << "Enter number of pieces: ";
            cin >> pieces;
            cout << "Enter wage per piece: ";
            cin >> pSalary;
            pay = pieces * pSalary;
            cout << "Piece Worker's pay is $ " << setprecision( 2 )
                 << pay << '\n';
            ++pWorkers;
            break;
         default:
            cout << "Invalid pay code.\n";
            break;
      }

      cout << "\nEnter paycode (-1 to end): ";
      cin >> payCode;
   }

   cout << "\n\nTotal number of managers paid          : "
        << managers
        << "\nTotal number of hourly workers paid    : "
        << hWorkers
        << "\nTotal number of commission workers paid: "
        << cWorkers
        << "\nTotal number of piece workers paid     : "
        << pWorkers << endl;

   return 0;
}
```

```
Enter paycode (-1 to end): 3
Commission worker selected.
Enter gross weekly sales: 4000
Commission Worker's pay is $ 478.00

Enter paycode (-1 to end): 2
Hourly worker selected.
Enter the hourly salary: 4.50
Enter the total hours worked: 20
Worker's pay is $ 90.00

Enter paycode (-1 to end): 4
Piece worker selected.
Enter number of pieces: 50
Enter wage per piece: 3
Piece Worker's pay is $ 150.00

Enter paycode (-1 to end): -1


Total number of managers paid          : 0
Total number of hourly workers paid    : 1
Total number of commission workers paid: 1
Total number of piece workers paid     : 1
```

2.57 *(De Morgan's Laws)* In this chapter, we discussed the logical operators **&&**, **||** and **!**. De Morgan's Laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression **!(***condition1* **&&** *condition2***)** is logically equivalent to the expression **(!***condition1* **|| !***condition2***)**. Also, the expression **!(***condition1* **||** *condition2***)** is logically equivalent to the expression**(!***condition1***&& !***condition2***)**. Use De Morgan's Laws to write equivalent expressions for each of the following, then write a program to show that both the original expression and the new expression in each case are equivalent:

a) `!( x < 5 ) && !( y >= 7 )`
b) `!( a == b ) || !( g != 5 )`
c) `!( ( x <= 8 ) && ( y > 4 ) )`
d) `!( ( i > 4 ) || ( j <= 6 ) )`

```
// Exercise 2.57 Solution
#include<iostream>

using std::cout;
using std::endl;

int main()
{
   int x = 10, y = 1, a = 3, b = 3,
       g = 5, Y = 1, i = 2, j = 9;

   cout << "current variable values are:" << endl
        << "x = " << x << ", y = " << y << ", a = " << a
        << ", b = " << b << endl << "g = " << g << ", Y = "
        << Y << ", i = " << i << ", j = " << j << "\n\n";

   if ((!(x < 5) && !(y >= 7)) && (!((x < 5) || (y >= 7))))
      cout << "!(x < 5) && !(y >= 7) is equivalent to"
           << " !((x < 5) || (y >= 7))" << endl;
   else
      cout << "!(x < 5) && !(y >= 7) is not equivalent to"
           << " !((x < 5) || (y >= 7))" << endl;

   if ((!(a == b) || !(g != 5)) && (!((a == b) && (g != 5))))
```

```
      cout << "!(a == b) || !(g != 5) is equivalent to"
           << " !((a == b) && (g != 5))" << endl;
   else
      cout << "!(a == b) || !(g != 5) is not equivalent to"
           << " !((a == b) && (g != 5))" << endl;

   if (!((x <= 8) && (Y > 4)) && (!((x <= 8) || (Y > 4))))
      cout << "!((x <= 8) && (Y > 4)) is equivalent to"
           << " !((x <= 8) || (Y > 4))" << endl;
   else
      cout << "!((x <= 8) && (Y > 4)) is not equivalent to"
           << " !((x <= 8) || (Y > 4))" << endl;

   if (!((i > 4) || (j <= 6)) && !((i > 4) && (j <= 6)))
      cout << "!((i > 4) || (j <= 6)) is equivalent to"
           << " !((i > 4) && (j <= 6))" << endl;
   else
      cout << "!((i > 4) || (j <= 6)) is not equivalent to"
           << " !((i > 4) && (j <= 6))" << endl;

   return 0;
}
```

```
current variable values are:
x = 10, y = 1, a = 3, b = 3
g = 5, Y = 1, i = 2, j = 9

!(x < 5) && !(y >= 7) is equivalent to !((x < 5) || (y >= 7))
!(a == b) || !(g != 5) is equivalent to !((a == b) && (g != 5))
!((x <= 8) && (Y > 4)) is equivalent to !((x <= 8) || (Y > 4))
!((i > 4) || (j <= 6)) is equivalent to !((i > 4) && (j <= 6))
```

2.58 Write a program that prints the following diamond shape. You may use output statements that print either a single asterisk (`*`) or a single blank. Maximize your use of repetition (with nested **`for`** structures) and minimize the number of output statements.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

```
// Exercise 2.58 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   // top half
   for ( int row = 1; row <= 5; ++row ) {

      for ( int space = 1; space <= 5 - row; ++space )
         cout << ' ';

      for ( int asterisk = 1; asterisk <= 2 * row - 1; ++asterisk )
         cout << '*';

      cout << '\n';
   }

   // bottom half
   for ( row = 4; row >= 1; --row ) {

      for ( int space = 1; space <= 5 - row; ++space )
         cout << ' ';

      for ( int asterisk = 1; asterisk <= 2 * row - 1; ++asterisk )
         cout << '*';

      cout << '\n';
   }

   cout << endl;

   return 0;
}
```

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

2.59 Modify the program you wrote in Exercise 2.58 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

```
// Exercise 2.59 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   int size;

   cout << "Enter an odd number for the diamond size (1-19): \n";
   cin >> size;

   // top half
   for ( int rows = 1; rows <= size -  2; rows += 2 ) {

      for ( int space = ( size - rows ) / 2; space > 0; --space )
         cout << ' ';

      for ( int asterisk = 1; asterisk <= rows; ++asterisk )
         cout << '*';

      cout << '\n';
   }

   // bottom half
   for ( rows = size; rows >= 0; rows -= 2 ) {

      for ( int space = ( size - rows ) / 2; space > 0; --space )
         cout << ' ';

      for ( int asterisk = 1; asterisk <= rows; ++asterisk )
         cout << '*';

      cout << '\n';
   }

   cout << endl;

   return 0;
}
```

```
Enter an odd number for the diamond size (1-19):
15
       *
      ***
     *****
    *******
   *********
  ***********
 *************
***************
 *************
  ***********
   *********
    *******
     *****
      ***
       *
```

2.60 A criticism of the **break** statement and the **continue** statement is that each is unstructured. Actually **break** statements and **continue** statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you would remove any **break** statement from a loop in a program and replace that statement with some structured equivalent. (Hint: The **break** statement leaves a loop from within the body of the loop. The other way to leave is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates "early exit because of a 'break' condition.") Use the technique you developed here to remove the break statement from the program of Fig. 2.26.

```
// Exercise 2.60 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   bool breakOut = false;
   int x;

```

```
    for ( x = 1; x <= 10 && !breakOut; ++x ) {

       if ( x == 4 )
          breakOut = true;

       cout << x << ' ';
    }

    cout << "\nBroke out of loop at x = " << x << endl;

    return 0;
}
```

```
1 2 3 4
Broke out of loop at x = 5
```

2.61 What does the following program segment do?

```
for ( i = 1; i <= 5; i++ ) {
   for ( j = 1; j <= 3; j++ ) {
      for ( k = 1; k <= 4; k++ )
         cout << '*';
      cout << endl;
   }
   cout << endl;
}
```

```
****
****
****

****
****
****

****
****
****

****
****
****

****
****
****
```

2.62 Describe in general how you would remove any **continue** statement from a loop in a program and replace that statement with some structured equivalent. Use the technique you developed here to remove the **continue** statement from the program of Fig. 2.27.

```
// Exercise 2.62 Solution
#include <iostream>

using std::cout;
using std::endl;
```

```

int main()
{
   for ( int x = 1; x <= 10; ++x ) {

      if ( x == 5 )
         ++x;

      cout << x << ' ';
   }

   cout << "\nUsed ++x to skip printing the value 5" << endl;

   return 0;
}
```

```
1 2 3 4 6 7 8 9 10
Used ++x to skip printing the value 5
```

2.63 *("The Twelve Days of Christmas" Song)* Write a program that uses repetition and **switch** structures to print the song "The Twelve Days of Christmas." One **switch** structure should be used to print the day (i.e., "First," "Second," etc.). A separate **switch** structure should be used to print the remainder of each verse.

```
// Exercise 2.63 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   for ( int day = 1; day < 13; day++ ) {
      cout << "On the ";

      switch ( day ) {       // switch for current day
         case 1:
            cout << "first";
            break;
         case 2:
            cout << "second";
            break;
         case 3:
            cout << "third";
            break;
         case 4:
            cout << "fourth";
            break;
         case 5:
            cout << "fifth";
            break;
         case 6:
            cout << "sixth";
            break;
         case 7:
            cout << "seventh";
            break;
         case 8:
            cout << "eighth";
            break;
         case 9:
            cout << "nineth";
            break;
         case 10:
            cout << "tenth";
            break;
         case 11:
            cout << "eleventh";
            break;
         case 12:
            cout << "twelfth";
            break;
      }

      cout << " day of Christmas,\nMy true love sent to me:\n";

      switch ( day ) {     // switch for gifts
         case 12:
            cout << "\tTwelve drummers drumming,\n";
         case 11:
            cout << "\tEleven pipers piping,\n";
         case 10:
            cout << "\tTen lords a-leaping,\n";
         case 9:
            cout << "\tNine ladies dancing,\n";
         case 8:
            cout << "\tEight maids a-milking,\n";
         case 7:
            cout << "\tSeven swans a-swimming,\n";
         case 6:
            cout << "\tSix geese a-laying,\n";
         case 5:
            cout << "\tFive golden rings,\n";
         case 4:
            cout << "\tFour calling birds,\n";
         case 3:
            cout << "\tThree French hens,\n";
         case 2:
            cout << "\tTwo turtle doves, and\n";
         case 1:
            cout << "A partridge in a pear tree.\n\n\n";
      }
   }

   cout << endl;

   return 0;
}
```

```
...
        Three French hens,
        Two turtle doves, and
A partridge in a pear tree.


On the twelfth day of Christmas,
My true love sent to me:
        Twelve drummers drumming,
        Eleven pipers piping,
        Ten lords a-leaping,
        Nine ladies dancing,
        Eight maids a-milking,
        Seven swans a-swimming,
        Six geese a-laying,
        Five golden rings,
        Four calling birds,
        Three French hens,
        Two turtle doves, and
A partridge in a pear tree.
```

Exercise 2.64 corresponds to Section 2.22, "Thinking About Objects."

2.64 Describe in 200 words or less what an automobile is and does. List the nouns and verbs separately. In the text, we stated that each noun might correspond to an object that will need to be built to implement a system, in this case a car. Pick five of the objects you listed, and, for each, list several attributes and several behaviors. Describe briefly how these objects interact wi th one another and other objects in your description. You have just performed several of the key steps in a typical object-oriented des ign.

ANS:

A specific type of vehicle containing 4 wheels, doors, seats, windows, steering wheel, brakes, radio, engine, exhaust system, transmission, axels, windshield, mirrors, etc.
A car can accelerate, decelerate, turn, move forward, move backward, stop, etc.
Wheels:
    Attributes: size, type, tread depth.
    Behaviors: rotate forward, rotate backward.
Doors:
    Attributes: type (passenger, trunk, etc.), open or closed.
    Behaviors: open, close, lock, unlock.
Steering Wheel:
    Attributes: adjustible.
    Behaviors: turn left, turn right, adjust up, adjust down.
Brakes:
    Attributes: pressed or not pressed, pressure of press.
    Behaviors: press, antilock.
Engine:
    Attributes: cylinders, radiator, timing belts, spark plugs, etc.
    Behaviors: accelerate, decelerate, turn on, turn off.
Interactions:
    Person turns the steering wheel which causes the wheels to turn in the appropriate direction.
    Person presses the accelerator pedal which causes the engine revolutions per minute to increase, resulting in a faster rotation of the wheels.
    Person opens door. Person closes door.
    Person releases accelerator pedal which causes engine RPMs to decrease, resulting in a slower rotation of the wheels.
    Person presses brake pedal which causes brakes to be applied to wheels slows the rotation of thewheels.

2.65 *(Peter Minuit Problem)* Legend has it that in 1626 Peter Minuit purchased Manhattan for $24.00 in barter. Did he make a good investment? To answer this question, modify the compound interest program of Fig. 2.21 to begin with a principal of $24.00 and to calculate the amount of interest on deposit if that money had been kept on deposit until this year (374 years through 200 0). Run the program with interest rates of 5%, 6%, 7%, 8%, 9% and 10% to observe the wonders of compound interest.

3

Functions Solutions

Solutions

3.11 Show the value of **`x`** after each of the following statements is performed:

a) **`x = fabs( 7.5 );`**
ANS: 7.5
b) **`x = floor( 7.5 );`**
ANS: 7.0
c) **`x = fabs( 0.0 );`**
ANS: 0.0
d) **`x = ceil( 0.0 );`**
ANS: 0.0
e) **`x = fabs( -6.4 );`**
ANS: 6.4
f) **`x = ceil( -6.4 );`**
ANS: -6.0
g) **`x = ceil( -fabs( -8 + floor( -5.5 ) ) );`**
ANS: -14.0

3.12 A parking garage charges a $2.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour *or part thereof* in excess of three hours. The maximum charge for any given 24-hour period is $10.00. Assume that no car parks for longer than 24 hours at a time. Write a program that will calculate and print the parking charges for each of 3 customers who parked their cars in this garage yesterday. You should enter the hours parked for each customer. Your program should print the results in a neat tabular format and should calculate and print the total of yesterday's receipts. The programshould use the function **`calculateCharges`** to determine the charge for each customer. Your outputs should appear in the following format:

```
Car       Hours       Charge
1           1.5         2.00
2           4.0         2.50
3          24.0        10.00
TOTAL      29.5        14.50
```

```cpp
// Exercise 3.12 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <cmath>

double calculateCharges( double );

main()
{
   double hour, currentCharge, totalCharges = 0.0, totalHours = 0.0;
   int first = 1;

   cout << "Enter the hours parked for 3 cars: ";

   for ( int i = 1; i <= 3; i++ ) {
      cin >> hour;
      totalHours += hour;

      if ( first ) {
         cout << setw( 5 ) << "Car" << setw( 15 ) << "Hours"
              << setw( 15 ) << "Charge\n";
         first = 0;   // prevents this from printing again
      }

      totalCharges += ( currentCharge = calculateCharges( hour ) );
      cout << setiosflags( ios::fixed | ios::showpoint )
           << setw( 3 ) << i << setw( 17 ) << setprecision( 1 ) << hour
           << setw( 15 ) << setprecision( 2 ) << currentCharge << "\n";
   }

   cout << setw( 7 ) << "TOTAL" << setw( 13 ) << setprecision( 1 )
        << totalHours << setw( 15 ) << setprecision( 2 )
        << totalCharges << endl;


   return 0;
}

double calculateCharges( double hours )
{
   double charge;

   if ( hours < 3.0 )
      charge = 2.0;
   else if ( hours < 19.0 )
      charge = 2.0 + .5 * ceil( hours - 3.0 );
   else
      charge = 10.0;

   return charge;
}
```

```
Enter the hours parked for 3 cars: 1 2 3
 Car           Hours         Charge
 1               1.0           2.00
 2               2.0           2.00
 3               3.0           2.00
 TOTAL           6.0           6.00
```

3.13 An application of function **floor** is rounding a value to the nearest integer. The statement

```
y = floor( x + .5 );
```

will round the number **x** to the nearest integer and assign the result to **y**. Write a program that reads several numbers and uses the preceding statement to round each of these numbers to the nearest integer. For each number processed, print both the original number and the rounded number.

```
// Exercise 3.13 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <cmath>

void roundToIntegers( void );

int main()
{
   roundToIntegers();

   return 0;
}

void roundToIntegers( void )
{
   double x, y;

   cout << setiosflags( ios::fixed | ios::showpoint );

   for ( int loop = 1; loop <= 5; loop++ ) {
      cout << "Enter a number: ";
      cin >> x;
      y = floor( x + .5 );
      cout << x << " rounded is " << setprecision( 1 ) << y << endl;
   }
}
```

```
Enter a number: 8.22
8.220000 rounded is 8.0
Enter a number: 7.98
8.0 rounded is 8.0
Enter a number: 4.52
4.5 rounded is 5.0
Enter a number: 6.9999
7.0 rounded is 7.0
Enter a number: 3.345
3.3 rounded is 3.0
```

3.14 Function **floor** can be used to round a number to a specific decimal place. The statement

```
y = floor( x * 10 + .5 ) / 10;
```

rounds **x** to the tenths position (the first position to the right of the decimal point). The statement

```
y = floor( x * 100 + .5 ) / 100;
```

rounds **x** to the hundredths position (the second position to the right of the decimal point). Write a program that defines four functions to round a number **x** in various ways:

a) **roundToInteger(number)**
b) **roundToTenths(number)**
c) **roundToHundredths(number)**
d) **roundToThousandths(number)**

For each value read, your program should print the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth and the number rounded to the nearest thousandth.

```
// Exercise 3.14 Solution
#include <iostream>

using std::cout;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include <cmath>

double roundToInteger( double );
double roundToTenths( double );
double roundToHundreths( double );
double roundToThousandths( double );

int main()
{
   int count;
   double number;

   cout << "How many numbers do you want to process? "
        << setiosflags( ios::fixed );
   cin >> count;

   for ( int i = 0; i < count; ++i ) {
      cout << "\nEnter number: ";
      cin >> number;
      cout << number << " rounded to the nearest integer is:    "
           << setprecision( 0 ) << roundToInteger( number ) << '\n'
           << setiosflags( ios::showpoint )
           << number << " rounded to the nearest tenth is:      "
           << setprecision( 1 ) << roundToTenths( number ) << '\n'
           << number << " rounded to the nearest hundredth is:  "
           << setprecision( 2 ) << roundToHundreths( number ) << '\n'
           << number << " rounded to the nearest thousandth is: "
           << setprecision( 3 ) << roundToThousandths( number ) << '\n'
           << resetiosflags( ios::showpoint );
   }

   return 0;
}

double roundToInteger( double n )
{
   return floor( n + .5 );
}

double roundToTenths( double n )
{
   return floor( n * 10 + .5 ) / 10;
}

double roundToHundreths( double n )
{
   return floor( n * 100 + .5 ) / 100;
}

double roundToThousandths( double n )
{
   return floor( n * 1000 + .5 ) / 1000.0;
}
```

```
How many numbers do you want to process? 1

Enter number: 2.564
2.564000 rounded to the nearest integer is:    3
3. rounded to the nearest tenth is:      2.6
2.6 rounded to the nearest hundredth is:  2.56
2.56 rounded to the nearest thousandth is: 2.564
```

3.15 Answer each of the following questions.

a) What does it mean to choose numbers “at random?”

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the **rand** function useful for simulating games of chance?

ANS: Because it produces a sequence of pseudo-random numbers that when scaled appear to be random.

c) Why would you randomize a program by using **srand**? Under what circumstances is it desirable not to randomize?

ANS: The sequence of numbers produced by the random number generator differ each time function **srand** is called. Not randomizing is useful for debugging purposes—the programmer knows the sequence of numbers.

d) Why is it often necessary to scale and/or shift the values produced by **rand**?

ANS: To produce random values in a specific range.

e) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events such as cars arriving at a toll booth, people arriving in lines, birds arriving at a tree, etc. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specified times.

3.16 Write statements that assign random integers to the variable n in the following ranges:

a) $1 \leq n \leq 2$

ANS: `n = 1 + rand() % 2;`

b) $1 \leq n \leq 100$

ANS: `n = 1 + rand() % 100;`

c) $0 \leq n \leq 9$

ANS: `n = rand() % 10;`

d) $1000 \leq n \leq 1112$

ANS: `n = 1000 + rand() % 13;`

e) $-1 \leq n \leq 1$

ANS: `n = rand() % 3 - 1;`

f) $-3 \leq n \leq 11$

ANS: `n = rand() % 15 - 3;`

3.17 For each of the following sets of integers, write a single statement that will print a number at random from the set.

a) 2, 4, 6, 8, 10.

ANS: `cout << 2 * ( 1 + rand() % 5 ) ) << '\n';`

b) 3, 5, 7, 9, 11.

ANS: `cout << 1 + 2 * ( 1 + rand() % 5 ) ) << '\n';`

c) 6, 10, 14, 18, 22.

ANS: `cout << 6 + 4 * ( rand() % 5 ) << '\n';`

3.18 Write a function **`integerPower( base, exponent )`** that returns the value of

$$base^{exponent}$$

For example, **`integerPower( 3, 4 ) = 3 * 3 * 3 * 3`**. Assume that **`exponent`** is a positive, nonzero integer and that **`base`** is an integer. The function **`integerPower`** should use **`for`** or **`while`** to control the calculation. Do not use any math library functions.

ANS:

```
// Exercise 3.18 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int integerPower( int, int );

int main()
{
   int exp, base;

   cout << "Enter base and exponent: ";
   cin >> base >> exp;
   cout << base << " to the power " << exp << " is: "
        << integerPower( base, exp ) << endl;

   return 0;
}

int integerPower( int b, int e )
{
   int product = 1;

   for ( int i = 1; i <= e; ++i )
      product *= b;

   return product;
}
```

```
Enter base and exponent: 5 2
5 to the power 2 is: 25
```

3.19 Define a function **`hypotenuse`** that calculates the length of the hypotenuse of a right triangle when the other two sides are given. Use this function in a program to determine the length of the hypotenuse for each of the following triangles. The function should take two arguments of type **`double`** and return the hypotenuse as a **`double`**.

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

```
// Exercise 3.19 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <cmath>

double hypotenuse( double, double );

int main()
{
   double side1, side2;

   cout << setiosflags( ios::fixed | ios::showpoint );

   for ( int i = 1; i <= 3; ++i ) {
      cout << "\nEnter 2 sides of right triangle: ";
      cin >> side1 >> side2;
      cout << "Hypotenuse:  " << setprecision( 1 )
           << hypotenuse( side1, side2 ) << endl;
   }

   return 0;
}

double hypotenuse( double s1, double s2 )
{
   return sqrt( s1 * s1 + s2 * s2 );
}
```

```
Enter 2 sides of right triangle: 4 5
Hypotenuse:  6.4

Enter 2 sides of right triangle: 3 4
Hypotenuse:  5.0

Enter 2 sides of right triangle: 12 7
Hypotenuse:  13.9
```

3.20 Write a function `multiple` that determines for a pair of integers whether the second integer is a multiple of the first. The function should take two integer arguments and return `true` if the second is a multiple of the first, `false` otherwise. Use this function in a program that inputs a series of pairs of integers.

```cpp
// Exercise 3.20 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

bool multiple( int, int );

int main()
{
   int x, y;

   for ( int i = 1; i <= 3; ++i ) {
      cout << "Enter two integers: ";
      cin >> x >> y;

      if ( multiple( x, y ) )
         cout << y << " is a multiple of " << x << "\n\n";
      else
         cout << y << " is not a multiple of " << x << "\n\n";
   }

   cout << endl;

   return 0;
}

bool multiple( int a, int b )
{
   return !( b % a );
}
```

```
Enter two integers: 3 4
4 is not a multiple of 3

Enter two integers: 12 3
3 is not a multiple of 12

Enter two integers: 3 12
12 is a multiple of 3
```

3.21 Write a program that inputs a series of integers and passes them one at a time to function **`even`**, which uses the modulus operator to determine whether an integer is even. The function should take an integer argument and return **`true`** if the integer is even and **`false`** otherwise.

```
// Exercise 3.21 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

bool even( int );

int main()
{
   int x;

   for ( int i = 1; i <= 3; ++i ) {
      cout << "Enter an integer: ";
      cin >> x;

      if ( even( x ) )
         cout << x << " is an even integer\n\n";
      else
         cout << x << " is an odd integer\n\n";
   }

   cout << endl;

   return 0;
}

bool even( int a )
{
   return !( a % 2 );
}
```

```
Enter an integer: 8
8 is an even integer

Enter an integer: 3
3 is an odd integer

Enter an integer: 99
99 is an odd integer
```

3.22 Write a function that displays at the left margin of the screen a solid square of asterisks whose side is specified in integer parameter **`side`**. For example, if **`side`** is **`4`**, the function displays

```
****
****
****
****
```

```
// Exercise 3.22 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

void square( int );

int main()
{
   int side;

   cout << "Enter side: ";
   cin >> side;
   cout << '\n';

   square( side );

   cout << endl;

   return 0;
}

void square( int s )
{
   for ( int row = 1; row <= s; ++row ) {

      for ( int col = 1; col <= s; ++col )
         cout << '*';

      cout << '\n';
   }
}
```

```
Enter side: 8

********
********
********
********
********
********
********
********
```

3.23 Modify the function created in Exercise 3.22 to form the square out of whatever character is contained in character parameter **fillCharacter**. Thus, if **side** is **5** and **fillCharacter** is "**#**," then this function should print

```
#####
#####
#####
#####
#####
```

```cpp
// Exercise 3.23 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

void square( int, char );

int main()
{
   int s;
   char c;

   cout << "Enter a character and the side length: ";
   cin >> c >> s;
   cout << '\n';

   square( s, c );

   cout << endl;

   return 0;
}

void square( int side, char fillCharacter )
{
   for ( int row = 1; row <= side; ++row ) {

      for ( int col = 1; col <= side; ++col )
         cout << fillCharacter;

      cout << '\n';
   }
}
```

```
Enter a character and the side length: H 5

HHHHH
HHHHH
HHHHH
HHHHH
HHHHH
```

3.24 Use techniques similar to those developed in Exercises 3.22 and 3.23 to produce a program that graphs a wide range of shapes.

3.25 Write program segments that accomplish each of the following:

a) Calculate the integer part of the quotient when integer **a** is divided by integer **b**.

b) Calculate the integer remainder when integer **a** is divided by integer **b**.

c) Use the program pieces developed in a) and b) to write a function that inputs an integer between **1** and **32767** and prints it as a series of digits, each pair of which is separated by two spaces. For example, the integer **4562** should be printed as

```
4  5  6  2
```

```
// Exercise 3.25 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;

int quotient( int, int );
int remainder( int, int );

int main()
{
   int number, divisor = 10000;

   cout << "Enter an integer between 1 and 32767: ";
   cin >> number;

   cout << "The digits in the number are:\n";

   while ( number >= 1 ) {

      if ( number >= divisor ) {
         cout << setw( 3 ) << quotient( number, divisor );
         number = remainder( number, divisor );
         divisor = quotient( divisor, 10 );
      }
      else
         divisor = quotient( divisor, 10 );
   }

   cout << endl;

   return 0;
}

// Part A: determine quotient using integer division
int quotient( int a, int b )
{
   return a / b;
}

// Part B: determine remainder using the modulus operator
int remainder( int a, int b )
{
   return a % b;
}
```

```
Enter an integer between 1 and 32767: 6758
The digits in the number are:
  6   7   5   8
```

3.26 Write a function that takes the time as three integer arguments (for hours, minutes and seconds), and returns the number of seconds since the last time the clock "struck 12." Use this function to calculate the amount of time in seconds between two time s, both of which are within one 12-hour cycle of the clock.

```
// Exercise 3.26 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

unsigned seconds( unsigned, unsigned, unsigned );

int main()
{
   unsigned hours, minutes, secs, temp;

   cout << "Enter the first time as three integers: ";
   cin >> hours >> minutes >> secs;

   temp = seconds( hours, minutes, secs );

   cout << "Enter the second time as three integers: ";
   cin >> hours >> minutes >> secs;

   cout << "The difference between the times is "
        << seconds( hours, minutes, secs ) - temp
        << " seconds" << endl;

   return 0;
}

unsigned seconds( unsigned h, unsigned m, unsigned s )
{
   return 3600 * ( h >= 12 ? h - 12 : h ) + 60 * m + s;
}
```

```
Enter the first time as three integers: 5 33 45
Enter the second time as three integers: 9 22 8
The difference between the times is 13703 seconds
```

3.27 Implement the following integer functions:

a) Function **`celsius`** returns the Celsius equivalent of a Fahrenheit temperature.

b) Function **`fahrenheit`** returns the Fahrenheit equivalent of a Celsius temperature.

c) Use these functions to write a program that prints charts showing the Fahrenheit equivalents of all Celsius temperatures from 0 to 100 degrees, and the Celsius equivalents of all Fahrenheit temperatures from 32 to 212 degrees. Print the outputs in a neat tabular format that minimizes the number of lines of output while remaining readable.

```
// Exercise 3.27 Solution
#include <iostream>

using std::cout;
using std::endl;

int celcius( int );
int fahrenheit( int );

int main()
{
   cout << "Fahrenheit equivalents of Celcius temperatures:\n"
        << "Celcius\t\tFahrenheit\n";

   for ( int i = 0; i <= 100; ++i )
      cout << i << "\t\t" << fahrenheit( i ) << '\n';

   cout << "\nCelcius equivalents of Fahrenheit temperatures:"
        << "\nFahrenheit\tCelcius\n";

   for ( int j = 32; j <= 212; ++j )
      cout << j << "\t\t" << celcius( j ) << '\n';

   cout << endl;

   return 0;
}

int celcius( int fTemp )
{
   return static_cast< int > ( 5.0 / 9.0 * ( fTemp - 32 ) );
}

int fahrenheit( int cTemp )
{
   return static_cast< int > ( 9.0 / 5.0 * cTemp + 32 );
}
```

```
Fahrenheit equivalents of Celcius temperatures:
Celcius         Fahrenheit
0               32
1               33
2               35
...
Celcius equivalents of Fahrenheit temperatures:
Fahrenheit      Celcius
32              0
33              0
34              1
...
```

3.28 Write a function that returns the smallest of three double-precision, floating-point numbers.

```
// Exercise 3.28 Solution
#include <iostream>
using std::cout;
using std::endl;
using std::cin;

double smallest3( double, double, double );

int main()
{
   double x, y, z;

   cout << "Enter three numbers: ";
   cin >> x >> y >> z;
   cout << "The smallest value is " << smallest3( x, y, z ) << endl;

   return 0;
}

double smallest3( double smallest, double b, double c )
{
   if ( b < smallest && c > smallest )
      return b;
   else if ( c < smallest )
      return c;
   else
      return smallest;
}
```

```
Enter three numbers: 4.3 6.77 9.76
The smallest value is 4.3
```

3.29 An integer number is said to be a *perfect number* if the sum of its factors, including 1 (but not the number itself), is equal to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a function **`perfect`** that determines whether parameter **`number`** is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the factors of each perfect number to confirm that the number is indeed perfect. Challenge the power of your computer by testing numbers much larger than 1000.

```
// Exercise 3.29 Solution
#include <iostream>

using std::cout;
using std::endl;

bool perfect( int );

int main()
{
   cout << "For the integers from 1 to 1000:\n";

   for ( int j = 2; j <= 1000; ++j )
      if ( perfect( j ) )
         cout << j << " is perfect\n";

   cout << endl;

   return 0;
}

bool perfect( int value )
{
   int factorSum = 1;

   for ( int i = 2; i <= value / 2; ++i )
      if ( value % i == 0 )
         factorSum += i;

   return factorSum == value ? true : false;
}
```

```
For the integers from 1 to 1000:
6 is perfect
28 is perfect
496 is perfect
```

3.30 An integer is said to be *prime* if it is divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

a) Write a function that determines whether a number is prime.
b) Use this function in a program that determines and prints all the prime numbers between 1 and 10,000. How many of these 10,000 numbers do you really have to test before being sure that you have found all the primes?
c) Initially, you might think that *n*/2 is the upper limit for which you must test to see whether a number is prime, but you need only go as high as the square root of *n*. Why? Rewrite the program, and run it both ways. Estimate the performance improvement.

```
// Exercise 3.30 Part A Solution
#include <iostream>
using std::cout;

#include <iomanip>

using std::setw;

bool prime( int );

int main()
{
   int count = 0;

   cout << "The prime numbers from 1 to 10000 are:\n";

   for ( int loop = 2; loop <= 10000; ++loop )
      if ( prime( loop ) ) {
         ++count;
         cout << setw( 6 ) << loop;

         if ( count % 10 == 0 )
            cout << '\n';
      }

   return 0;
}

bool prime( int n )
{
   for ( int loop2 = 2; loop2 <= n / 2; loop2++ )
      if ( n % loop2 == 0 )
```

```
            return false;

      return true;
   }
```

```
The prime numbers from 1 to 10000 are:
     2     3     5     7    11    13    17    19    23    29
    31    37    41    43    47    53    59    61    67    71
    73    79    83    89    97   101   103   107   109   113
   127   131   137   139   149   151   157   163   167   173
   179   181   191   193   197   199   211   223   227   229
   233   239   241   251   257   263   269   271   277   281
   283   293   307   311   313   317   331   337   347   349
   353   359   367   373   379   383   389   397   401   409
   419   421   431   433   439   443   449   457   461   463
  ...
  9739  9743  9749  9767  9769  9781  9787  9791  9803  9811
  9817  9829  9833  9839  9851  9857  9859  9871  9883  9887
  9901  9907  9923  9929  9931  9941  9949  9967  9973
```

```
// Exercise 3.30 Part C Solution
#include <iostream>

using std::cout;

#include <iomanip>
using std::setw;

#include <cmath>

bool prime( int n );

int main()
{
   int count = 0;

   cout << "The prime numbers from 1 to 10000 are:\n";

   for ( int j = 2; j <= 10000; ++j )
      if ( prime( j ) ) {
         ++count;
         cout << setw( 5 ) << j;

         if ( count % 10 == 0 )
            cout << '\n';
      }

   return 0;
}

bool prime( int n )
{
   for ( int i = 2; i <= static_cast< int > ( sqrt( n ) ); ++i )
      if ( n % i == 0 )
         return false;

   return true;
}
```

```
The prime numbers from 1 to 10000 are:
     2     3     5     7    11    13    17    19    23    29
    31    37    41    43    47    53    59    61    67    71
    73    79    83    89    97   101   103   107   109   113
   127   131   137   139   149   151   157   163   167   173
   179   181   191   193   197   199   211   223   227   229
   233   239   241   251   257   263   269   271   277   281
   283   293   307   311   313   317   331   337   347   349
   353   359   367   373   379   383   389   397   401   409
   419   421   431   433   439   443   449   457   461   463
  ...
  9739  9743  9749  9767  9769  9781  9787  9791  9803  9811
  9817  9829  9833  9839  9851  9857  9859  9871  9883  9887
  9901  9907  9923  9929  9931  9941  9949  9967  9973
```

3.31 Write a function that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the function should return 1367.

```
// Exercise 3.31 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;
using std::setfill;

int reverseDigits( int );
int width( int );

int main()
{
   int number;

   cout << "Enter a number between 1 and 9999: ";
   cin >> number;

   cout << "The number with its digits reversed is: "
        << setw( ( width( number ) ) ) << setfill( '0' )
        << reverseDigits( number )
        << endl;

   return 0;
}

int reverseDigits( int n )
{
   int reverse = 0, divisor = 1000, multiplier = 1;

   while ( n > 10 ) {

      if ( n >= divisor ) {
         reverse += n / divisor * multiplier;
         n %= divisor;
         divisor /= 10;
         multiplier *= 10;
      }
      else
         divisor /= 10;
   }

   reverse += n * multiplier;
   return reverse;
}

int width( int n )
{
    if ( n /= 1000 )
       return 4;
    else if ( n /= 100 )
       return 3;
    else if ( n /= 10 )
       return 2;
    else
       return 1;
}
```

```
Enter a number between 1 and 9999: 8765
The number with its digits reversed is: 5678
```

3.32 The *greatest common divisor (GCD)* of two integers is the largest integer that evenly divides each of the numbers. Write a function **gcd** that returns the greatest common divisor of two integers.

```
// Exercise 3.32 Solution
#include <iostream>

using std::cout;
using std::cin;

int gcd( int, int );

int main()
{
   int a, b;

   for ( int j = 1; j <= 5; ++j ) {
      cout << "Enter two integers: ";
      cin >> a >> b;
      cout << "The greatest common divisor of " << a << " and "
           << b << " is " <<  gcd( a, b ) << "\n\n";
   }

   return 0;
}

int gcd( int x, int y )
{
   int greatest = 1;

   for ( int i = 2; i <= ( ( x < y ) ? x: y ); ++i )
      if ( x % i == 0 && y % i == 0 )
         greatest = i;

   return greatest;
}
```

```
Enter two integers: 6 8
The greatest common divisor of 6 and 8 is 2

Enter two integers: 789 4
The greatest common divisor of 789 and 4 is 1

Enter two integers: 9999 27
The greatest common divisor of 9999 and 27 is 9

Enter two integers: 73652 8
The greatest common divisor of 73652 and 8 is 4

Enter two integers: 99 11
The greatest common divisor of 99 and 11 is 11
```

3.33 Write a function **qualityPoints** that inputs a student's average and returns 4 if a student's average is 90–100, 3 if the average is 80–89, 2 if the average is 70–79, 1 if the average is 60–69 and 0 if the average is lower than 60.

```
// Exercise 3.33 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int qualityPoints( int );

int main()
{
   int average;

   for ( int loop = 1; loop <= 5; ++loop ) {
      cout << "\nEnter the student's average: ";
      cin >> average;
      cout << average << " on a 4 point scale is "
           << qualityPoints( average ) << '\n';
   }

   cout << endl;
   return 0;
}

int qualityPoints( int average )
{
   if ( average >= 90 )
      return 4;
   else if ( average >= 80 )
      return 3;
   else if ( average >= 70 )
      return 2;
   else if ( average >= 60 )
      return 1;
   else
      return 0;
}
```

```
Enter the student's average: 99
99 on a 4 point scale is 4

Enter the student's average: 72
72 on a 4 point scale is 2

Enter the student's average: 88
88 on a 4 point scale is 3

Enter the student's average: 65
65 on a 4 point scale is 1

Enter the student's average: 33
33 on a 4 point scale is 0
```

3.34 Write a program that simulates coin tossing. For each toss of the coin, the program should print `Heads` or `Tails`. Let the program toss the coin 100 times and count the number of times each side of the coin appears. Print the results. The program should call a separate function `flip` that takes no arguments and returns `0` for tails and `1` for heads. *Note:* If the program realistically simulates the coin tossing, then each side of the coin should appear approximately half the time.

```cpp
// Exercise 3.34 Solution

#include <iostream>

using std::cout;
using std::endl;

#include <cstdlib>
#include <ctime>

int flip( void );

int main()
{
   int headCount = 0, tailCount = 0;

   srand( time( 0 ) );

   for ( int loop = 1; loop <= 100; loop++ ) {

      if ( flip() == 0 ) {
         tailCount++;
         cout << "Tails ";
      }
      else {
         headCount++;
         cout << "Heads ";
      }

      if ( loop % 10 == 0 )
         cout << '\n';
   }

   cout << "\nThe total number of Heads was "
        << headCount << "\nThe total number of Tails was "
        << tailCount << endl;

   return 0;
}
```

```

int flip( void )
{
   return rand() % 2;
}
```

```
Tails Heads Heads Heads Heads Tails Tails Tails Tails Tails
Tails Heads Heads Tails Tails Heads Tails Tails Heads Tails
Heads Tails Heads Heads Tails Tails Tails Heads Tails Heads
Heads Heads Heads Tails Heads Heads Heads Tails Heads Heads
Heads Heads Heads Heads Heads Heads Tails Heads Tails Heads
Heads Tails Tails Heads Heads Heads Heads Heads Tails Tails
Tails Heads Heads Heads Tails Tails Heads Heads Heads Heads
Heads Tails Tails Heads Tails Heads Tails Tails Heads Heads
Heads Tails Heads Tails Tails Tails Heads Tails Heads Heads
Heads Heads Heads Heads Heads Tails Tails Heads Heads Heads

The total number of Heads was 60
The total number of Tails was 40
```

3.35 Computers are playing an increasing role in education. Write a program that will help an elementary school student learn multiplication. Use **rand** to produce two positive one-digit integers. It should then type a question such as:

How much is 6 times 7?

The student then types the answer. Your program checks the student's answer. If it is correct, print **"Very good!"**, and then ask another multiplication question. If the answer is wrong, print **"No. Please try again."** and then let the student try the same question again repeatedly until the student finally gets it right.

```
// Exercise 3.35 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

void multiplication( void );

int main()
{
   srand( time( 0 ) );
   multiplication();
   return 0;
}

void multiplication( void )
{
   int x, y, response = 0;

   cout << "Enter -1 to End.\n";

   while ( response != -1 ) {
      x = rand() % 10;
      y = rand() % 10;

      cout << "How much is " << x << " times " << y << " (-1 to End)? ";
      cin >> response;

      while ( response != -1 && response != x * y ) {
         cout << "No. Please try again.\n? ";
         cin >> response;
      }

      if ( response != -1 )
         cout << "Very good!\n\n";
   }

   cout << "That's all for now. Bye." << endl;
}
```

```
Enter -1 to End.
How much is 4 times 9 (-1 to End)? 36
Very good!

How much is 7 times 0 (-1 to End)? 0
Very good!

How much is 7 times 8 (-1 to End)? 55
No. Please try again.
? 56
Very good!

How much is 5 times 0 (-1 to End)? -1
That's all for now. Bye.
```

3.36 The use of computers in education is referred to as *computer-assisted instruction* (CAI). One problem that develops in CAI environments is student fatigue. This can be eliminated by varying the computer's dialogue to hold the student's attention. Modi fy the program of Exercise 3.35 so the various comments are printed for each correct answer and each incorrect answer as follows:

Responses to a correct answer

```
Very good!
Excellent!
Nice work!
Keep up the good work!
```

Responses to an incorrect answer

```
No. Please try again.
Wrong. Try once more.
Don't give up!
No. Keep trying.
```

Use the random number generator to choose a number from 1 to 4 to select an appropriate response to each answer. Use a `switch` structure to issue the responses.

```
// Exercise 3.36 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

```

```
void correctMessage( void );
void incorrectMessage( void );
void multiplication( void );

int main()
{
   srand( time( 0 ) );
   multiplication();

   return 0;
}

void correctMessage( void )
{
   switch ( rand() % 4 ) {
      case 0:
         cout << "Very good!";
         break;
      case 1:
         cout << "Excellent!";
         break;
      case 2:
         cout << "Nice work!";
         break;
      case 3:
         cout << "Keep up the good work!";
         break;
   }

   cout << "\n\n";
}

void incorrectMessage( void )
{
   switch ( rand() % 4 ) {
      case 0:
         cout << "No. Please try again.";
         break;
      case 1:
         cout << "Wrong. Try once more.";
         break;
      case 2:
         cout << "Don't give up!";
         break;
      case 3:
         cout << "No. Keep trying.";
         break;
   }

   cout << "\n? ";
}

void multiplication( void )
{
   int x, y, response = 0;

   while ( response != -1 ) {
      x = rand() % 10;
      y = rand() % 10;

      cout << "How much is " << x << " times " << y
           << " (-1 to End)? ";
      cin >> response;

      while ( response != -1 && response != x * y ) {
         incorrectMessage();
         cin >> response;
      }

      if ( response != -1 ) {
         correctMessage();
      }
   }

   cout << "That's all for now. Bye." << endl;
}
```

```
Enter -1 to End.
How much is 4 times 9 (-1 to End)? 36
Very good!

How much is 7 times 0 (-1 to End)? 0
Nice Work!

How much is 7 times 8 (-1 to End)? 55
No. Please try again.
? 56
Excellent!

How much is 5 times 0 (-1 to End)? -1
That's all for now. Bye.
```

3.37 More sophisticated computer-aided instruction systems monitor the student's performance over a period of time. The decision to begin a new topic is often based on the student's success with previous topics. Modify the program of Exercise 3.36 to count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage of correct responses. If the percentage is lower than 75 percent, your program should print `"Please ask your instructor for extra help"` and then terminate.

```
// Exercise 3.37 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

void multiplication( void );
void correctMessage( void );
void incorrectMessage( void );

int main()
{
   srand( time( 0 ) );
   multiplication();

   return 0;
}

```

```
void multiplication( void )
{
   int x, y, response, right = 0, wrong = 0;

   for ( int i = 1; i <= 10; ++i ) {
      x = rand() % 10;
      y = rand() % 10;

      cout << "How much is " << x << " times " << y << "? ";
      cin >> response;

      while ( response != x * y ) {
         ++wrong;
         incorrectMessage();
         cin >> response;
      }

      ++right;
      correctMessage();
  }

   if ( static_cast< double > ( right ) / ( right + wrong ) < .75 )
      cout << "Please ask your instructor for extra help.\n";

   cout << "That's all for now. Bye." << endl;
}

void correctMessage( void )
{
   switch ( rand() % 4 ) {
      case 0:
         cout << "Very good!";
         break;
      case 1:
         cout << "Excellent!";
         break;
      case 2:
         cout << "Nice work!";
         break;
      case 3:
         cout << "Keep up the good work!";
         break;
   }

   cout << "\n\n";
}

void incorrectMessage( void )
{
   switch ( rand() % 4 ) {
      case 0:
         cout << "No. Please try again.";
         break;
      case 1:
         cout << "Wrong. Try once more.";
         break;
      case 2:
         cout << "Don't give up!";
         break;
      case 3:
         cout << "No. Keep trying.";
         break;
   }

   cout << "\n? ";
}
```

```
...

How much is 3 times 7? 21
Nice work!

How much is 5 times 9? 45
Very good!

That's all for now. Bye.
```

3.38 Write a program that plays the game of "guess the number" as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1 to 1000. The program then types:

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
```

The player then types a first guess. The program responds with one of the following:

```
1. Excellent! You guessed the number!
   Would you like to play again (y or n)?
2. Too low. Try again.
3. Too high. Try again.
```

If the player's guess is incorrect, your program should loop until the player finally gets the number right. Your program should keep telling the player **Too high** or **Too low** to help the player "zero in" on the correct answer.

```
// Exercise 3.38 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <cstdlib>
#include <ctime>

void guessGame( void );
bool isCorrect( int, int );

int main()
{
   srand( time( 0 ) );
   guessGame();

   return 0;
}

void guessGame( void )
```

```
{
   int answer, guess;
   char response;

   do {
      answer = 1 + rand() % 1000;
      cout << "\nI have a number between 1 and 1000.\n"
           << "Can you guess my number?\nPlease type your"
           << " first guess.\n? ";
      cin >> guess;

      while ( !isCorrect( guess, answer ) )
         cin >> guess;

      cout << "\nExcellent! You guessed the number!\n"
           << "Would you like to play again?\nPlease type (y/n)? ";
      cin >> response;

   } while ( response == 'y' );
}

bool isCorrect( int g, int a )
{
   if ( g == a )
      return true;

   if ( g < a )
      cout << "Too low. Try again.\n? ";
   else
      cout << "Too high. Try again.\n? ";

   return false;
}
```

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
? 375
Too low. Try again.
? 438
Too high. Try again.
? 405
Too high. Try again.
? 380
Too low. Try again.
? 393
Too low. Try again.
? 399
Too high. Try again.
? 396
Too low. Try again.
? 398
Too high. Try again.
? 397

Excellent! You guessed the number!
Would you like to play again?
Please type (y/n)? n
```

3.39 Modify the program of Exercise 3.38 to count the number of guesses the player makes. If the number is 10 or fewer, print **`Either you know the secret or you got lucky!`** If the player guesses the number in 10 tries, then print **`Ahah! You know the secret!`** If the player makes more than 10 guesses, then print **`You should be able to do better!`** Why should it take no more than 10 guesses? Well, with each "good guess" the player should be able to eliminate half of the numbers. Now show why any number from 1 to 1000 can be guessed in 10 or fewer tries.

```
// Exercise 3.39 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <cstdlib>
#include <ctime>

void guessGame( void );
bool isCorrect( int, int );
void display( int );

int main()
{
   srand( time( 0 ) );
   guessGame();

   return 0;
}

void guessGame( void )
{
   int answer, guess, total = 1;
   char response;

   do {
      answer = 1 + rand() % 1000;
      cout << "I have a number between 1 and 1000."
           << "\nCan you guess my number?\nPlease type"
           << " your first guess.\n? ";
      cin >> guess;

      while ( !isCorrect( guess, answer ) ) {
         cin >> guess;
         ++total;
      }

      cout << "\nExcellent! You guessed the number!\n";

      display( total );

      cout << "Would you like to play again?\nPlease type (y/n)? ";
      cin >> response;

   } while ( response == 'y' );
}

bool isCorrect( int g, int a )
{
   if ( g == a )
      return true;

   if ( g < a )
      cout << "Too low. Try again.\n? ";
   else
      cout << "Too high. Try again.\n? ";

   return false;
}

void display( int t )
{
   if ( t < 10 )
      cout << "Either you know the secret or you got lucky!\n";
   else if ( t == 10 )
      cout << "Ahah! You know the secret!\n";
   else
      cout << "You should be able to do better!\n\n";
}
```

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too high. Try again.
? 250
Too low. Try again.
? 375
Too high. Try again.
? 313
Too low. Try again.
? 344
Too low. Try again.
? 360
Too high. Try again.
? 352
Too high. Try again.
? 348
Too low. Try again.
? 350
Too low. Try again.
? 351

Excellent! You guessed the number!
Ahah! You know the secret!
Would you like to play again?
Please type (y/n)? n
Press any key to continue
```

3.40 Write a recursive function **`power( base, exponent )`** that, when invoked, returns

$$base^{exponent}$$

For example, **`power( 3, 4 ) = 3 * 3 * 3 * 3`**. Assume that **`exponent`** is an integer greater than or equal to 1. *Hint:* The recursion step would use the relationship

$$base^{exponent} = base \cdot base^{exponent - 1}$$

and the terminating condition occurs when **`exponent`** is equal to **`1`** because

$$base^{1} = base$$

```
// Exercise 3.40 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

long power( long, long );

int main()
{
   long b, e;

   cout << "Enter a base and an exponent: ";
   cin >> b >> e;
   cout << b << " raised to the " << e << " is "
```

```
         << power( b, e ) << endl;

   return 0;
}

long power( long base, long exponent )
{
   return exponent == 1 ? base : base * power( base, exponent - 1 );
}
```

```
Enter a base and an exponent: 3 4
3 raised to the 4 is 81
```

3.41 The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, …

begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms. a) Write a *nonrecursive* function **fibonacci(n)** that calculates the *n*th Fibonacci number. b) Determine the largest Fibonacci number that can be printed on your system. Modify the program of part a) to use **double** instead of **int** to calculate and return Fibonacci numbers, and use this modified program to repeat part b).

```
// Exercise 3.41 Part A Solution
// NOTE: This exercise was accidently placed in this
// chapter. The solution utilizes ARRAYS which are
// introduced in the next chapter.
#include <iostream>

using std::cout;
using std::endl;

int MAX = 22;    // the maximum number for which the
                 // fibonacci value can be calculated
                 // on 2-byte integer systems

int fibonacci( int );

int main()
{
   for ( int loop = 0; loop <= MAX; ++loop )
      cout << "fibonacci(" << loop << ") = " << fibonacci( loop )
           << "\n";

   cout << endl;
   return 0;
}

int fibonacci( int n )
{
   int fib[ 23 ];

   fib[ 0 ] = 0;
   fib[ 1 ] = 1;

   for ( int j = 2; j <= n; ++j )
      fib[ j ] = fib[ j - 1 ] + fib[ j - 2 ];

   return fib[ n ];
}
```

```
fibonacci(0) = 0
fibonacci(1) = 1
fibonacci(2) = 1
fibonacci(3) = 2
fibonacci(4) = 3
fibonacci(5) = 5
fibonacci(6) = 8
fibonacci(7) = 13
fibonacci(8) = 21
fibonacci(9) = 34
fibonacci(10) = 55
fibonacci(11) = 89
fibonacci(12) = 144
fibonacci(13) = 233
fibonacci(14) = 377
fibonacci(15) = 610
fibonacci(16) = 987
fibonacci(17) = 1597
fibonacci(18) = 2584
fibonacci(19) = 4181
fibonacci(20) = 6765
fibonacci(21) = 10946
fibonacci(22) = 17711
fibonacci(23) = 28658
```

```
// Exercise 3.41 Part B Solution
// NOTE: This exercise was accidently placed in this
// chapter. The solution utiliizes ARRAYS which are
// introduced in the next chapter.
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

double fibonacci( int );

int main()
{
   cout << setiosflags( ios::fixed | ios::showpoint );

   for ( int loop = 0; loop < 100; ++loop )
      cout << setprecision( 1 ) << "fibonacci(" << loop << ") = "
           << fibonacci( loop ) << endl;

   return 0;
}

double fibonacci( int n )
{
   double fib[ 100 ];

   fib[ 0 ] = 0.0;
   fib[ 1 ] = 1.0;

   for ( int j = 2; j <= n; j++ )
      fib[ j ] = fib[ j - 1 ] + fib[ j - 2 ];

   return fib[ n ];
}
```

```
fibonacci(0) = 0.0
fibonacci(1) = 1.0
fibonacci(2) = 1.0
fibonacci(3) = 2.0
fibonacci(4) = 3.0
fibonacci(5) = 5.0
fibonacci(6) = 8.0
fibonacci(7) = 13.0
...
fibonacci(96) = 51680708854858326000.0
fibonacci(97) = 83621143489848426000.0
fibonacci(98) = 135301852344706760000.0
fibonacci(99) = 218922995834555200000.0
```

3.42 *(Towers of Hanoi)* Every budding computer scientist must grapple with certain classic problems. The Towers of Hanoi (see Fig.3.28) is one of the most famous of these. Legend has it that in a temple in the Far East, priests are attempting to move a s tack of disks from one peg to another. The initial stack had 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from this peg to a second peg under the constraints that exactly one disk is moved at a time, and at no time may a larger disk be placed above a smaller disk. A third peg is available for temporarily holding dis ks. Supposedly, the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

Let us assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that will print the precise sequence of peg-to-peg disk transfers.

If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, if we attack the problem with recursion in mind, it immediately becomes tractable. Moving *n* disks can be viewed in terms of moving only *n* - 1 disks (hence, the recursion), as follows:

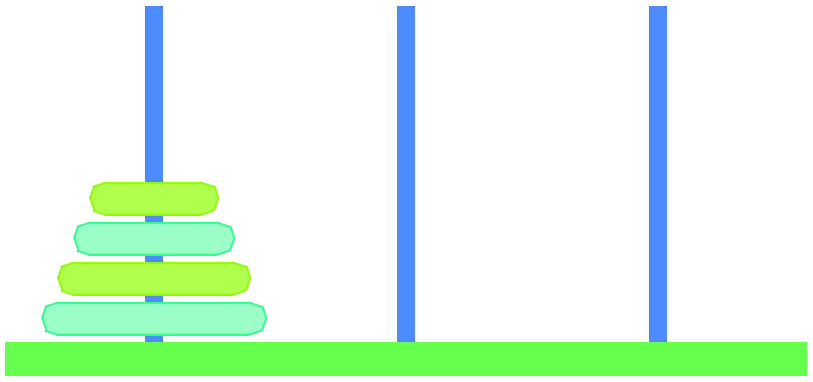

Fig. 3.1 The Towers of Hanoi for the case with four disks.

a) Move *n* - 1 disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
b) Move the last disk (the largest) from peg 1 to peg 3.
c) Move the *n* - 1 disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving *n* = 1 disk, i.e., the base case. This is accomplished by trivially moving the disk without the need for a temporary holding area.

Write a program to solve the Towers of Hanoi problem. Use a recursive function with four parameters:

a) The number of disks to be moved
b) The peg on which these disks are initially threaded
c) The peg to which this stack of disks is to be moved
d) The peg to be used as a temporary holding area

Your program should print the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from peg 1 to peg 3, your program should print the following series of moves:

1 → 3 (This means move one disk from peg 1 to peg 3.)
1 → 2
3 → 2
1 → 3
2 → 1
2→ 3
1→ 3

```
// Exercise 3.42 Solution
#include <iostream>

using std::cout;
using std::cin;

void towers( int, int, int, int );

int main()
{
   int nDisks;

   cout << "Enter the starting number of disks: ";
   cin >> nDisks;
   towers( nDisks, 1, 3, 2 );

   return 0;
}

void towers( int disks, int start, int end, int temp )
{
   if ( disks == 1 ) {
      cout << start << " --> " << end << '\n';
      return;
   }

   // move disks - 1 disks from start to temp
   towers( disks - 1, start, temp, end );

   // move last disk from start to end
   cout << start << " --> " << end << '\n';

   // move disks - 1 disks from temp to end
   towers( disks - 1, temp, end, start );
}
```

```
Enter the starting number of disks: 4
1 --> 2
1 --> 3
2 --> 3
1 --> 2
3 --> 1
3 --> 2
1 --> 2
1 --> 3
2 --> 3
2 --> 1
3 --> 1
2 --> 3
1 --> 2
1 --> 3
2 --> 3
```

3.43 Any program that can be implemented recursively can be implemented iteratively, although sometimes with more difficulty and less clarity. Try writing an iterative version of the Towers of Hanoi. If you succeed, compare your iterative version with the recursive version you developed in Exercise 3.42. Investigate issues of performance, clarity and your ability to demonstrate the correctness of the programs.

3.44 (Visualizing Recursion) It is interesting to watch recursion "in action." Modify the factorial function of Fig. 3.14 to print its local variable and recursive call parameter. For each recursive call, display the outputs on a separate line and add a level of indentation. Do your utmost to make the outputs clear, interesting and meaningful. Your goal here is to design and implement an output format that helps a person understand recursion better. You may want to add such display capabilities to the many other recursion examples and exercises throughout the text.

```
// Exercise 3.44 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

long factorial( long );
void printRecursion( int );

int main()
{
   for ( int i = 0; i <= 10; ++i )
      cout << setw( 3 ) << i << "! = " << factorial( i ) << endl;

   return 0;
}

long factorial( long number )
{
   if ( number <= 1 )
      return 1;
   else {
      printRecursion( number );
      return ( number * factorial( number - 1 ) );
   }
}

```

```
void printRecursion( int n )
{
   cout << "number =" << setw( n ) << n << '\n';
}
```

```
  0! = 1
  1! = 1
number = 2
2! = 2
number =  3
number = 2
3! = 6
number =   4
number =  3
number = 2
4! = 24
number =    5
number =   4
number =  3
number = 2
5! = 120
number =     6
number =    5
number =   4
number =  3
number = 2
...
number =         10
number =        9
number =       8
number =      7
number =     6
number =    5
number =   4
number =  3
number = 2
10! = 3628800
```

3.45 The greatest common divisor of integers **x** and **y** is the largest integer that evenly divides both **x** and **y**. Write a recursive function **gcd** that returns the greatest common divisor of **x** and **y**. The **gcd** of **x** and **y** is defined recursively as follows: If **y** is equal to **0**, then **gcd(x, y)** is **x**; otherwise **gcd(x, y)** is **gcd(y, x % y)**, where **%** is the modulus operator.

```
// Exercise 3.45 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

unsigned gcd( unsigned int, unsigned int );

int main()
{
   unsigned x, y, gcDiv;

   cout << "Enter two integers: ";
   cin >> x >> y;

   gcDiv = gcd( x, y );
   cout << "Greatest common divisor of " << x << " and "
        << y << " is " << gcDiv << endl;

   return 0;
}

unsigned gcd( unsigned xMatch, unsigned yMatch )
{
   return yMatch == 0 ? xMatch : gcd( yMatch, xMatch % yMatch );
}
```

```
Enter two integers: 32727 9
Greatest common divisor of 32727 and 9 is 3
```

3.46 Can **main** be called recursively? Write a program containing a function **main**. Include **static** local variable **count** and initialize it to 1. Postincrement and print the value of **count** each time **main** is called. Compile your program. What happens?

```
// Exercise 3.46 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   static int count = 1;

   ++count;
   cout << count << endl;
   main();

   return 0;
}
```

```
1
2
3
...
12298
12299
12300
12301
12302
12303
12304
...
```

3.47 Exercises 3.35 through 3.37 developed a computer-assisted instruction program to teach an elementary school student multiplication. This exercise suggests enhancements to that program.

a) Modify the program to allow the user to enter a grade-level capability. A grade level of 1 means to use only single-digit numbers in the problems, a grade level of two means to use numbers as large as two digits, etc.

b) Modify the program to allow the user to pick the type of arithmetic problems he or she wishes to study. An option of 1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only, and 5 means to randomly intermix problems of all these types.

```
// Exercise 3.47 Part A Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

int randValue( int );
void multiplication( void );
void correctMessage( void );
void incorrectMessage( void );

int main()
{
   srand( time( 0 ) );
   multiplication();

   return 0;
}

int randValue( int level )
{
   switch ( level ) {
      case 1:
         return rand() % 10;
      case 2:
         return rand() % 100;
      case 3:
         return rand() % 1000;
      default:
         return rand() % 10;
   }
}

void multiplication( void )
{
   int x, y, gradeLevel, right = 0, wrong = 0;
   unsigned int response;

   cout << "Enter the grade-level (1 to 3): ";
   cin >> gradeLevel;

   for ( int i = 1; i <= 10; i++ ) {
      x = randValue( gradeLevel );
      y = randValue( gradeLevel );

      cout << "How much is " << x << " times " << y << "? ";
      cin >> response;

      while ( response != x * y ) {
         ++wrong;
         incorrectMessage();
         cin >> response;
      }
```

```

         ++right;
         correctMessage();
      }

      if ( static_cast< double > ( right ) / ( right + wrong ) < .75 )
         cout << "Please ask your instructor for extra help.\n";

      cout << "That's all for now. Bye." << endl;
   }

   void correctMessage( void )
   {
      switch ( rand() % 4 ) {
         case 0:
            cout << "Very good!";
            break;
         case 1:
            cout << "Excellent!";
            break;
         case 2:
            cout << "Nice work!";
            break;
         case 3:
            cout << "Keep up the good work!";
            break;
      }

      cout << "\n\n";
   }

   void incorrectMessage( void )
   {
      switch ( rand() % 4 ) {
         case 0:
            cout << "No. Please try again.";
            break;
         case 1:
            cout << "Wrong. Try once more.";
            break;
         case 2:
            cout << "Don't give up!";
            break;
         case 3:
            cout << "No. Keep trying.";
            break;
      }

      cout << "\n? ";
   }
```

```
Enter the grade-level (1 to 3): 3
How much is 643 times 462? 297066
Very good!

How much is 763 times 731? 537753
Don't give up!
? 557753
Keep up the good work!

How much is 382 times 120?
...
```

```
// Exercise 3.47 Part B Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

int menu( void );
void arithmetic( void );
void correctMessage( void );
void incorrectMessage( void );

int main()
{
   srand( time( 0 ) );
   arithmetic();

   return 0;
}

int menu( void )
{
   int choice;

   do {
      cout << "Choose type of problem to study."
           << "\nEnter: 1 for addition, 2 for subtraction"
           << "\nEnter: 3 for multiplication, 4 for division"
           << "\nEnter: 5 for a combination of 1 through 4\n? ";
      cin >> choice;

   } while ( choice < 1 || choice > 5 );

   return choice;
}

void incorrectMessage( void )
{
   switch ( rand() % 4 ) {
      case 0:
         cout << "No. Please try again.";
         break;
      case 1:
         cout << "Wrong. Try once more.";
         break;
      case 2:
         cout << "Don't give up!";
         break;
      case 3:
         cout << "No. Keep trying.";
         break;
   }

   cout << "\n? ";
}

void correctMessage( void )
{
   switch ( rand() % 4 ) {
      case 0:
         cout << "Very good!";
         break;
      case 1:
         cout << "Excellent!";
         break;
      case 2:
         cout << "Nice work!";
         break;
      case 3:
         cout << "Keep up the good work!";
         break;
   }

   cout << "\n\n";
}

void arithmetic( void )
{
   int x, y, response, answer, selection, right = 0, wrong = 0;
   int type, problemMix;
   char op;

   selection = menu();
   type = selection;

   for ( int i = 1; i <= 10; ++i ) {
      x = rand() % 10;
      y = rand() % 10;

      if ( selection == 5 ) {
         problemMix = 1 + rand() % 4;
         type = problemMix;
      }

      switch ( type ) {
         case 1:
            op = '+';
            answer = x + y;
            break;
         case 2:              // note negative answers can exist
            op = '-';
            answer = x - y;
            break;
         case 3:
            op = '*';
            answer = x * y;
            break;
         case 4:                  // note this is integer division
            op = '/';

            if ( y == 0 ) {
               y = 1;          // eliminate divide by zero error
               answer = x / y;
            }
            else {
               x *= y;          // create "nice" division
               answer = x / y;
            }

            break;
      }

      cout << "How much is " << x << " " << op << " " << y << "? ";
      cin >> response;

      while ( response != answer ) {
         ++wrong;
         incorrectMessage();
         cin >> response;
      }

      ++right;
      correctMessage();
   }

   if ( static_cast< double > ( right ) / ( right + wrong ) < .75 )
      cout << "Please ask your instructor for extra help.\n";

   cout << "That's all for now. Bye." << endl;
}
```

```
Choose type of problem to study.
Enter: 1 for addition, 2 for subtraction
Enter: 3 for multiplication, 4 for division
Enter: 5 for a combination of 1 through 4
? 5
How much is 7 * 5? 35
Keep up the good work!

How much is 45 / 5? 9
Keep up the good work!

How much is 0 * 7? 0
Excellent!

How much is 1 * 1? 1
Nice work!

How much is 9 * 5? 45
Very good!

How much is 4 + 6?
...
```

3.48 Write function **`distance`** that calculates the distance between two points *(x1, y1)* and *(x2, y2)*. All numbers and return values should be of type **`double`**.

```
// Exercise 3.48 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <cmath>

double distance( double, double, double, double );

int main()
{
   double x1, y1, x2, y2, dist;

   cout << "Enter the first point: ";
   cin >> x1 >> y1;

   cout << "Enter the second point: ";
   cin >> x2 >> y2;

   dist = distance( x1, y1, x2, y2 );

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "Distance between (" << setprecision( 1 ) << x1 << ", "
        << y1 << ") and (" << x2 << ", " << y2 << ") is "
        << dist << endl;

   return 0;
}

double distance( double xOne, double yOne, double xTwo, double yTwo )
{
   return sqrt( pow( xOne - xTwo, 2 ) + pow( yOne - yTwo, 2 ) );
}
```

```
Enter the first point: 8 9
Enter the second point: 0 1
Distance between (8.0, 9.0) and (0.0, 1.0) is 11.3
```

3.49 What is wrong with the following program?

```
// ex03_49.cpp
#include <iostream>

using std::cin;
using std::cout;

int main()
{
   int c;

   if ( ( c = cin.get() ) != EOF ) {
      main();
      cout << c;
   }

   return 0;
}
```

ANS: Standard C++ does not allow recursive calls of `main`, however, on nonstandard compilers, this program would print the characters input in reverse order.

3.50 What does the following program do?

```
// ex03_50.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int mystery( int, int );

int main()
{
   int x, y;

   cout << "Enter two integers: ";
   cin >> x >> y;
   cout << "The result is " << mystery( x, y ) << endl;
   return 0;
}

// Parameter b must be a positive
// integer to prevent infinite recursion
int mystery( int a, int b )
{
   if ( b == 1 )
      return a;
   else
      return a + mystery( a, b - 1 );
}
```

```
Enter two integers: 8 2
The result is 16
```

3.51 After you determine what the program of Exercise 3.50 does, modify the program to function properly after removing the restriction that the second argument be nonnegative.

```
// Exercise 3.51 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int mystery( int, int );

int main()
{
   int x, y;

   cout << "Enter two integers: ";
   cin >> x >> y;
   cout << "The result is " << mystery( x, y ) << endl;
   return 0;
}

int mystery( int a, int b )
{
   if ( ( a < 0 && b < 0 ) || b < 0 ) {
      a *= -1;
      b *= -1;
   }

   return b == 1 ? a : a + mystery( a, b - 1 );
}
```

```
Enter two integers: -8 8
The result is -64
```

3.52 Write a program that tests as many of the math library functions in Fig. 3.2 as you can. Exercise each of these functions by having your program print out tables of return values for a diversity of argument values.

```
// Exercise 3.52 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <cmath>

int main()
{
   cout << "function";      // header

   for ( int a = 1; a < 6; ++a )
      cout << setw( 12 ) << a << ' ';

   cout << setiosflags( ios::fixed | ios::showpoint ) << "\n\nsqrt()  ";

   for ( int b = 1; b < 6; ++b )
      cout << setw( 12 ) << setprecision( 2 ) << sqrt( b ) << ' ';

   cout << "\nexp()   ";

   for ( int c = 1; c < 6; ++c )
      cout << setw( 12 ) << setprecision( 2 ) << exp( c ) << ' ';

   cout << "\nlog()   ";

   for ( int d = 1; d < 6; ++d )
      cout << setw( 12 ) << setprecision( 2 ) << log( d ) << ' ';

   cout << "\nlog10() ";

   for ( int e = 1; e < 6; ++e )
      cout << setw( 12 ) << setprecision( 2 ) << log10( e ) << ' ';

   cout << "\npow(2,x)";

   for ( int f = 1; f < 6; ++f )
      cout << setw( 12 ) << setprecision( 2 ) << pow( 2, f ) << ' ';

   cout << "\n\n\nfunction";      // header

   for ( double g = -1.5; g < 3.0; g += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << g << ' ';

   cout << "\n\n\nfabs()  ";

   for ( double h = -1.5; h < 3.0; h += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << fabs( h ) << ' ';

   cout << "\nceil()  ";

   for ( double i = -1.5; i < 3.0; i += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << ceil( i ) << ' ';

   cout << "\nfloor() ";

   for ( double j = -1.5; j < 3.0; j += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << floor( j ) << ' ';

   cout << "\nsin()   ";

   for ( double k = -1.5; k < 3.0; k += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << sin( k ) << ' ';

   cout << "\ncos()   ";

   for ( double l = -1.5; l < 3.0; l += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << cos( l ) << ' ';

   cout << "\ntan()   ";

   for ( double m = -1.5; m < 3.0; m += 1.1 )
      cout << setw( 12 ) << setprecision( 2 ) << tan( m ) << ' ';

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

```
function           1           2           3           4           5

sqrt()          1.00        1.41        1.73        2.00        2.24
exp()           2.72        7.39       20.09       54.60      148.41
log()           0.00        0.69        1.10        1.39        1.61
log10()         0.00        0.30        0.48        0.60        0.70
pow(2,x)        2.00        4.00        8.00       16.00       32.00


function       -1.50       -0.40        0.70        1.80        2.90


fabs()          1.50        0.40        0.70        1.80        2.90
ceil()         -1.00        0.00        1.00        2.00        3.00
floor()        -2.00       -1.00        0.00        1.00        2.00
sin()          -1.00       -0.39        0.64        0.97        0.24
cos()           0.07        0.92        0.76       -0.23       -0.97
tan()         -14.10       -0.42        0.84       -4.29       -0.25
```

3.53 Find the error in each of the following program segments and explain how to correct it:

a)
```
float cube( float );           // function prototype

double cube( float number )    // function definition
{
   return number * number * number;
}
```
ANS: The function definition defaults to a return type of `int`. Specify a return type of `float` for the definition.

b) `register auto int x = 7;`

ANS: Only one storage class specifier can be used. Either `register` or `auto` must be removed.

c) `int randomNumber = srand();`

ANS: Function `srand` takes an `unsigned` argument and does not return a value. Use `rand` instead of `srand`.

d)
```
float y = 123.45678;
int x;

x = y;
cout << static_cast< float >( x ) << endl;
```
ANS: The assignment of `y` to `x` truncates decimal places.

e)
```
double square( double number )
{
   double number;
   return number * number;
}
```
ANS: Variable `number` is declared twice. Remove the declaration within the `{}`.

f)
```
int sum( int n )
{
   if ( n == 0 )
      return 0;
   else
      return n + sum( n );
}
```
ANS: Infinite recursion. Change operator `+` to operator `-`.

3.54 Modify the craps program of Fig. 3.10 to allow wagering. Package as a function the portion of the program that runs one game of craps. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a `wager`. Use a `while` loop to check that `wager` is less than or equal to `bankBalance` and, if not, prompt the user to reenter `wager` until a valid `wager` is entered. After a correct `wager` is entered, run one game of craps. If the player wins, increase `bankBalance` by `wager` and print the new `bankBalance`. If the player loses, decrease `bankBalance` by `wager`, print the new `bankBalance`, check on whether `bank-`

Balance has become zero and, if so, print the message **"Sorry. You busted!"** As the game progresses, print various messages to create some “chatter” such as **"Oh, you're going for broke, huh?"** or **"Aw cmon, take a chance!"**, or **"You're up big. Now's the time to cash in your chips!"**.

```
// Exercise 3.54 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

enum Status { WON, LOST, CONTINUE };

int rollDice( void );
int craps( void );
void chatter( void );

int main()
{
   int result, wager = 0, bankBalance = 1000;
   char playAgain;

   srand( time( 0 ) );

   do {
      cout << "You have $" << bankBalance
           << " in the bank.\nPlace your wager: ";
      cin >> wager;

      while ( wager <= 0 || wager > bankBalance ) {
         cout << "Please bet a valid amount.\n";
         cin >> wager;
      }

      result = craps();

      if ( result == LOST ) {
         bankBalance -= wager;
         cout << "Your new bank balance is $" << bankBalance << "\n";

         if ( bankBalance == 0 ) {
            cout << "Sorry. You Busted! Thank You For Playing.\n";
            break;
         }
      }
      else {
         bankBalance += wager;
         cout << "Your new bank balance is $" << bankBalance << "\n";
      }

      cout << "Would you like to try your luck again (y/n)? ";
      cin >> playAgain;
   } while ( playAgain == 'y' || playAgain == 'Y' );

   cout << endl;

   return 0;
}

```

```
int rollDice( void )
{
   int die1, die2, workSum;

   die1 = 1 + rand() % 6;
   die2 = 1 + rand() % 6;
   workSum = die1 + die2;
   cout << "Player rolled " << die1 << " + " << die2 << " = "
        << workSum << '\n';

   return workSum;
}

int craps( void )
{
   int gameStatus, sum, myPoint;

   sum = rollDice();

   switch ( sum ) {
      case 7: case 11:
         gameStatus = WON;
         chatter();
         break;
      case 2: case 3: case 12:
         gameStatus = LOST;
         chatter();
         break;
      default:
         gameStatus = CONTINUE;
         myPoint = sum;
         cout << "Point is " << myPoint << '\n';
         chatter();
         break;
   }

   while ( gameStatus == CONTINUE ) {
      chatter();
      sum = rollDice();

      if ( sum == myPoint )
         gameStatus = WON;
      else if ( sum == 7 )
         gameStatus = LOST;
  }

   if ( gameStatus == WON ) {
      cout << "Player wins\n";
      return WON;
   }
   else {
      cout << "Player loses\n";
      return LOST;
   }
}

void chatter( void )
{
   switch ( 1 + rand() % 9 ) {
      case 1:
         cout << "Oh, you're going for broke, huh?";
         break;
      case 2:
         cout << "Aw cmon, take a chance!";
         break;
      case 3:
         cout << "Hey, I think this guy is going to break the bank!!";
         break;
      case 4:
         cout << "You're up big. Now's the time to cash in your chips!";
         break;
      case 5:
         cout << "Way too lucky! Those dice have to be loaded!";
         break;
      case 6:
         cout << "Bet it all! Bet it all!";
         break;
      case 7:
         cout << "Can I borrow a chip?";
         break;
      case 8:
         cout << "Let's try our luck at another table.";
         break;
      case 9:
         cout << "You're a cheat! It is just a matter of time"
              << "\nbefore I catch you!!!";
         break;
   }

   cout << endl;
}
```

```
You have $1000 in the bank.
Place your wager: 1000
Player rolled 5 + 4 = 9
Point is 9
Way too lucky! Those dice have to be loaded!
You're up big. Now's the time to cash in your chips!
Player rolled 5 + 4 = 9
Player wins
Your new bank balance is $2000
Would you like to try your luck again (y/n)? n
```

3.55 Write a C++ program that uses an **inline** function **circleArea** to prompt the user for the radius of a circle and to calculate and print the area of that circle.

```
// Exercise 3.55 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

double pi = 3.14159;   // global variable

inline double circleArea( double r ) { return pi * r * r; }

int main()
{
   double radius;

   cout << "Enter the radius of the circle: ";
   cin >> radius;
   cout << "The area of the circle is " << circleArea( radius ) << endl;

   return 0;
}
```

```
Enter the radius of the circle: 10
The area of the circle is 314.159
```

3.56 Write a complete C++ program with the two alternate functions specified below, of which each simply triples the variable **count** defined in **main**. Then compare and contrast the two approaches. These two functions are

a) Function **tripleCallByValue** that passes a copy of **count** call-by-value, triples the copy and returns the new value.

b) Function **tripleByReference** that passes **count** with true call-by-reference via a reference parameter and triples the original copy of **count** through its alias (i.e., the reference parameter).

```
// Exercise 3.56 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int tripleCallByValue( int );
void tripleByReference( int & );

int main()
{
   int value, &valueRef = value;

   cout << "Enter an integer: ";
   cin >> value;

   cout << "\nValue before call to tripleCallByValue() is: "
        << value << "\nValue returned from tripleCallByValue() is: "
        << tripleCallByValue( value )
        << "\nValue (in main) after tripleCallByValue() is: " << value
        << "\n\nValue before call to tripleByReference() is: "
        << value << '\n';

   tripleByReference( valueRef );

   cout << "Value (in main) after call to tripleByReference() is: "
        << value << endl;

   return 0;
}

int tripleCallByValue( int valueCopy )
{
   return valueCopy *= 3;
}

void tripleByReference( int &aliasRef )
{
   aliasRef *= 3;
}
```

```
Enter an integer: 8

Value before call to tripleCallByValue() is: 8
Value returned from tripleCallByValue() is: 24
Value (in main) after tripleCallByValue() is: 8

Value before call to tripleByReference() is: 8
Value (in main) after call to tripleByReference() is: 24
```

3.57 What is the purpose of the unary scope resolution operator?
ANS: The unary scope resolution operator is used to access a global variable. In particular, the unary scope resolution operator is useful when a global variable needs to be accessed and a local varible has the same name.

3.58 Write a program that uses a function template called `min` to determine the smaller of two arguments. Test the program using integer, character and floating-point number pairs.

```cpp
// Exercise 3.58 Solution
#include <iostream>

using std::cout;
using std::endl;

template < class T >
void min( T value1, T value2 )   // find the smallest value
{
   if ( value1 > value2 )
      cout << value2 << " is smaller than " << value1;
   else
      cout << value1 << " is smaller than " << value2;

   cout << endl;
}

int main()
{
   min( 7, 54 );      // integers
   min( 4.35, 8.46 ); // doubles
   min( 'g', 'T' );   // characters

   return 0;
}
```

```
7 is smaller than 54
4.35 is smaller than 8.46
T is smaller than g
```

3.59 Write a program that uses a function template called `max` to determine the largest of three arguments. Test the program using integer, character and floating-point number pairs.

```cpp
// Exercise 3.59 Solution
#include <iostream>

using std::cout;
using std::endl;

template < class T >
void max( T value1, T value2, T value3 )  // find the largest value
```

```
{
   if ( value1 > value2 && value1 > value3 )
      cout << value1 << " is greater than " << value2
           << " and " << value3;
   else if ( value2 > value1 && value2 > value3 )
      cout << value2 << " is greater than " << value1
           << " and " << value3;
   else
      cout << value3 << " is greater than " << value1
           << " and " << value2;

   cout << endl;
}


int main()
{
   max( 7, 5, 2 );            // integers
   max( 9.35, 8.461, 94.3 );  // doubles
   max( '!', 'T', '$' );      // characters

   return 0;
}
```

```
7 is greater than 5 and 2
94.3 is greater than 9.35 and 8.461
T is greater than ! and $
```

3.60 Determine whether the following program segments contain errors. For each error, explain how it can be corrected. Note: For a particular program segment, it is possible that no errors are present in the segment.

a)
```
template < class A >
int sum( int num1, int num2, int num3 )
{
   return num1 + num2 + num3;
}
```
ANS: The function return type and parameter types should be **`A`**.

b)
```
void printResults( int x, int y )
{
   cout << "The sum is " << x + y << '\n';
   return x + y;
}
```
ANS: The function specifies a **`void`** return type and attempts to return a value. Two possible solutions: (1) change **`void`** to **`int`**. or (2) remove the line **`return x + y;`**.

c)
```
template < A >
A product( A num1, A num2, A num3 )
{
   return num1 * num2 * num3;
}
```
ANS: Keyword **`class`** is needed in the template declaration **`template <class A>`**.

d)
```
double cube( int );
int cube( int );
```
ANS: The signatures are not different. Overloaded functions must have different signatures meaning that the name and parameter list must be different. If only the returns types differ, the compiler generates an error message.

4

Arrays Solutions

Solutions

4.6 Fill in the blanks in each of the following:

a) C++ stores lists of values in __________.

ANS: arrays.

b) The elements of an array are related by the fact that they __________.

ANS: have the same name and type.

c) When referring to an array element, the position number contained within square brackets is called a __________.

ANS: subscript.

d) The names of the four elements of array **`p`** are __________, __________, __________ and __________.

ANS: **`p[ 0 ]`**, **`p[ 1 ]`**, **`p[ 2 ]`**, **`p[ 3 ]`**

e) Naming an array, stating its type and specifying the number of elements in the array is called __________ the array.

ANS: declaring.

f) The process of placing the elements of an array into either ascending or descending order is called __________.

ANS: sorting.

g) In a double-subscripted array, the first subscript (by convention) identifies the __________ of an element, and the second subscript (by convention) identifies the __________ of an element.

ANS: row, column.

h) An *m*-by-*n* array contains __________ rows, __________ columns and __________ elements.

ANS: m, n, m x n.

i) The name of the element in row 3 and column 5 of array **`d`** is __________.

ANS: **`d[ 2 ][ 4 ]`**.

4.7 State which of the following are true and which are false; for those that are false, explain why they are false.

a) To refer to a particular location or element within an array, we specify the name of the array and the value of the particular element.

ANS: False. The name of the array and the subscript of the array are specified.

b) An array declaration reserves space for the array.

ANS: True.

c) To indicate that 100 locations should be reserved for integer array **`p`**, the programmer writes the declaration

```
p[ 100 ];
```

ANS: False. A data type must be specified. An example of a correct definition would be: **`unsigned p[ 100 ];`**.

d) A C++ program that initializes the elements of a 15-element array to zero must contain at least one **for** statement.
ANS: False. The array can be initialized in a declaration with a member initializer list.
e) A C++ program that totals the elements of a double-subscripted array must contain nested **for** statements.
ANS: False. The sum of the elements can be obtained without **for** loops, with one **for** loop, three **for** loops, etc.

4.8 Write C++ statements to accomplish each of the following:
a) Display the value of the seventh element of character array **f**.
ANS: `cout << f[ 6 ] << '\n';`
b) Input a value into element 4 of single-subscripted floating-point array **b**.
ANS: `cin >> b[ 4 ];`
c) Initialize each of the 5 elements of single-subscripted integer array **g** to **8**.
ANS:

```
int g[ 5 ] = { 8, 8, 8, 8, 8 };
```

or

```
for ( int j = 0; j < 5; ++j )
   g[ j ] = 8;
```

d) Total and print the elements of floating-point array **c** of 100 elements.
ANS:

```
for ( int k = 0; k < 5; ++k ) {
    total += c[ k ];  // assume total declared and initalized
    cout << c[ k ] << '\n'
}
```

e) Copy array **a** into the first portion of array **b**. Assume **double a[11], b[34];**
ANS:

```
for ( int i = 0; i < 11; ++i )
   b[ i ] = a[ i ];
```

f) Determine and print the smallest and largest values contained in 99-element floating-point array **w**.
ANS:

```
// assume all variables declared and initialized
for ( int j = 0; j < 99; ++j )
   if ( w[ j ] < smallest )
      smallest = w[ j ];
   else if ( w[ j ] > largest )
      largest = w[ j ];
```

4.9 Consider a 2-by-3 integer array **t**.
a) Write a declaration for **t**.
ANS: `int t[ 2 ][ 3 ];`
b) How many rows does **t** have?
ANS: 2
c) How many columns does **t** have?
ANS: 3
d) How many elements does **t** have?
ANS: 6
e) Write the names of all the elements in the second row of **t**.
ANS: `t[ 1 ][ 0 ]`, `t[ 1 ][ 1 ]`, `t[ 1 ][ 2 ]`
f) Write the names of all the elements in the third column of **t**.
ANS: `t[ 0 ][ 2 ]`, `t[ 1 ][ 2 ]`
g) Write a single statement that sets the element of **t** in row 1 and column 2 to zero.
ANS: `t[ 0 ][ 1 ] = 0;`
h) Write a series of statements that initialize each element of **t** to zero. Do not use a repetition structure.
ANS:

```
t[ 0 ][ 0 ] = 0;
t[ 0 ][ 1 ] = 0;
t[ 0 ][ 2 ] = 0;
t[ 1 ][ 0 ] = 0;
t[ 1 ][ 1 ] = 0;
t[ 1 ][ 2 ] = 0;
```

i) Write a nested **for** structure that initializes each element of **t** to zero.

ANS:
```
for ( int i = 0; i < 2; ++i )
   for ( int j = 0; j < 3; ++j )
      t[ i ][ j ] = 0;
```
j) Write a statement that inputs the values for the elements of **t** from the terminal.

ANS:
```
for ( int r = 0; r < 2; ++r )
   for ( int c = 0; c < 3; ++c )
      t[ r ][ c ] = 0;
```
k) Write a series of statements that determine and print the smallest value in array **t**.

ANS:
```
int smallest = t[ 0 ][ 0 ];
for ( int r = 1; r < 2; ++r )
   for ( int c = 1; c < 3; ++c )
      if ( t[ r ][ c ] < smallest )
         smallest = t[ r ][ c ];
cout << smallest;
```
l) Write a statement that displays the elements of the first row of **t**

ANS: `cout << t[ 0 ][ 0 ] << ' ' << t[ 0 ][ 1 ] << ' ' << t[ 0 ][ 2 ] << '\n';`

m) Write a statement that totals the elements of the fourth column of **t**.

ANS: **t** does not contain a fourth column.

n) Write a series of statements that prints the array **t** in neat, tabular format. List the column subscripts as headings across the top and list the row subscripts at the left of each row.

ANS:
```
cout << "  0     1    2\n";
for ( int r = 0; r < 2; ++r ) {
   cout << r << ' ';

   for ( int c = 0; c < 3; ++c )
      cout << t[ r ][ c ] << "   ";

   cout << '\n';
}
```
4.10 Use a single-subscripted array to solve the following problem. A company pays its salespeople on a commission basis. The salespeople receive $200 per week plus 9 percent of their gross sales for that week. For example, a salesperson who grosses $5000 in sales in a week receives $200 plus 9 percent of $5000, or a total of $650. Write a program (using an array of counters) that determines how many of the salespeople earned salaries in each of the following ranges (assume that each salesperson's salary is truncated to an integer amount):

a) $200–$299
b) $300–$399
c) $400–$499
d) $500–$599
e) $600–$699
f) $700–$799
g) $800–$899
h) $900–$999
i) $1000 and over

```
// Exercise 4.10 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

```

```
#include <iomanip>

using std::setprecision;
using std::setiosflags;

void wages( int [] );
void display( const int [] );

int main()
{
   int salaries[ 11 ] = { 0 };

   cout << setiosflags( ios::fixed | ios::showpoint );
   wages( salaries );
   display( salaries );

   return 0;
}

void wages( int money[] )
{
   double sales, i = 0.09;

   cout << "Enter employee gross sales (-1 to end): ";
   cin >> sales;

   while ( sales != -1 ) {
      double salary = 200.0 + sales * i;
      cout << setprecision( 2 ) << "Employee Commission is $"
           << salary << '\n';

      int x = static_cast< int > ( salary ) / 100;
      ++money[ ( x < 10 ? x : 10 ) ];

      cout << "\nEnter employee gross sales (-1 to end): ";
      cin >> sales;
   }
}

void display( const int dollars[] )
{
   cout << "Employees in the range:";
   for ( int i = 2; i < 10; ++i )
      cout << "\n$" << i << "00-$" << i << "99 : " << dollars[ i ];

   cout << "\nOver $1000: " << dollars[ 10 ] << endl;
}
```

```
Enter employee gross sales (-1 to end): 10000
Employee Commission is $1100.00

Enter employee gross sales (-1 to end): 4235
Employee Commission is $581.15

Enter employee gross sales (-1 to end): 600
Employee Commission is $254.00

Enter employee gross sales (-1 to end): 12500
Employee Commission is $1325.00

Enter employee gross sales (-1 to end): -1
Employees in the range:
$200-$299 : 1
$300-$399 : 0
$400-$499 : 0
$500-$599 : 1
$600-$699 : 0
$700-$799 : 0
$800-$899 : 0
$900-$999 : 0
Over $1000: 2
```

4.11 The bubble sort presented in Fig. 4.16 is inefficient for large arrays. Make the following simple modifications to improve the performance of the bubble sort:

a) After the first pass, the largest number is guaranteed to be in the highest-numbered element of the array; after the second pass, the two highest numbers are "in place," and so on. Instead of making nine comparisons on every pass, modify the bubble sort to make eight comparisons on the second pass, seven on the third pass, and so on.

b) The data in the array may already be in the proper order or near-proper order, so why make nine passes if fewer will suffice? Modify the sort to check at the end of each pass if any swaps have been made. If none have been made, then the data must already be in the proper order, so the program should terminate. If swaps have been made, then at least one more pass is needed.

```cpp
// Exercise 4.11 Part A Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

int main()
{
   const int SIZE = 10;
   int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
   int hold, numberOfComp = 0, comp;

   cout << "Data items in original order\n";
   for ( int i = 0; i < SIZE; ++i )
      cout << setw( 4 ) << a[ i ];

   cout << "\n\n";

   for ( int pass = 1; pass < SIZE; ++pass ) {
      cout << "After pass " << pass - 1 << ": ";

      for ( comp = 0; comp < SIZE - pass; ++comp ) {
         ++numberOfComp;

         if ( a[ comp ] > a[ comp + 1 ] ) {
            hold = a[ comp ];
            a[ comp ] = a[ comp + 1 ];
            a[ comp + 1 ] = hold;
         }

         cout << setw( 3 ) << a[ comp ];
      }

      cout << setw( 3 ) << a[ comp ] << '\n';   // print last array value
   }

   cout << "\nData items in ascending order\n";

   for ( int j = 0; j < SIZE; ++j )
      cout << setw( 4 ) << a[ j ];

   cout << "\nNumber of comparisons = " << numberOfComp << endl;

   return 0;
}
```

```
Data items in original order
   2   6   4   8  10  12  89  68  45  37

After pass 0:   2  4  6  8 10 12 68 45 37 89
After pass 1:   2  4  6  8 10 12 45 37 68
After pass 2:   2  4  6  8 10 12 37 45
After pass 3:   2  4  6  8 10 12 37
After pass 4:   2  4  6  8 10 12
After pass 5:   2  4  6  8 10
After pass 6:   2  4  6  8
After pass 7:   2  4  6
After pass 8:   2  4

Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
Number of comparisons = 45
```

```
// Exercise 4.11 Part B Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

int main()
{
   const int SIZE = 10;
   int a[ SIZE ] = { 6, 4, 2, 8, 10, 12, 37, 45, 68, 89 };
   int hold, numberOfComp = 0, comp;
   bool swapCheck = true;

   cout << "Data items in original order\n";
   for ( int i = 0; i < SIZE; ++i )
      cout << setw( 4 ) << a[ i ];

   cout << "\n\n";

   for ( int pass = 1; pass < SIZE - 1 && swapCheck == true; ++pass ) {
      cout << "After pass " << pass - 1 << ": ";
      swapCheck = false;  // assume no swaps will be made

      for ( comp = 0; comp < SIZE - pass; ++comp ) {
         ++numberOfComp;

         if ( a[ comp ] > a[ comp + 1 ] ) {
            hold = a[ comp ];
            a[ comp ] = a[ comp + 1 ];
            a[ comp + 1 ] = hold;
            swapCheck = true;  // a swap has been made
         }

         cout << setw( 3 ) << a[ comp ];
      }

      cout << setw( 3 ) << a[ comp ] << '\n';   // print last array value
   }

   cout << "\nData items in ascending order\n";

   for ( int q = 0; q < SIZE; ++q )
      cout << setw( 4 ) << a[ q ];

   cout << "\nNumber of comparisons = " << numberOfComp << endl;

   return 0;
}
```

```
Data items in original order
   6   4   2   8  10  12  37  45  68  89

After pass 0:   4  2  6  8 10 12 37 45 68 89
After pass 1:   2  4  6  8 10 12 37 45 68
After pass 2:   2  4  6  8 10 12 37 45

Data items in ascending order
   2   4   6   8  10  12  37  45  68  89
Number of comparisons = 24
```

4.12 Write single statements that perform the following single-subscripted array operations:

a) Initialize the 10 elements of integer array **`counts`** to zero.

ANS: `int counts[ 10 ] = { 0 };`

b) Add 1 to each of the 15 elements of integer array **`bonus`**.

ANS:

```
   for ( int i = 0; i < 15; ++i )
      ++bonus[ i ];
```

c) Read 12 values for **`double`** array **`monthlyTemperatures`** from the keyboard.

ANS:

```
   for ( int p = 0; p < 12; ++p )
      cin >> monthlyTemperatures[ p ];
```

d) Print the 5 values of integer array **`bestScores`** in column format.

ANS:
```
for ( int u = 0; u < 5; ++u )
   cout << bestScores[ u ] << '\t';
```

4.13 Find the error(s) in each of the following statements:

a) Assume that: `char str[ 5 ];`
`cin >> str;    // User types hello`

ANS: Inadequate length. The string input exceeds the valid subscript range.

b) Assume that: `int a[ 3 ];`
`cout << a[ 1 ] << " " << a[ 2 ] << " " << a[ 3 ] << endl;`

ANS: `a[ 3 ]` is not a valid location in the array. `a[ 2 ]` is the last valid location.

c) `double f[ 3 ] = { 1.1, 10.01, 100.001, 1000.0001 };`

ANS: Too many initializers in the initializer list. Only 1, 2, or 3 values may be provided in the initializer list.

d) Assume that: `double d[ 2 ][ 10 ];`
`d[ 1, 9 ] = 2.345;`

ANS: Incorrect syntax array element access. `d[ 1 ][ 9 ]` is the correct syntax.

4.14 Modify the program of Fig. 4.17 so function **mode** is capable of handling a tie for the mode value. Also modify function **median** so the two middle elements are averaged in an array with an even number of elements.

```
// Exercise 4.14 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

void mean( int [], int );
void median( int [], int );
void mode( int [], int [], int, int );

int main()
{
   const int SIZE = 100, MAXFREQUENCY = 10;
   int response[ SIZE ] = { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
                            7, 8, 9, 5, 9, 8, 7, 8, 7, 1,
                            6, 7, 8, 9, 3, 9, 8, 7, 1, 7,
                            7, 8, 9, 8, 9, 8, 9, 7, 1, 9,
                            6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
                            7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
                            5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
                            7, 8, 9, 6, 8, 7, 8, 9, 7, 1,
                            7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
                            4, 5, 6, 1, 6, 5, 7, 8, 7, 9 };
   int frequency[ MAXFREQUENCY ] = { 0 };

   mean( response, SIZE );
   median( response, SIZE );
   mode( frequency, response, SIZE, MAXFREQUENCY );

   return 0;
}

void mean( int answer[], int size )      // mean
{
   int total = 0;

   cout << "******\nMean\n******\n";

   for ( int j = 0; j < size; ++j )
      total += answer[ j ];

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "The mean is the average value of the data items.\n"
        << "The mean is equal to the total of all the data\n"
        << "items divided by the number of data items (" << size << ")."
        << "\nThe mean value for this run is: " << total
        << " / " << size << " = " << setprecision( 2 )
        << static_cast< double > ( total ) / size << "\n\n";
}

void median( int answer[], int size )               // median
{
   int hold;
   bool firstRow = true;

   cout << "\n******\nMedian\n******\n"
        << "The unsorted array of responses is\n";

   for ( int loop = 0; loop < size; loop++ ) {
      if ( loop % 20 == 0 && !firstRow )
         cout << '\n';

      cout << setw( 2 ) << answer[ loop ];
      firstRow = false;
   }

   cout << "\n\n";

   for ( int pass = 0; pass <= size - 2; ++pass )
      for ( int k = 0; k <= size - 2; ++k )
         if ( answer[ k ] > answer[ k + 1 ] ) {
            hold = answer[ k ];
            answer[ k ] = answer[ k + 1 ];
            answer[ k + 1 ] = hold;
         }

   cout << "The sorted array is\n";

   firstRow = true;
   for ( int j = 0; j < size; ++j ) {
      if ( j % 20 == 0 && !firstRow )
         cout << '\n';

      cout << setw( 2 ) << answer[ j ];
      firstRow = false;
   }

   cout << "\n\n";

   if ( size % 2 == 0 )     // even number of elements
      cout << "The median is the average of elements " << ( size + 1 ) / 2
           << " and " << 1 + ( size + 1 ) / 2 << " of the sorted "
           << size << " element array.\nFor this run the median is "
           << setprecision( 1 )
           << static_cast< double >( answer[ ( size + 1 ) / 2 ] +
                                     answer[ ( size + 1 ) / 2 + 1 ] ) / 2
```

```
              << "\n\n";
   else                    // odd number of elements
      cout << "The median is element " << ( size + 1 ) / 2  << " of "
           << "the sorted " << size << " element array.\n"
           << "For this run the median is " << answer[ ( size + 1 ) / 2 - 1 ]
           << "\n\n";
}

void mode( int freq[], int answer[], int aSize, int fSize )
{
   const int SIZE2 = 10;
   int largest = 0, array[ SIZE2 ] = { 0 }, count = 0;

   cout << "\n******\nMode\n******\n";

   for ( int rating = 1; rating < fSize; ++rating )
      freq[ rating ] = 0;

   for ( int loop = 0; loop < aSize; ++loop )
      ++freq[ answer[ loop ] ];

   cout << setw( 13 ) << "Response" << setw( 11 ) << "Frequency" << setw( 19 )
        << "Histogram\n\n" << setw( 55 ) << "1    1    2    2\n"
        << setw( 56 ) << "5    0    5    0    5\n\n";

   for ( int r = 1; r < fSize; ++r ) {
      cout << setw( 8 ) << rating << setw( 11 ) << freq[ r ] << "          ";

      if ( freq[ r ] > largest ) {
         largest = freq[ r ];

         for ( int v = 0; v < SIZE2; ++v )
            array[ v ] = 0;

         array[ r ] = largest;
         ++count;
      }
      else if ( freq[ r ] == largest ) {
         array[ r ] = largest;
         ++count;
      }

      for ( int b = 1; b <= freq[ r ]; b++ )
         cout << '*';

      cout << '\n';
   }

   cout << ( count > 1 ? "\nThe modes are:  " : "\nThe mode is: " );

   for ( int m = 1; m < SIZE2; ++m )
      if ( array[ m ] != 0 )
         cout << m << " with a frequency of " << array[ m ] << "\n\t\t";

   cout << endl;
}
```

```
******
Mean
******
The mean is the average value of the data items.
The mean is equal to the total of all the data
items divided by the number of data items (100).
The mean value for this run is: 662 / 100 = 6.62


******
Median
******
The unsorted array of responses is
 6 7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 1
 6 7 8 9 3 9 8 7 1 7 7 8 9 8 9 8 9 7 1 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 1
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7 9

The sorted array is
 1 1 1 1 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5
 5 5 5 5 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is the average of elements 50 and 51 of the sorted 100 element array.

For this run the median is 7.0


******
Mode
******
     Response  Frequency        Histogram

                                       1    1    2    2
                                  5    0    5    0    5

      10           5           *****
      10           3           ***
      10           4           ****
      10           5           *****
      10           8           ********
      10           9           *********
      10          23           ***********************
      10          23           ***********************
      10          20           ********************

The modes are:  7 with a frequency of 23
                8 with a frequency of 23
```

4.15 Use a single-subscripted array to solve the following problem. Read in 20 numbers, each of which is between 10 and 100, inclusive. As each number is read, print it only if it is not a duplicate of a number already read. Provide for the "worst case" in which all 20 numbers are different. Use the smallest possible array to solve this problem.

```
// Exercise 4.15 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;

int main()
{
   const int SIZE = 20;
   int a[ SIZE ] = { 0 }, subscript = 0, duplicate, value;

   cout << "Enter 20 integers between 10 and 100:\n";

   for ( int i = 0; i < SIZE; ++i ) {
      duplicate = 0;
      cin >> value;

      for ( int j = 0; j < subscript; ++j )
         if ( value == a[ j ] ) {
            duplicate = 1;
            break;
         }

      if ( !duplicate )
         a[ subscript++ ] = value;
   }

   cout << "\nThe nonduplicate values are:\n";

   for ( i = 0; a[ i ] != 0; ++i )
      cout << setw( 4 ) << a[ i ];

   cout << endl;

   return 0;
}
```

```
Enter 20 integers between 10 and 100:
22 56 78 94 22 94 38 10 11 12 22 12 13 14 15 16 17 88 88 77

The nonduplicate values are:
  22  56  78  94  38  10  11  12  13  14  15  16  17  88  77
```

4.16 Label the elements of 3-by-5 double-subscripted array `sales` to indicate the order in which they are set to zero by the following program segment:

```
for ( row = 0; row < 3; row++ )
   for ( column = 0; column < 5; column++ )
      sales[ row ][ column ] = 0;
```

ANS:
```
sales[ 0 ][ 0 ], sales[ 0 ][ 1 ], sales[ 0 ][ 2 ], sales[ 0 ][ 3 ],
sales[ 0 ][ 4 ], sales[ 1 ][ 0 ], sales[ 1 ][ 1 ], sales[ 1 ][ 2 ],
sales[ 1 ][ 3 ], sales[ 1 ][ 4 ], sales[ 2 ][ 0 ], sales[ 2 ][ 1 ],
sales[ 2 ][ 2 ], sales[ 2 ][ 3 ], sales[ 2 ][ 4 ]
```

4.17 Write a program that simulates the rolling of two dice. The program should use **`rand`** to roll the first die and should use **`rand`** again to roll the second die. The sum of the two values should then be calculated. *Note:* Since each die can show an integer value from 1 to 6, then the sum of the two values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums. Figure 4.24 shows the 36 possible combinations of the two dice. Your program should roll the two dice 36,000 times. Use a single-subscripted array to tally the numbers of times each possible sum appears. Print the results in a tabular format. Also, determine if the totals are reasonable (i.e., there are six ways to roll a 7, so approximately one sixth of all the rolls should be 7).

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 4.24 The 36 possible outcomes of rolling two dice.

```cpp
// Exercise 4.17 Solution
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <cstdlib>
#include <ctime>

int main()
{
   const long ROLLS = 36000;
   const int SIZE = 13;
   // array exepected contains counts for the expected
   // number of times each sum occurs in 36 rolls of the dice
   int expected[ SIZE ] = { 0, 0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1 };
   int x, y, sum[ SIZE ] = { 0 };

   srand( time( 0 ) );

   for ( long i = 1; i <= ROLLS; ++i ) {
      x = 1 + rand() % 6;
      y = 1 + rand() % 6;
      ++sum[ x + y ];
   }

   cout << setw( 10 ) << "Sum" << setw( 10 ) << "Total" << setw( 10 )
```

```cpp
        << "Expected" << setw( 10 ) << "Actual\n"
        << setiosflags( ios::fixed | ios::showpoint );

   for ( int j = 2; j < SIZE; ++j )
      cout << setw( 10 ) << j << setw( 10 ) << sum[ j ] << setprecision( 3 )
           << setw( 9 ) << 100.0 * expected[ j ] / 36 << "%" << setprecision(3)
           << setw( 9 ) << 100.0 * sum[ j ] / 36000 << "%\n";

   return 0;
}
```

```
       Sum     Total  Expected   Actual
         2       996    2.778%    2.767%
         3      2015    5.556%    5.597%
         4      3005    8.333%    8.347%
         5      3965   11.111%   11.014%
         6      5003   13.889%   13.897%
         7      5985   16.667%   16.625%
         8      5000   13.889%   13.889%
         9      4035   11.111%   11.208%
        10      3042    8.333%    8.450%
        11      1947    5.556%    5.408%
        12      1007    2.778%    2.797%
```

4.18 What does the following program do?

```cpp
// ex04_18.cpp
#include <iostream>

using std::cout;
using std::endl;

int whatIsThis( int [], int );

int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

   int result = whatIsThis( a, arraySize );

   cout << "Result is " << result << endl;
   return 0;
}

int whatIsThis( int b[], int size )
{
   if ( size == 1 )
      return b[ 0 ];
   else
      return b[ size - 1 ] + whatIsThis( b, size - 1 );
}
```

```
Result is 55
```

4.19 Write a program that runs 1000 games of craps and answers the following questions:

a) How many games are won on the 1st roll, 2nd roll, …, 20th roll, and after the 20th roll?

b) How many games are lost on the 1st roll, 2nd roll, …, 20th roll, and after the 20th roll?
c) What are the chances of winning at craps? (*Note:* You should discover that craps is one of the fairest casino games. What do you suppose this means?)
d) What is the average length of a game of craps?
e) Do the chances of winning improve with the length of the game?

```
// Exercise 4.19 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <cstdlib>
#include <ctime>

int rollDice( void );

int main()
{
   enum Outcome { CONTINUE, WIN, LOSE };
   const int SIZE = 22, ROLLS = 1000;
   int gameStatus, sum, myPoint, roll, length = 0, wins[ SIZE ] = { 0 },
       losses[ SIZE ] = { 0 }, winSum = 0, loseSum = 0;

   srand( time( 0 ) );

   for ( int i = 1; i <= ROLLS; ++i ) {
      sum = rollDice();
      roll = 1;

      switch ( sum ) {
         case 7: case 11:
            gameStatus = WIN;
            break;
         case 2: case 3: case 12:
            gameStatus = LOSE;
            break;
         default:
            gameStatus = CONTINUE;
            myPoint = sum;
            break;
      }

      while ( gameStatus == CONTINUE ) {
         sum = rollDice();
         ++roll;

         if ( sum == myPoint )
            gameStatus = WIN;
         else if ( sum == 7 )
            gameStatus = LOSE;
      }

      if ( roll > 21 )
```

```
         roll = 21;

      if ( gameStatus == WIN ) {
         ++wins[ roll ];
         ++winSum;
      }
      else {
         ++losses[ roll ];
         ++loseSum;
      }
   }

   cout << "Games won or lost after the 20th roll"
        << "\nare displayed as the 21st roll.\n\n";

   for ( int z = 1; z <= 21; ++z )
      cout << setw( 3 ) << wins[ z ] << " games won and " << setw( 3 )
           << losses[ z ] << " games lost on roll " << z << '\n';

   // calculate chances of winning
   cout << setiosflags( ios::fixed | ios::showpoint )
        << "\nThe chances of winning are " << winSum << " / "
        << winSum + loseSum << " = " << setprecision( 2 )
        << 100.0 * winSum / ( winSum + loseSum ) << "%\n";

   // calculate average length of game
   for ( int k = 1; k <= 21; ++k )
      length += wins[ k ] * k + losses[ k ] * k;

   cout << "The average game length is " << setprecision( 2 )
        << length / 1000.0 << " rolls." << endl;

   return 0;
}

int rollDice( void )
{
   int die1, die2, workSum;

   die1 = 1 + rand() % 6;
   die2 = 1 + rand() % 6;
   workSum = die1 + die2;

   return workSum;
}
```

```
Games won or lost after the 20th roll
are displayed as the 21st roll.

225 games won and 102 games lost on roll 1
 72 games won and 106 games lost on roll 2
 63 games won and  86 games lost on roll 3
 40 games won and  64 games lost on roll 4
 28 games won and  32 games lost on roll 5
 21 games won and  37 games lost on roll 6
 12 games won and  26 games lost on roll 7
 13 games won and  11 games lost on roll 8
  9 games won and   8 games lost on roll 9
  2 games won and  10 games lost on roll 10
  3 games won and   9 games lost on roll 11
  2 games won and   6 games lost on roll 12
  1 games won and   0 games lost on roll 13
  1 games won and   3 games lost on roll 14
  2 games won and   0 games lost on roll 15
  1 games won and   1 games lost on roll 16
  0 games won and   1 games lost on roll 17
  1 games won and   0 games lost on roll 18
  0 games won and   0 games lost on roll 19
  0 games won and   1 games lost on roll 20
  1 games won and   0 games lost on roll 21

The chances of winning are 497 / 1000 = 49.70%
The average game length is 3.36 rolls.
```

4.20 (*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. You have been asked to program the new system. You are to write a program to assign seats on each flight of the airline's only plane (capacity: 10 seats).

Your program should display the following menu of alternatives— **`Please type 1 for "First Class"`** and **`Please type 2 for "Economy"`**. If the person types **`1`**, your program should assign a seat in the first class section (seats 1-5). If the person types **`2`**, your program should assign a seat in the economy section (seats 6-10). Your program should print a boarding pass indicating the person's seat number and whether it is in the first class or economy section of the plane.

Use a single-subscripted array to represent the seating chart of the plane. Initialize all the elements of the array to 0 to indcate that all seats are empty. As each seat is assigned, set the corresponding elements of the array to 1 to indicate that the seat i s no longer available.

Your program should, of course, never assign a seat that has already been assigned. When the first class section is full, your program should ask the person if it is acceptable to be placed in the nonsmoking section (and vice versa). If yes, then make the appropriate seat assignment. If no, then print the message **`"Next flight leaves in 3 hours."`**

```cpp
// Exercise 4.20 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cctype>

int main()
{
   const int SEATS = 11;
   int plane[ SEATS ] = { 0 }, people = 0, economy = 1, firstClass = 6,
       choice;
   char response;

   while ( people < 10 ) {
      cout << "\nPlease type 1 for \"firstClass\"\n"
           << "Please type 2 for \"economy\"\n";
      cin >> choice;

      if ( choice == 1 ) {
         if ( !plane[ firstClass ] && firstClass <= 10 ) {
            cout << "Your seat assignment is " << firstClass << ' ';
            plane[ firstClass++ ] = 1;
            ++people;
         }
         else if ( firstClass > 10 && economy <= 5 ) {
            cout << "The firstClass section is full.\n"
                 << "Would you like to sit in the economy"
                 << " section (Y or N)? ";
            cin >> response;

            if ( toupper( response ) == 'Y' ) {
               cout << "Your seat assignment is " << economy << ' ';
               plane[ economy++ ] = 1;
               ++people;
            }
            else
               cout << "Next flight leaves in 3 hours.\n";
         }
         else
            cout << "Next flight leaves in 3 hours.\n";
      }
      else {
         if ( !plane[ economy ] && economy <= 5 ) {
            cout << "Your seat assignment is " << economy << '\n';
            plane[ economy++ ] = 1;
            ++people;
         }
         else if ( economy > 5 && firstClass <= 10 ) {
            cout << "The economy section is full.\n"
                 << "Would you like to sit in the firstClass"
                 << " section (Y or N)? ";
            cin >> response;

            if ( toupper( response ) == 'Y' ) {
               cout << "Your seat assignment is " << firstClass << '\n';
               plane[ firstClass++ ] = 1;
               ++people;
            }
            else
               cout << "Next flight leaves in 3 hours.\n";
         }
         else
            cout << "Next flight leaves in 3 hours.\n";
      }
   }

   cout << "All seats for this flight are sold." << endl;

   return 0;
}
```

```
Please type 1 for "firstClass"
Please type 2 for "economy"
1
...

Please type 1 for "firstClass"
Please type 2 for "economy"
1
Your seat assignment is 9
Please type 1 for "firstClass"
Please type 2 for "economy"
1
Your seat assignment is 10
Please type 1 for "firstClass"
Please type 2 for "economy"
2
Your seat assignment is 5
All seats for this flight are sold.
```

4.21 What does the following program do?

```cpp
// ex04_21.cpp
#include <iostream>

using std::cout;
using std::endl;

void someFunction( int [], int );

int main()
{
   const int arraySize = 10;
   int a[ arraySize ] =
      32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

   cout << "The values in the array are:" << endl;
   someFunction( a, arraySize );
   cout << endl;

   return 0;
}

void someFunction( int b[], int size )
{
   if ( size > 0 ) {
      someFunction( &b[ 1 ], size - 1 );
      cout << b[ 0 ] << "  ";
   }
}
```

```
The values in the array are:
37 60 70 90 14 95 18 64 27 32
```

4.22 Use a double-subscripted array to solve the following problem. A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each different type of product sold. Each slip contains the following:

a) The salesperson number

b) The product number
c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write a program that will read all this information for last month's sales and summarize the total sales by salesperson by product. All totals should be stored in the double-subscripted array **sales**. After processing all the information for last month, print the results in tabular format with each of the columns representing a particular salesperson and each of the rows representing a particular product. Cross total each row to get the total sales of each product for last month; cross total each column to get the total sales by salesperson for last month. Your tabular printout should include these cross totals to the right of the totaled rows and to the bottom of the totaled columns.

```
// Exercise 4.22 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

int main()
{
   const int PEOPLE = 5, PRODUCTS = 6;
   double sales[ PEOPLE ][ PRODUCTS ] = { 0.0 }, value, totalSales,
          productSales[ PRODUCTS ] = { 0.0 };
   int salesPerson, product;

   cout << "Enter the salesperson (1 - 4), product number (1 - 5), "
        << "and total sales.\nEnter -1 for the salesperson"
        << " to end input.\n";
   cin >> salesPerson;

   while ( salesPerson != -1 ) {
      cin >> product >> value;
      sales[ salesPerson ][ product ] += value;
      cin >> salesPerson;
   }

   cout << "\nThe total sales for each salesperson are displayed"
        << " at the end of each row,\n" << "and the total sales for"
        << " each product are displayed at the bottom of each\n"
        << "column.\n " << setw( 12 ) << 1 << setw( 12 ) << 2
        << setw( 12 ) << 3 << setw( 12 ) << 4 << setw( 12 ) << 5 << setw( 13 )
        << "Total\n" << setiosflags( ios::fixed | ios::showpoint );

   for ( int i = 1; i < PEOPLE; ++i ) {
      totalSales = 0.0;
      cout << i;

      for ( int j = 1; j < PRODUCTS; ++j ) {
         totalSales += sales[ i ][ j ];
         cout << setw( 12 ) << setprecision( 2 ) << sales[ i ][ j ];
         productSales[ j ] += sales[ i ][ j ];
      }

      cout << setw( 12 ) << setprecision( 2 ) << totalSales << '\n';
   }

   cout << "\nTotal" << setw( 8 ) << setprecision( 2 ) << productSales[ 1 ];

   for ( int j = 2; j < PRODUCTS; ++j )
      cout << setw( 12 ) << setprecision( 2 ) << productSales[ j ];

   cout << endl;

   return 0;
}
```

```
Enter the salesperson (1 - 4), product number (1 - 5), and total sales.
Enter -1 for the salesperson to end input.
1 1 9.99
3 3 5.99
2 2 4.99
-1

The total sales for each salesperson are displayed at the end of each row,
and the total sales for each product are displayed at the bottom of each
column.
            1           2           3           4           5       Total
1        9.99        0.00        0.00        0.00        0.00        9.99
2        0.00        4.99        0.00        0.00        0.00        4.99
3        0.00        0.00        5.99        0.00        0.00        5.99
4        0.00        0.00        0.00        0.00        0.00        0.00

Total    9.99        4.99        5.99        0.00        0.00
```

4.23 (*Turtle Graphics*) The Logo language, which is particularly popular among personal computer users, made the concept of *turtle graphics* famous. Imagine a mechanical turtle that walks around the room under the control of a C++ program. The turtle holds a pen in one of two positions, up or down. While the pen is down, the turtle traces out shapes as it moves; while the peris up, the turtle moves about freely without writing anything. In this problem, you will simulate the operation of the turtle and creat e a computerized sketchpad as well.

Use a 20-by-20 array **floor** that is initialized to zeros. Read commands from an array that contains them. Keep track of the current position of the turtle at all times and whether the pen is currently up or down. Assume that the turtle always starts at posi-tion 0,0 of the floor with its pen up. The set of turtle commands your program must process are as follows:

Command	Meaning
1	Pen up
2	Pen down
3	Turn right
4	Turn left
5,10	Move forward 10 spaces (or a number other than 10)
6	Print the 20-by-20 array
9	End of data (sentinel)

Suppose that the turtle is somewhere near the center of the floor. The following “program” would draw and print a 12-by-12 square and end with the pen in the up position:

```
2
5,12
3
5,12
```

```
3
5,12
3
5,12
1
6
9
```

As the turtle moves with the pen down, set the appropriate elements of array **floor** to **1**'s. When the **6** command (print) is given, wherever there is a **1** in the array, display an asterisk or some other character you choose. Wherever there is a zero, display a blank. Write a program to implement the turtle graphics capabilities discussed here. Write several turtle graphics programs to draw interesting shapes. Add other commands to increase the power of your turtle graphics language.

```cpp
// Exercise 4.23 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

const int MAXCOMMANDS = 100, SIZE = 20;

int turnRight( int );
int turnLeft( int );
void getCommands( int [][ 2 ] );
void movePen( int, int [][ SIZE ], int, int );
void printArray( const int [][ SIZE ] );

int main()
{
   int floor[ SIZE ][ SIZE ] = { 0 }, command, direction = 0,
       commandArray[ MAXCOMMANDS ][ 2 ] = { 0 }, distance, count = 0;
   bool penDown = false;

   getCommands( commandArray );
   command = commandArray[ count ][ 0 ];

   while ( command != 9 ) {
      switch ( command ) {
         case 1:
            penDown = false;
            break;
         case 2:
            penDown = true;
            break;
         case 3:
            direction = turnRight( direction );
            break;
         case 4:
            direction = turnLeft( direction );
            break;
         case 5:
            distance = commandArray[ count ][ 1 ];
            movePen( penDown, floor, direction, distance );
            break;
         case 6:
            cout << "\nThe drawing is:\n\n";
            printArray( floor );
            break;
      }

      command = commandArray[ ++count ][ 0 ];
   }

   return 0;
}

void getCommands( int commands[][ 2 ] )
{
   int tempCommand;

   cout << "Enter command (9 to end input): ";
   cin >> tempCommand;

   for ( int i = 0; tempCommand != 9 && i < MAXCOMMANDS; ++i ) {
      commands[ i ][ 0 ] = tempCommand;

      if ( tempCommand == 5 ) {
         cin.ignore();   // skip comma
         cin >> commands[ i ][ 1 ];
      }

      cout << "Enter command (9 to end input): ";
      cin >> tempCommand;
   }

   commands[ i ][ 0 ] = 9;  // last command
}

int turnRight( int d )
{
   return ++d > 3 ? 0 : d;
}

int turnLeft( int d )
{
   return --d < 0 ? 3 : d;
}

void movePen( int down, int a[][ SIZE ], int dir, int dist )
{
   static int xPos = 0, yPos = 0;
   int j;  // looping variable

   switch ( dir ) {
      case 0:   // move to the right
         for ( j = 1; j <= dist && yPos + j < SIZE; ++j )
            if ( down )
               a[ xPos ][ yPos + j ] = 1;

         yPos += j - 1;
         break;
      case 1:   // move down
         for ( j = 1; j <= dist && xPos + j < SIZE; ++j )
            if ( down )
               a[ xPos + j ][ yPos ] = 1;

         xPos += j - 1;
         break;
      case 2:   // move to the left
         for ( j = 1; j <= dist && yPos - j >= 0; ++j )
            if ( down )
               a[ xPos ][ yPos - j ] = 1;

         yPos -= j - 1;
         break;
      case 3:   // move up
         for ( j = 1; j <= dist && xPos - j >= 0; ++j )
            if ( down )
               a[ xPos - j ][ yPos ] = 1;

         xPos -= j - 1;
         break;
   }
}

void printArray( const int a[][ SIZE ] )
{
   for ( int i = 0; i < SIZE; ++i ) {
      for ( int j = 0; j < SIZE; ++j )
         cout << ( a[ i ][ j ] ? '*' : ' ' );

      cout << endl;
   }
}
```

```
Enter command (9 to end input): 2
Enter command (9 to end input): 5,6
Enter command (9 to end input): 3
Enter command (9 to end input): 5,12
Enter command (9 to end input): 3
Enter command (9 to end input): 5,12
Enter command (9 to end input): 3
Enter command (9 to end input): 5,12
Enter command (9 to end input): 1
Enter command (9 to end input): 6
Enter command (9 to end input): 9

The drawing is:

*******
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*     *
*******
```

4.24 (*Knight's Tour*) One of the more interesting puzzlers for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. The question is this: Can the chess piece called the knight move around an empty chessboard and touch each of the 64 squares once and only once? We study this intriguing problem in depth here.

The knight makes L-shaped moves (over two in one direction and then over one in a perpendicular direction). Thus, from a square in the middle of an empty chessboard, the knight can make eight different moves (numbered 0 through 7) as shown in Fig. 4.25.

	0	1	2	3	4	5	6	7
0								
1				2		1		
2			3				0	
3					K			
4			4				7	
5				5		6		
6								
7								

Fig. 4.25 The eight possible moves of the knight.

a) Draw an 8-by-8 chessboard on a sheet of paper and attempt a Knight's Tour by hand. Put a **`1`** in the first square you move to, a **`2`** in the second square, a **`3`** in the third, etc. Before starting the tour, estimate how far you think you will get, remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?

b) Now let us develop a program that will move the knight around a chessboard. The board is represented by an 8-by-8 double-subscripted array **`board`**. Each of the squares is initialized to zero. We describe each of the eight possible moves in terms of both their horizontal and vertical components. For example, a move of type 0, as shown in Fig. 4.25, consists of moving two squares horizontally to the right and one square vertically upward. Move 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two single-subscripted arrays, **`horizontal`** and **`vertical`**, as follows:

```
horizontal[ 0 ] = 2
horizontal[ 1 ] = 1
horizontal[ 2 ] = -1
horizontal[ 3 ] = -2
horizontal[ 4 ] = -2
horizontal[ 5 ] = -1
horizontal[ 6 ] = 1
horizontal[ 7 ] = 2

vertical[ 0 ] = -1
vertical[ 1 ] = -2
vertical[ 2 ] = -2
vertical[ 3 ] = -1
vertical[ 4 ] = 1
vertical[ 5 ] = 2
vertical[ 6 ] = 2
vertical[ 7 ] = 1
```

Let the variables **`currentRow`** and **`currentColumn`** indicate the row and column of the knight's current position. To make a move of type **`moveNumber`**, where **`moveNumber`** is between 0 and 7, your program uses the statements

```
currentRow += vertical[ moveNumber ];
currentColumn += horizontal[ moveNumber ];
```

Keep a counter that varies from **`1`** to **`64`**. Record the latest count in each square the knight moves to. Remember to test each potential move to see if the knight has already visited that square, and, of course, test every potential move to make sure that the knight does not land off the chessboard. Now write a program to move the knight around the chessboard. Run the program. How many moves did the knight make?

c) After attempting to write and run a Knight's Tour program, you have probably developed some valuable insights. We will use these to develop a *heuristic* (or strategy) for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic greatly improves the chance of success. You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome, or inaccessible, squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to, so when the board gets congested near the end of the tour, there will be a greater chance of success.

We may develop an "accessibility heuristic" by classifying each of the squares according to how accessible they are and then always moving the knight to the square (within the knight's L-shaped moves, of course) that is most inaccessible. We label a double-subscripted array **`accessibility`** with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, each center square is rated as **`8`**, each corner square is rated as **`2`** and the other squares have accessibility numbers of **`3`**, **`4`** or **`6`** as follows:

```
2  3  4  4  4  4  3  2
3  4  6  6  6  6  4  3
4  6  8  8  8  8  6  4
4  6  8  8  8  8  6  4
4  6  8  8  8  8  6  4
4  6  8  8  8  8  6  4
3  4  6  6  6  6  4  3
2  3  4  4  4  4  3  2
```

Now write a version of the Knight's Tour program using the accessibility heuristic. At any time, the knight should move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in any of the four corners. (*Note:* As the knight moves around the chessboard, your program should reduce the accessibility numbers as more and more squares become occupied. In this way, at any given time during the tour, each available square's accessibility number will remain equal to precisely the number of squares from which that square may be reached.) Run this version of your program. Did you get a full tour? Now modify the program to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?

d) Write a version of the Knight's Tour program which, when encountering a tie between two or more squares, decides what square to choose by looking ahead to those squares reachable from the "tied" squares. Your program should move to the square for which the next move would arrive at a square with the lowest accessibility number.

```
// Exercise 4.24 Part C Solution
// Knight's Tour - access version
// runs one tour
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

const int SIZE = 8;

void clearBoard( int [][ SIZE ] );
void printBoard( const int [][ SIZE ] );
bool validMove( int, int, const int [][ SIZE ] );

int main()
{
   int board[ SIZE ][ SIZE ], currentRow, currentColumn, moveNumber = 0,
       access[ SIZE ][ SIZE ] = { 2, 3, 4, 4, 4, 4, 3, 2,
                                  3, 4, 6, 6, 6, 6, 4, 3,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  3, 4, 6, 6, 6, 6, 4, 3,
                                  2, 3, 4, 4, 4, 4, 3, 2 },

       testRow, testColumn, minRow, minColumn,
       minAccess = 9, accessNumber,
       horizontal[ SIZE ] = { 2, 1, -1, -2, -2, -1, 1, 2 },
       vertical[ SIZE ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
   bool done;

   srand( time( 0 ) );

   clearBoard( board );   // initialize array board
   currentRow = rand() % 8;
   currentColumn = rand() % 8;
   board[ currentRow ][ currentColumn ] = ++moveNumber;
   done = false;

   while ( !done ) {
      accessNumber = minAccess;

      for ( int moveType = 0; moveType < SIZE; ++moveType ) {
         testRow = currentRow + vertical[ moveType ];
         testColumn = currentColumn + horizontal[ moveType ];

         if ( validMove( testRow, testColumn, board ) ) {

            if ( access[ testRow ][ testColumn ] < accessNumber ) {
               accessNumber = access[ testRow ][ testColumn ];
               minRow = testRow;
               minColumn = testColumn;
            }

            --access[ testRow ][ testColumn ];
         }
      }

      if ( accessNumber == minAccess )
         done = true;
      else {
         currentRow = minRow;
         currentColumn = minColumn;
         board[ currentRow ][ currentColumn ] = ++moveNumber;
      }
   }

   cout << "The tour ended with " << moveNumber << " moves.\n";

   if ( moveNumber == 64 )
      cout << "This was a full tour!\n\n";
   else
      cout << "This was not a full tour.\n\n";

   cout << "The board for this test is:\n\n";
   printBoard( board );

   return 0;
```

```
}

void clearBoard( int workBoard[][ SIZE ] )
{
   for ( int row = 0; row < SIZE; ++row )
      for ( int col = 0; col < SIZE; ++col )
         workBoard[ row ][ col ] = 0;
}

void printBoard( const int workBoard[][ SIZE ] )
{
   cout << "   0  1  2  3  4  5  6  7\n";

   for ( int row = 0; row < SIZE; ++row ) {
      cout << row;

      for ( int col = 0; col < SIZE; ++col )
         cout << setw( 3 ) << workBoard[ row ][ col ];

      cout << '\n';
   }

   cout << endl;
}

bool validMove( int row, int column, const int workBoard[][ SIZE ] )
{
   // NOTE: This test stops as soon as it becomes false
   return ( row >= 0 && row < SIZE && column >= 0 && column < SIZE
            && workBoard[ row ][ column ] == 0 );
}
```

```
The tour ended with 64 moves.
This was a full tour!

The board for this test is:

   0  1  2  3  4  5  6  7
0  7 36  5 44  9 34 29 56
1  4 45  8 35 30 55 10 33
2 37  6 43 50 47 32 57 28
3 42  3 46 31 58 49 54 11
4 21 38 51 48 53 60 27 62
5  2 41 20 59 24 63 12 15
6 19 22 39 52 17 14 61 26
7 40  1 18 23 64 25 16 13
```

4.25 (*Knight's Tour: Brute-Force Approaches*) In Exercise 4.24, we developed a solution to the Knight's Tour problem. The approach used, called the "accessibility heuristic," generates many solutions and executes efficiently.

As computers continue increasing in power, we will be able to solve more problems with sheer computer power and relatively unsophisticated algorithms. Let us call this approach "brute force" problem solving.

a) Use random-number generation to enable the knight to walk around the chessboard (in its legitimate L-shaped moves, of course) at random. Your program should run one tour and print the final chessboard. How far did the knight get?

b) Most likely, the preceding program produced a relatively short tour. Now modify your program to attempt 1000 tours. Use a single-subscripted array to keep track of the number of tours of each length. When your program finishes attempting the 1000 tours, it should print this information in neat tabular format. What was the best result?

c) Most likely, the preceding program gave you some “respectable” tours, but no full tours. Now “pull all the stops out” and simply let your program run until it produces a full tour.(*Caution:* This version of the program could run for hours on a powerful computer.) Once again, keep a table of the number of tours of each length, and print this table when the first full tour is found. How many tours did your program attempt before producing a full tour? How much time did it take?

d) Compare the brute-force version of the Knight’s Tour with the accessibility-heuristic version. Which required a more careful study of the problem? Which algorithm was more difficult to develop? Which required more computer power? Could we be certain (in advance) of obtaining a full tour with the accessibility heuristic approach? Could we be certain (in advance) of obtaining a full tour with the brute-force approach? Argue the pros and cons of brute-force problem solving in general.

```cpp
// Exercise 4.25 Part A Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

const int SIZE = 8;

bool validMove( int, int, const int [][ SIZE ] );
void printBoard( const int [][ SIZE ] );

int main()
{
   int currentRow, currentColumn, moveType, moveNumber = 0,
       testRow, testColumn, board[ SIZE ][ SIZE ] = { 0 },
       horizontal[ SIZE ] = { 2, 1, -1, -2, -2, -1, 1, 2 },
       vertical[ SIZE ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
   bool done, goodMove;

   srand( time( 0 ) );

   currentRow = rand() % SIZE;
   currentColumn = rand() % SIZE;
   board[ currentRow ][ currentColumn ] = ++moveNumber;
   done = false;

   while ( !done ) {
      moveType = rand() % SIZE;
      testRow = currentRow + vertical[ moveType ];
      testColumn = currentColumn + horizontal[ moveType ];
      goodMove = validMove( testRow, testColumn, board );

      if ( goodMove ) {
         currentRow = testRow;
         currentColumn = testColumn;
         board[ currentRow ][ currentColumn ] = ++moveNumber;
      }
      else {

         for ( int count = 0; count < SIZE - 1 && !goodMove; ++count ) {
            moveType = ++moveType % SIZE;
            testRow = currentRow + vertical[ moveType ];
            testColumn = currentColumn + horizontal[ moveType ];
            goodMove = validMove( testRow, testColumn, board );

            if ( goodMove ) {
               currentRow = testRow;
               currentColumn = testColumn;
               board[ currentRow ][ currentColumn ] = ++moveNumber;
            }
         }

         if ( !goodMove )
            done = true;
      }

      if ( moveNumber == 64 )
         done = true;
   }

   cout << "The tour has ended with " << moveNumber << " moves.\n";

   if ( moveNumber == 64 )
      cout << "This was a full tour!\n";
   else
      cout << "This was not a full tour.\n";

   cout << "The board for this random test was:\n\n";
   printBoard( board );
   return 0;
}

bool validMove( int row, int column, const int workBoard[][ SIZE ] )
{
   // NOTE: This test stops as soon as it becomes false
   return ( row >= 0 && row < SIZE && column >= 0 && column < SIZE
            && workBoard[ row ][ column ] == 0 );
}

void printBoard( const int board[][ SIZE ] )
{
   cout << "   0  1  2  3  4  5  6  7\n";

   for ( int row = 0; row < SIZE; ++row ) {
      cout << row;

      for ( int col = 0; col < SIZE; ++col )
         cout << setw( 3 ) << board[ row ][ col ];

      cout << '\n';
   }

   cout << endl;
}
```

```
The tour has ended with 30 moves.
This was not a full tour.
The board for this random test was:

    0  1  2  3  4  5  6  7
0 30 27  0 17  0  0 14  0
1  0 18 25 28 15  0  0  0
2 26 29 16 21  0  0  0 13
3  0 24 19  0 11 22  0  0
4  0  1  0 23 20  9 12  7
5  0  0  0 10  0  6  0  0
6  2  0  0  0  4  0  8  0
7  0  0  3  0  0  0  5  0
```

```cpp
// Exercise 4.25 Part B Solution
#include <iostream>

using std::cout;
using std::endl;
#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

const int SIZE = 8, TOURS = 1000, MAXMOVES = 65;

bool validMove( int, int, int, const int [][ SIZE ] );

int main()
{
   int currentRow, currentColumn, moveType, moveNumber, testRow, testColumn,
       moveTotal[ MAXMOVES ] = { 0 }, goodMove, board[ SIZE ][ SIZE ],
       horizontal[ SIZE ] = { 2, 1, -1, -2, -2, -1, 1, 2 },
       vertical[ SIZE ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
   bool done;

   srand( time( 0 ) );

   for ( int i = 0; i < TOURS; ++i ) {
      for ( int row = 0; row < SIZE; ++row )
         for ( int col = 0; col < SIZE; ++col )
            board[ row ][ col ] = 0;

      moveNumber = 0;

      currentRow = rand() % SIZE;
      currentColumn = rand() % SIZE;
      board[ currentRow ][ currentColumn ] = ++moveNumber;
      done = false;

      while ( !done ) {
         moveType = rand() % SIZE;
         testRow = currentRow + vertical[ moveType ];
         testColumn = currentColumn + horizontal[ moveType ];
         goodMove = validMove( testRow, testColumn, moveType, board );

         if ( goodMove ) {
            currentRow = testRow;
            currentColumn = testColumn;
            board[ currentRow ][ currentColumn ] = ++moveNumber;
         }
         else {

            for ( int count = 0; count < SIZE - 1 && !goodMove; ++count ) {
               moveType = ++moveType % SIZE;
               testRow = currentRow + vertical[ moveType ];
               testColumn = currentColumn + horizontal[ moveType ];
               goodMove = validMove( testRow, testColumn, moveType, board );

               if ( goodMove ) {
                  currentRow = testRow;
                  currentColumn = testColumn;
                  board[ currentRow ][ currentColumn ] = ++moveNumber;
               }

            }

            if ( !goodMove )
               done = true;
         }

         if ( moveNumber == 64 )
            done = true;
      }

      ++moveTotal[ moveNumber ];
   }

   for ( int j = 1; j < MAXMOVES; ++j )
      if ( moveTotal[ j ] )
         cout << "There were " << moveTotal[ j ] << " tours of " << j
              << " moves." << endl;

   return 0;
}

bool validMove( int testRow, int testColumn, int moveType,
                const int board[][ SIZE ] )
{
   if ( testRow >= 0 && testRow < SIZE && testColumn >= 0 &&
        testColumn < SIZE )
      return board[ testRow ][ testColumn ] != 0 ? false : true;
   else
      return false;
}
```

```
There were 1 tours of 5 moves.
There were 3 tours of 6 moves.
There were 3 tours of 8 moves.
There were 2 tours of 10 moves.
There were 6 tours of 11 moves.
There were 4 tours of 12 moves.
There were 6 tours of 13 moves.
There were 5 tours of 14 moves.
There were 11 tours of 15 moves.
There were 7 tours of 16 moves.
There were 5 tours of 17 moves.
There were 15 tours of 18 moves.
There were 6 tours of 19 moves.
There were 11 tours of 20 moves.
There were 6 tours of 21 moves.
There were 10 tours of 22 moves.
There were 19 tours of 23 moves.
There were 20 tours of 24 moves.
There were 12 tours of 25 moves.
There were 25 tours of 26 moves.
There were 21 tours of 27 moves.
There were 29 tours of 28 moves.
There were 24 tours of 29 moves.
There were 23 tours of 30 moves.
There were 23 tours of 31 moves.
There were 25 tours of 32 moves.
There were 26 tours of 33 moves.
There were 34 tours of 34 moves.
There were 32 tours of 35 moves.
There were 30 tours of 36 moves.
There were 24 tours of 37 moves.
There were 42 tours of 38 moves.
There were 21 tours of 39 moves.
There were 42 tours of 40 moves.
There were 31 tours of 41 moves.
There were 49 tours of 42 moves.
There were 46 tours of 43 moves.
There were 42 tours of 44 moves.
There were 28 tours of 45 moves.
There were 33 tours of 46 moves.
There were 31 tours of 47 moves.
There were 28 tours of 48 moves.
There were 22 tours of 49 moves.
There were 24 tours of 50 moves.
There were 17 tours of 51 moves.
There were 22 tours of 52 moves.
There were 12 tours of 53 moves.
There were 13 tours of 54 moves.
There were 13 tours of 55 moves.
There were 6 tours of 56 moves.
There were 4 tours of 57 moves.
There were 4 tours of 58 moves.
There were 2 tours of 59 moves.
```

4.26 (*Eight Queens*) Another puzzler for chess buffs is the Eight Queens problem. Simply stated: Is it possible to place eight queens on an empty chessboard so that no queen is "attacking" any other, i.e., no two queens are in the same row, the same column, or along the same diagonal? Use the thinking developed in Exercise 4.24 to formulate a heuristic for solving the Eight Queens problem. Run your program. (*Hint:* It is possible to assign a value to each square of the chessboard indicating how many squares of an empty chessboard are "eliminated" if a queen is placed in that square. Each of the corners would be assigned the value 22, as in Fig.

4.26.) Once these “elimination numbers” are placed in all 64 squares, an appropriate heuristic might be: Place the next queen inthe square with the smallest elimination number. Why is this strategy intuitively appealing?

```
* * * * * * * *
* *
*   *
*     *
*       *
*         *
*           *
*             *
```

Fig. 4.26 The 22 squares eliminated by placing a queen in the upper-left corner.

4.27 (*Eight Queens: Brute-Force Approaches*) In this exercise, you will develop several brute-force approaches to solving the Eight Queens problem introduced in Exercise 4.26.

a) Solve the Eight Queens exercise, using the random brute-force technique developed in Exercise 4.25.
b) Use an exhaustive technique, i.e., try all possible combinations of eight queens on the chessboard.
c) Why do you suppose the exhaustive brute-force approach may not be appropriate for solving the Knight’s Tour problem?
d) Compare and contrast the random brute-force and exhaustive brute-force approaches in general.

```
// Exercise 4.27 Part A Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <ctime>
#include <cstdlib>

bool queenCheck( const char [][ 8 ], int, int );
void placeQueens( char [][ 8 ] );
void printBoard( const char [][ 8 ] );
void xConflictSquares( char [][ 8 ], int, int );
void xDiagonals( char [][ 8 ], int, int );
bool availableSquare( const char [][ 8 ] );
inline int validMove( const char board[][ 8 ], int row, int col )
   { return ( row >= 0 && row < 8 && col >= 0 && col < 8 ); }

int main()
{
   char board [ 8 ][ 8 ] = { '\0' };

   srand( time( 0 ) );

   placeQueens( board );
   printBoard( board );

   return 0;
}

bool availableSquare( const char board[][ 8 ] )
{
   for ( int row = 0; row < 8; ++row )
```

```
      for ( int col = 0; col < 8; ++col )
         if ( board[ row ][ col ] == '\0' )
            return false;  // at least one open square is available

   return true;  // no available squares
}

void placeQueens( char board[][ 8 ] )
{
   const char QUEEN = 'Q';
   int rowMove, colMove, queens = 0;
   bool done = false;

   while ( queens < 8 && !done ) {
      rowMove = rand() % 8;
      colMove = rand() % 8;

      if ( queenCheck( board, rowMove, colMove ) ) {
         board[ rowMove ][ colMove ] = QUEEN;
         xConflictSquares( board, rowMove, colMove );
         ++queens;
      }

      done = availableSquare( board );
   }
}

void xConflictSquares( char board[][ 8 ], int row, int col )
{
    for ( int loop = 0; loop < 8; ++loop ) {
       // place an '*' in the row occupied by the queen
       if ( board[ row ][ loop ] == '\0' )
          board[ row ][ loop ] = '*';

       // place an '*' in the col occupied by the queen
       if ( board[ loop ][ col ] == '\0' )
          board[ loop ][ col ] = '*';
    }

    // place an '*' in the diagonals occupied by the queen
    xDiagonals( board, row, col );
}

bool queenCheck( const char board[][ 8 ], int row, int col )
{
   int r = row, c = col;

   // check row and column for a queen
   for ( int d = 0; d < 8; ++d )
      if ( board[ row ][ d ] == 'Q' || board[ d ][ col ] == 'Q' )
         return false;

   // check upper left diagonal for a queen
   for ( int e = 0; e < 8 && validMove( board, --r, --c ); ++e )
      if ( board[ r ][ c ] == 'Q' )
         return false;

   r = row;
   c = col;
   // check upper right diagonal for a queen
   for ( int f = 0; f < 8 && validMove( board, --r, ++c ); ++f )
      if ( board[ r ][ c ] == 'Q' )
```

```
            return false;

      r = row;
      c = col;
      // check lower left diagonal for a queen
      for ( int g = 0; g < 8 && validMove( board, ++r, --c ); ++g )
         if (board[ r ][ c ] == 'Q' )
            return false;

      r = row;
      c = col;
      // check lower right diagonal for a queen
      for ( int h = 0; h < 8 && validMove( board, ++r, ++c ); ++h )
         if ( board[ r ][ c ] == 'Q' )
            return false;

      return true;  // no queen in conflict
}

void xDiagonals( char board[][ 8 ], int row, int col )
{
   int r = row, c = col;

   // upper left diagonal
   for ( int a = 0; a < 8 && validMove( board, --r, --c ); ++a )
      board[ r ][ c ] = '*';

   r = row;
   c = col;
   // upper right diagonal
   for ( int b = 0; b < 8 && validMove( board, --r, ++c ); ++b )
      board[ r ][ c ] = '*';

   r = row;
   c = col;
   // lower left diagonal
   for ( int d = 0; d < 8 && validMove( board, ++r, --c ); ++d )
      board[ r ][ c ] = '*';

   r = row;
   c = col;
   // lower right diagonal
   for ( int e = 0; e < 8 && validMove( board, ++r, ++c ); ++e )
      board[ r ][ c ] = '*';
}

void printBoard( const char board[][ 8 ] )
{
   int queens = 0;

   // header for columns
   cout << "   0 1 2 3 4 5 6 7\n";

   for ( int r = 0; r < 8; ++r ) {
      cout << setw( 2 ) << r << ' ';

      for ( int c = 0; c < 8; ++c ) {
         cout << board[ r ][ c ] << ' ';

         if ( board[ r ][ c ] == 'Q' )
            ++queens;
      }

      cout << '\n';
   }

   if ( queens == 8 )
      cout << "\nEight Queens were placed on the board!" << endl;
   else
      cout << '\n' << queens << " Queens were placed on the board." << endl;
}
```

```
   0 1 2 3 4 5 6 7
 0 * * * * Q * * *
 1 * Q * * * * * *
 2 * * * * * Q * *
 3 * * Q * * * * *
 4 * * * * * * * *
 5 * * * * * * * Q
 6 Q * * * * * * *
 7 * * * Q * * * *

7 Queens were placed on the board.
```

4.28 (*Knight's Tour: Closed-Tour Test*) In the Knight's Tour, a full tour occurs when the knight makes 64 moves touching each square of the chess board once and only once. A closed tour occurs when the 64th move is one move away from the location in which the knight started the tour. Modify the Knight's Tour program you wrote in Exercise 4.24 to test for a closed tour if a fu ll tour has occurred.

```
// Exercise 4.28 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

const int SIZE = 8;

void clearBoard( int [][ SIZE ] );
void printBoard( const int [][ SIZE ] );
bool validMove( int, int, const int [][ SIZE ] );

int main()
{
   int board[ SIZE ][ SIZE ], firstMoveRow, firstMoveCol,
       access[ SIZE ][ SIZE ] = { 2, 3, 4, 4, 4, 4, 3, 2,
                                  3, 4, 6, 6, 6, 6, 4, 3,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  4, 6, 8, 8, 8, 8, 6, 4,
                                  3, 4, 6, 6, 6, 6, 4, 3,
                                  2, 3, 4, 4, 4, 4, 3, 2 },
       currentRow, currentColumn, moveNumber = 0, testRow, testColumn,
        minRow, minColumn, minAccess = 9, accessNumber,
        horizontal[ SIZE ] = { 2, 1, -1, -2, -2, -1, 1, 2 },
        vertical[ SIZE ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
   bool done, closedTour = false;

   srand( time( 0 ) );

   clearBoard( board );   // initialize array board
   currentRow = rand() % SIZE;
   currentColumn = rand() % SIZE;
   firstMoveRow = currentRow;      // store first moves row
   firstMoveCol = currentColumn;   // store first moves col

   board[ currentRow ][ currentColumn ] = ++moveNumber;
   done = false;

   while ( !done ) {
      accessNumber = minAccess;

      for ( int moveType = 0; moveType < SIZE; ++moveType ) {
         testRow = currentRow + vertical[ moveType ];
         testColumn = currentColumn + horizontal[ moveType ];

         if ( validMove( testRow, testColumn, board ) ) {
            if ( access[ testRow ][ testColumn ] < accessNumber ) {
               accessNumber = access[ testRow ][ testColumn ];
               minRow = testRow;
               minColumn = testColumn;
            }

            --access[ testRow ][ testColumn ];
         }
      }

      if ( accessNumber == minAccess )
         done = true;
      else {
         currentRow = minRow;
         currentColumn = minColumn;
         board[ currentRow ][ currentColumn ] = ++moveNumber;

         // check for closed tour
         if ( moveNumber == 64 )
            for ( int m = 0; m < SIZE; ++m ) {
               testRow = currentRow + vertical[ m ];
               testColumn = currentColumn + horizontal[ m ];

               if ( testRow == firstMoveRow && testColumn == firstMoveCol )
                  closedTour = true;
            }
      }
   }

   cout << "The tour ended with " << moveNumber << " moves.\n";

   if ( moveNumber == 64 && closedTour == true )
      cout << "This was a CLOSED tour!\n\n";
   else if ( moveNumber == 64 )
      cout << "This was a full tour!\n\n";
   else
      cout << "This was not a full tour.\n\n";

   cout << "The board for this test is:\n\n";
   printBoard( board );

   return 0;
}

void clearBoard( int workBoard[][ SIZE ] )
{
   for ( int row = 0; row < SIZE; ++row )
      for ( int col = 0; col < SIZE; ++col )
         workBoard[ row ][ col ] = 0;
}

void printBoard( const int workBoard[][ SIZE ] )
{
   cout << "   0  1  2  3  4  5  6  7\n";

   for ( int row = 0; row < SIZE; ++row ) {
      cout << row;

      for ( int col = 0; col < SIZE; ++col )
         cout << setw( 3 ) << workBoard[ row ][ col ];

      cout << '\n';
   }

   cout << endl;
}

bool validMove( int row, int column, const int workBoard[][ SIZE ] )
{
   // NOTE: This test stops as soon as it becomes false
   return ( row >= 0 && row < SIZE && column >= 0 && column < SIZE
            && workBoard[ row ][ column ] == 0 );
}
```

```
The tour ended with 64 moves.
This was a full tour!

The board for this test is:

   0  1  2  3  4  5  6  7
0 42 25 50  7 40 23 52  5
1 49  8 41 24 51  6 39 22
2 26 43 48 55 46 59  4 53
3  9 56 45 60 63 54 21 38
4 44 27 64 47 58 37 62  3
5 13 10 57 36 61 18 31 20
6 28 35 12 15 30 33  2 17
7 11 14 29 34  1 16 19 32
```

4.29 (*The Sieve of Eratosthenes*) A prime integer is any integer that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

a) Create an array with all elements initialized to 1 (true). Array elements with prime subscripts will remain 1. All other array elements will eventually be set to zero.
b) Starting with array subscript 2 (subscript 1 must be prime), every time an array element is found whose value is 1, loop through the remainder of the array and set to zero every element whose subscript is a multiple of the subscript for the element with value 1. For array subscript 2, all elements beyond 2 in the array that are multiples of 2 will be set to zero (subscripts 4, 6, 8, 10, etc.); for array subscript 3, all elements beyond 3 in the array that are multiples of 3 will be set to zero (subscripts 6, 9, 12, 15, etc.); and so on.

When this process is complete, the array elements that are still set to one indicate that the subscript is a prime number. These subscripts can then be printed. Write a program that uses an array of 1000 elements to determine and print the prime numbers between 1 and 999. Ignore element 0 of the array.

```
// Exercise 4.29 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

int main()
{
   const int SIZE = 1000;
   int array[ SIZE ], count = 0;

   for ( int k = 0; k < SIZE; ++k )
      array[ k ] = 1;

   for ( int i = 1; i < SIZE; ++i )
      if ( array[ i ] == 1 && i != 1 )
         for ( int j = i; j <= SIZE; ++j )
            if ( j % i == 0 && j != i )
               array[ j ] = 0;

   // range 2 - 197
   for ( int q = 2; q < SIZE; ++q )
      if ( array[ q ] == 1 ) {
         cout << setw( 3 ) << q << " is a prime number.\n";
         ++count;
      }

   cout << "A total of " << count << " prime numbers were found." << endl;

   return 0;
}
```

```
  3 is a prime number.
  5 is a prime number.
  7 is a prime number.
 11 is a prime number.
 ...
983 is a prime number.
991 is a prime number.
997 is a prime number.
A total of 168 prime numbers were found.
```

4.30 (*Bucket Sort*) A bucket sort begins with a single-subscripted array of positive integers to be sorted and a double-subscripted array of integers with rows subscripted from 0 to 9 and columns subscripted from 0 to *n* - 1, where *n* is the number of values in the array to be sorted. Each row of the double-subscripted array is referred to as a bucket. Write a function **`bucketSort`** that takes an integer array and the array size as arguments and performs as follows:

a) Place each value of the single-subscripted array into a row of the bucket array based on the value's ones digit. For example, 97 is placed in row 7, 3 is placed in row 3 and 100 is placed in row 0. This is called a "distribution pass."
b) Loop through the bucket array row by row, and copy the values back to the original array. This is called a "gathering pass." The new order of the preceding values in the single-subscripted array is 100, 3 and 97.
c) Repeat this process for each subsequent digit position (tens, hundreds, thousands, etc.).

On the second pass, 100 is placed in row 0, 3 is placed in row 0 (because 3 has no tens digit) and 97 is placed in row 9. After the gathering pass, the order of the values in the single-subscripted array is 100, 3 and 97. On the third pass, 100 is placed in row 1, 3 is placed in row zero and 97 is placed in row zero (after the 3). After the last gathering pass, the original array is now in sorted order.

Note that the double-subscripted array of buckets is 10 times the size of the integer array being sorted. This sorting technique provides better performance than a bubble sort, but requires much more memory. The bubble sort requires space for only one additional element of data. This is an example of the space–time trade-off: The bucket sort uses more memory than the bubble sort, but performs better. This version of the bucket sort requires copying all the data back to the original array on each pass. Another possibility is to create a second double-subscripted bucket array and repeatedly swap the data between the two bucket arrays.

```
// Exercise 4.30 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

// constant size must be defined as the array size for bucketSort to work
const int SIZE = 12;

void bucketSort( int [] );
void distributeElements( int [], int [][ SIZE ], int );
void collectElements( int [], int [][ SIZE ] );
int numberOfDigits( int [], int );
void zeroBucket( int [][ SIZE ] );

int main()
{
   int array[ SIZE ] = { 19, 13, 5, 27, 1, 26, 31, 16, 2, 9, 11, 21 };

   cout << "Array elements in original order:\n";

   for ( int i = 0; i < SIZE; ++i )
      cout << setw( 3 ) << array[ i ];

   cout << '\n';
   bucketSort( array );

   cout << "\nArray elements in sorted order:\n";

   for ( int j = 0; j < SIZE; ++j )
      cout << setw( 3 ) << array[ j ];

   cout << endl;

   return 0;
}

// Perform the bucket sort algorithm
void bucketSort( int a[] )
{
   int totalDigits, bucket[ 10 ][ SIZE ] = { 0 };

   totalDigits = numberOfDigits( a, SIZE );

   for ( int i = 1; i <= totalDigits; ++i ) {
      distributeElements( a, bucket, i );
      collectElements( a, bucket );

      if ( i != totalDigits )
         zeroBucket( bucket );  // set all bucket contents to zero
   }
}

// Determine the number of digits in the largest number
int numberOfDigits( int b[], int arraySize )
{
   int largest = b[ 0 ], digits = 0;

   for ( int i = 1; i < arraySize; ++i )
      if ( b[ i ] > largest )
         largest = b[ i ];

   while ( largest != 0 ) {
      ++digits;
      largest /= 10;
   }

   return digits;
}

// Distribute elements into buckets based on specified digit
void distributeElements( int a[], int buckets[][ SIZE ], int digit )
{
   int divisor = 10, bucketNumber, elementNumber;

   for ( int i = 1; i < digit; ++i )   // determine the divisor
      divisor *= 10;                 // used to get specific digit

   for ( int k = 0; k < SIZE; ++k ) {
      // bucketNumber example for hundreds digit:
      // (1234 % 1000 - 1234 % 100) / 100 --> 2
      bucketNumber = ( a[ k ] % divisor - a[ k ] %
                     ( divisor / 10 ) ) / ( divisor / 10 );

      // retrieve value in buckets[bucketNumber][0] to determine
      // which element of the row to store a[i] in.
      elementNumber = ++buckets[ bucketNumber ][ 0 ];
      buckets[ bucketNumber ][ elementNumber ] = a[ k ];
   }
}

// Return elements to original array
void collectElements( int a[], int buckets[][ SIZE ])
{
   int subscript = 0;

   for ( int i = 0; i < 10; ++i )
      for ( int j = 1; j <= buckets[ i ][ 0 ]; ++j )
         a[ subscript++ ] = buckets[ i ][ j ];
}

// Set all buckets to zero
void zeroBucket( int buckets[][ SIZE ] )
{
   for ( int i = 0; i < 10; ++i )
      for ( int j = 0; j < SIZE; ++j )
         buckets[ i ][ j ] = 0;
}
```

```
Array elements in original order:
 19 13  5 27  1 26 31 16  2  9 11 21

Array elements in sorted order:
  1  2  5  9 11 13 16 19 21 26 27 31
```

RECURSION EXERCISES

4.31 (*Selection Sort*) A selection sort searches an array looking for the smallest element in the array. Then, the smallest element is swapped with the first element of the array. The process is repeated for the subarray beginning with the second element of th e array. Each pass of the array results in one element being placed in its proper location. This sort performs comparably to the bubble sort—for an array of *n* elements, *n* - 1 passes must be made, and for each subarray, *n* - 1 comparisons must be made to find the smallest value. When the subarray being processed contains one element, the array is sorted. Write recursive function **selectionSort** to perform this algorithm.

```
// Exercise 4.31 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <cstdlib>
#include <ctime>

void selectionSort( int [], int );

int main()
{
   const int SIZE = 10, MAXRANGE = 1000;
   int sortThisArray[ SIZE ] = { 0 };

   srand( time( 0 ) );

   for ( int i = 0; i < SIZE; ++i )
      sortThisArray[ i ] = 1 + rand() % MAXRANGE;

   cout << "\nUnsorted array is:\n";
   for ( int j = 0; j < SIZE; ++j )
      cout << ' ' << sortThisArray[ j ] << ' ';

   selectionSort( sortThisArray, SIZE );

   cout << "\n\nSorted array is:\n";
   for ( int k = 0; k < SIZE; ++k )
      cout << ' ' << sortThisArray[ k ] << ' ';

   cout << '\n' << endl;

   return 0;
}

void selectionSort( int array[], int size )
{
   int temp;

   if ( size >= 1 ) {

      for ( int loop = 0; loop < size; ++loop )
         if ( array[ loop ] < array[ 0 ] ) {
            temp = array[ loop ];
            array[ loop ] = array[ 0 ];
            array[ 0 ] = temp;
         }

      selectionSort( &array[ 1 ], size - 1 );
   }
}
```

```
Unsorted array is:
 957  848  727  597  384  617  228  424  878  10

Sorted array is:
 10  228  384  424  597  617  727  848  878  957
```

4.32 (*Palindromes*) A palindrome is a string that is spelled the same way forwards and backwards. Some examples of palindromes are "radar," "able was i ere i saw elba" and (if blanks are ignored) "a man a plan a canal panama." Write a recursive function **testPalindrome** that returns **true** if the string stored in the array is a palindrome, and **false** otherwise. The function should ignore spaces and punctuation in the string.

```
// Exercise 4.32 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

bool testPalindrome( const char [], int, int );

int main()
{
   const int SIZE = 80;
   char c, string[ SIZE ], copy[ SIZE ];
   int count = 0, copyCount, i;

   cout << "Enter a sentence:\n";

   while ( ( c = cin.get() ) != '\n' && count < SIZE )
      string[ count++ ] = c;

   string[ count ] = '\0';    // terminate cstring

   // make a copy of cstring without spaces
   for ( copyCount = 0, i = 0; string[ i ] != '\0'; ++i )
      if ( string[ i ] != ' ' )
         copy[ copyCount++ ] = string[ i ];

   if ( testPalindrome( copy, 0, copyCount - 1 ) )
      cout << '\"' << string << "\" is a palindrome" << endl;
   else
      cout << '\"' << string << "\" is not a palindrome" << endl;

   return 0;
}

bool testPalindrome( const char array[], int left, int right )
{
   if ( left == right || left > right )
      return true;
   else if ( array[ left ] != array[ right ] )
      return false;
   else
      return testPalindrome( array, left + 1, right - 1 );
}
```

```
Enter a sentence:
alucard e dracula
"alucard e dracula" is a palindrome
```

4.33 (*Linear Search)* Modify the program in Fig. 4.19 to use recursive function **linearSearch** to perform a linear search of the array. The function should receive an integer array and the size of the array as arguments. If the search key is found, return the array subscript; otherwise, return –1.

```
// Exercise 4.33 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int linearSearch( const int [], int, int, int );

int main()
{
   const int SIZE = 100;
   int array[ SIZE ], searchKey, element;

   for ( int loop = 0; loop < SIZE; ++loop )
      array[ loop ] = 2 * loop;

   cout << "Enter the integer search key: ";
   cin >> searchKey;

   element = linearSearch( array, searchKey, 0, SIZE - 1 );

   if ( element != -1 )
      cout << "Found value in element " << element << endl;
   else
      cout << "Value not found" << endl;

   return 0;
}

int linearSearch( const int array[], int key, int low, int high )
{
   if ( array[low] == key )
      return low;
   else if ( low == high )
      return -1;
   else
      return linearSearch( array, key, low + 1, high );
}
```

```
Enter the integer search key: 22
Found value in element 11
```

4.34 (*Binary Search*) Modify the program of Fig. 4.20 to use a recursive function **binarySearch** to perform the binary search of the array. The function should receive an integer array and the starting subscript and ending subscript as arguments. If the search key is found, return the array subscript; otherwise, return –1.

```
// Exercise 4.34 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;

const int SIZE = 15;

int binarySearch( const int [], int, int, int );
void printRow( const int [], int, int, int );
void printHeader( void );

int main()
{
   int a[ SIZE ], key, result;

   for ( int i = 0; i < SIZE; ++i )
      a[ i ] = 2 * i;

   cout << "Enter a number between 0 and 28: ";
   cin >> key;

   printHeader();
   result = binarySearch( a, key, 0, SIZE - 1 );

   if ( result != -1 )
      cout << '\n' << key << " found in array element " << result << endl;
   else
      cout << '\n' << key << " not found" << endl;

   return 0;
}

int binarySearch( const int b[], int searchKey, int low, int high )
{
   int middle;

   if ( low <= high ) {
      middle = ( low + high ) / 2;
      printRow( b, low, middle, high );

      if ( searchKey == b[ middle ] )
         return middle;
      else if ( searchKey < b[ middle ] )
         return binarySearch( b, searchKey, low, middle - 1 );
      else
         return binarySearch( b, searchKey, middle + 1, high );
   }

   return -1;   // searchKey not found
}

// Print a header for the output
```

```
void printHeader( void )
{
   cout << "Subscripts:\n";

   for ( int i = 0; i < SIZE; ++i )
      cout << setw( 3 ) << i << ' ';

   cout << '\n';

   for ( int k = 1; k <= 4 * SIZE; ++k )
      cout << '-';

   cout << '\n';
}

// print one row of output showing the current
// part of the array being processed.
void printRow( const int b[], int low, int mid, int high )
{
   for ( int i = 0; i < SIZE; ++i )
      if ( i < low || i > high )
         cout << "    ";
      else if ( i == mid )
         cout << setw( 3 ) << b[ i ] << '*';    // mark middle value
      else
         cout << setw( 3 ) << b[ i ] << ' ';

   cout << '\n';
}
```

```
Enter a number between 0 and 28: 17
Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                 16  18* 20
                                 16*

17 not found
```

4.35 (*Eight Queens)* Modify the Eight Queens program you created in Exercise 4.26 to solve the problem recursively.

4.36 (*Print an array*) Write a recursive function **printArray** that takes an array and the size of the array as arguments and returns nothing. The function should stop processing and return when it receives an array of size zero.

```
// Exercise 4.36 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

void printArray( const int [], int, int );
```

```

int main()
{
   const int SIZE = 10, MAXNUMBER = 500;
   int array[ SIZE ];

   srand( time( 0 ) );

   for ( int loop = 0; loop < SIZE; ++loop )
      array[ loop ] = 1 + rand() % MAXNUMBER;

   cout << "Array values printed in main:\n";

   for ( int j = 0; j < SIZE; ++j )
      cout << setw( 5 ) << array[ j ];

   cout << "\n\nArray values printed in printArray:\n";
   printArray( array, 0, SIZE - 1 );
   cout << endl;

   return 0;
}

void printArray( const int array[], int low, int high )
{
   cout << setw( 5 ) << array[ low ];

   if ( low == high )
      return;
   else
      printArray( array, low + 1, high );
}
```

```
Array values printed in main:
  432   14   59  484   45  388  355  384  425  448

Array values printed in printArray:
  432   14   59  484   45  388  355  384  425  448
```

4.37 (*Print a string backwards*) Write a recursive function **stringReverse** that takes a character array containing a string as an argument, prints the string backwards and returns nothing. The function should stop processing and return when the terminating null character is encountered.

```
// Exercise 4.37 Solution
#include <iostream>

using std::cout;
using std::endl;

void stringReverse( const char [] );

int main()
{
   const int SIZE = 30;
   char strArray[ SIZE ] = "Print this string backwards.";

   for ( int loop = 0; loop < SIZE; ++loop )
      cout << strArray[ loop ];

   cout << '\n';
   stringReverse( strArray );
   cout << endl;

   return 0;
}

void stringReverse( const char strArray[] )
{
   if ( strArray[ 0 ] == '\0' )
      return;

   stringReverse( &strArray[ 1 ] );
   cout << strArray[ 0 ];
}
```

```
Print this string backwards.
.sdrawkcab gnirts siht tnirP
```

4.38 (*Find the minimum value in an array*) Write a recursive function **recursiveMinimum** that takes an integer array and the array size as arguments and returns the smallest element of the array. The function should stop processing and return when i t receives an array of 1 element.

```
// Exercise 4.38 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

const int MAXRANGE = 1000;
int recursiveMinimum( const int [], int, int );

int main()
{
   const int SIZE = 10;
   int array[ SIZE ], smallest;

   srand( time( 0 ) );

   for ( int loop = 0; loop < SIZE; ++loop )
      array[ loop ] = 1 + rand() % MAXRANGE;

   cout << "Array members are:\n";
   for ( int k = 0; k < SIZE; ++k )
      cout << setw( 5 ) << array[ k ];

   cout << '\n';
   smallest = recursiveMinimum( array, 0, SIZE - 1 );
   cout << "\nSmallest element is: " << smallest << endl;

   return 0;
}
```

```

int recursiveMinimum( const int array[], int low, int high )
{
   static int smallest = MAXRANGE;

   if ( array[ low ] < smallest )
      smallest = array[ low ];

   return low == high ? smallest : recursiveMinimum( array, low + 1, high );
}
```

```
Array members are:
    7   84  951  884  404  167  905   93  744  115
Smallest element is: 7
```

5 Pointers and Strings Solutions

Solutions

5.8 State whether the following are *true* or *false*. If *false*, explain why.

a) Two pointers that point to different arrays cannot be compared meaningfully.

ANS: True.

b) Because the name of an array is a pointer to the first element of the array, array names may be manipulated in precisely the same manner as pointers.

ANS: False. An array name cannot be used to refer to another location in memory.

5.9 Answer each of the following. Assume that unsigned integers are stored in 2 bytes and that the starting address of the array is at location 1002500 in memory.

a) Declare an array of type **`unsigned int`** called **`values`** with 5 elements, and initialize the elements to the even integers from 2 to 10. Assume that the symbolic constant **`SIZE`** has been defined as **`5`**.

ANS: **`unsigned values[ SIZE ] = { 2, 4, 6, 8, 10 };`**

b) Declare a pointer **`vPtr`** that points to an object of type **`unsigned int`**.

ANS: **`unsigned *vPtr;`**

c) Print the elements of array **`values`** using array subscript notation. Use a **`for`** structure and assume integer control variable **`i`** has been declared.

ANS:

```
for ( i = 0; i < SIZE; ++i )
   cout << setw( 4 ) << values[ i ];
```

d) Give two separate statements that assign the starting address of array **`values`** to pointer variable **`vPtr`**.

ANS: **`vPtr = values;`** and **`vPtr = &values[ 0 ];`**

e) Print the elements of array **`values`** using pointer/offset notation.

ANS:

```
for ( i = 0; i < SIZE; ++i )
   cout << setw( 4 ) << *( vPtr + i );
```

f) Print the elements of array **`values`** using pointer/offset notation with the array name as the pointer.

ANS:

```
for ( i = 0; i < SIZE; ++i )
   cout << setw( 4 ) << *( values + i );
```

g) Print the elements of array **`values`** by subscripting the pointer to the array.

ANS:

```
for ( i = 0; i < SIZE; ++i )
   cout << setw( 4 ) << vPtr[ i ];
```

h) Refer to the fifth element of `values` using array subscript notation pointer/offset notation with the array name as the pointer, pointer subscript notation, and pointer/offset notation.

ANS: `values[ 4 ]`, `*( values + 4 )`, `vPtr[ 4 ]`, `*( vPtr + 4 )`

i) What address is referenced by `vPtr + 3`? What value is stored at that location?

ANS: The address of the location pertaining to `values[ 3 ]` (i.e., 1002506). 8.

j) Assuming `vPtr` points to `values[ 4 ]`, what address is referenced by `vPtr -= 4`? What value is stored at that location?

ANS: The address of where `values` begins in memory (i.e., 1002500). 2.

5.10 For each of the following, write a single statement that performs the indicated task. Assume that long integer variables `value1` and `value2` have been declared and that `value1` has been initialized to `200000`.

a) Declare the variable `lPtr` to be a pointer to an object of type `long`.

ANS: `long *lPtr;`

b) Assign the address of variable `value1` to pointer variable `lPtr`.

ANS: `lPtr = &value1;`

c) Print the value of the object pointed to by `lPtr`.

ANS: `cout << *lPtr << '\n';`

d) Assign the value of the object pointed to by `lPtr` to variable `value2`.

ANS: `value2 = *lPtr;`

e) Print the value of `value2`.

ANS: `cout << value2 << '\n';`

f) Print the address of `value1`.

ANS: `cout << &value1 << '\n';`

g) Print the address stored in `lPtr`. Is the value printed the same as the address of `value1`?

ANS: `cout << lPtr << '\n';` yes.

5.11 Do each of the following.

a) Write the function header for function `zero` that takes a long integer array parameter `bigIntegers` and does not return a value.

ANS: `void zero( long bigIntegers[] )` or
`void zero( long *bigIntegers )`

b) Write the function prototype for the function in part **(a)**.

ANS: `void zero( long [] );` or `void zero( long * );`

c) Write the function header for function `add1AndSum` that takes an integer array parameter `oneTooSmall` and returns an integer.

ANS: `int add1Andsum( int oneTooSmall[] )` or
`int add1Andsum( int *oneTooSmall )`

d) Write the function prototype for the function described in part **(c)**.

ANS: `int add1Andsum( int [] );` or `int add1Andsum( int * );`

Note: Exercises 5.12 through 5.15 are reasonably challenging. Once you have done these problems, you ought to be able to implement most popular card games easily.

5.12 Modify the program in Fig. 5.24 so that the card dealing function deals a five-card poker hand. Then write functions to accomplish each of the following:

a) Determine if the hand contains a pair.
b) Determine if the hand contains two pairs.
c) Determine if the hand contains three of a kind (e.g., three jacks).
d) Determine if the hand contains four of a kind (e.g., four aces).
e) Determine if the hand contains a flush (i.e., all five cards of the same suit).
f) Determine if the hand contains a straight (i.e., five cards of consecutive face values).

```
// Exercise 5.12 Solution
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

```

```
using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include <cstdlib>
#include <ctime>

void shuffle( int [][ 13 ] );
void deal( const int [][ 13 ], const char *[], const char *[], int [][ 2 ] );
void pair( const int [][ 13 ], const int [][ 2 ], const char *[] );
void threeOfKind( const int [][ 13 ], const int [][ 2 ], const char *[] );
void fourOfKind( const int [][ 13 ], const int [][ 2 ], const char *[] );
void flushHand( const int [][ 13 ], const int [][ 2 ], const char *[] );
void straightHand( const int [][ 13 ], const int [][ 2 ], const char *[],
                   const char *[] );

int main()
{
   const char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades" },
              *face[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
                          "Seven", "Eight", "Nine", "Ten", "Jack", "Queen",
                          "King" };
   int deck[ 4 ][ 13 ] = { 0 }, hand[ 5 ][ 2 ] = { 0 };

   srand( time( 0 ) );

   shuffle( deck );
   deal( deck, face, suit, hand );
   pair( deck, hand, face );
   threeOfKind( deck, hand, face );
   fourOfKind( deck, hand, face );
   flushHand( deck, hand, suit );
   straightHand( deck, hand, suit, face );

   return 0;
}

void shuffle( int wDeck[][ 13 ] )
{
   int row, column;

   for ( int card = 1; card <= 52; ++card ) {
      do {
        row = rand() % 4;
        column = rand() % 13;
      } while ( wDeck[ row ][ column ] != 0 );

      wDeck[ row ][ column ] = card;
   }
}

// deal a five card poker hand
void deal( const int wDeck[][ 13 ], const char *wFace[],
           const char *wSuit[], int wHand[][ 2 ] )
{
   int r = 0;

   cout << "The hand is:\n";

   for ( int card = 1; card < 6; ++card )
      for ( int row = 0; row <= 3; ++row )
         for ( int column = 0; column <= 12; ++column )
```

```
            if ( wDeck[ row ][ column ] == card ) {
               wHand[ r ][ 0 ] = row;
               wHand[ r ][ 1 ] = column;
               cout << setw( 5 ) << wFace[ column ]
                    << " of " << setw( 8 ) << setiosflags( ios::left )
                    << wSuit[ row ] << ( card % 2 == 0 ? '\n' : '\t' )
                    << resetiosflags( ios::left );
               ++r;
            }

   cout << '\n';
}

// pair determines if the hand contains one or two pair
void pair( const int wDeck[][ 13 ], const int wHand[][ 2 ],
           const char *wFace[] )
{
   int counter[ 13 ] = { 0 };

   for ( int r = 0; r < 5; ++r )
      ++counter[ wHand[ r ][ 1 ] ];

   cout << '\n';

   for ( int p = 0; p < 13; ++p )
      if ( counter[ p ] == 2 )
        cout << "The hand contains a pair of " << wFace[ p ] << "'s.\n";
}

void threeOfKind( const int wDeck[][ 13 ], const int wHand[][ 2 ],
                  const char *wFace[] )
{
   int counter[ 13 ] = { 0 };

   for ( int r = 0; r < 5; ++r )
      ++counter[ wHand[ r ][ 1 ] ];

   for ( int t = 0; t < 13; t++ )
      if ( counter[ t ] == 3 )
         cout << "The hand contains three " << wFace[ t ] << "'s.\n";
}

void fourOfKind( const int wDeck[][ 13 ], const int wHand[][ 2 ],
                 const char *wFace[] )
{
   int counter[ 13 ] = { 0 };

   for ( int r = 0; r < 5; ++r )
      ++counter[ wHand[ r ][ 1 ] ];

   for ( int k = 0; k < 13; ++k )
      if ( counter[ k ] == 4 )
         cout << "The hand contains four " << wFace[ k ] << "'s.\n";
}

void flushHand( const int wDeck[][ 13 ], const int wHand[][ 2 ],
                const char *wSuit[] )
{
   int count[ 4 ] = { 0 };

   for ( int r = 0; r < 5; ++r )
      ++count[ wHand[ r ][ 0 ] ];

   for ( int f = 0; f < 4; ++f )
      if ( count[ f ] == 5 )
         cout << "The hand contains a flush of " << wSuit[ f ] << "'s.\n";
}

void straightHand( const int wDeck[][ 13 ], const int wHand[][ 2 ],
                   const char *wSuit[], const char *wFace[] )
{
   int s[ 5 ] = { 0 }, temp;

   // copy column locations to sort
   for ( int r = 0; r < 5; ++r )
      s[ r ] = wHand[ r ][ 1 ];

   // bubble sort column locations
   for ( int pass = 1; pass < 5; ++pass )
      for ( int comp = 0; comp < 4; ++comp )
         if ( s[ comp ] > s[ comp + 1 ] ) {
            temp = s[ comp ];
            s[ comp ] = s[ comp + 1 ];
            s[ comp + 1 ] = temp;
         }

   // check if sorted columns are a straight
   if ( s[ 4 ] - 1 == s[ 3 ] && s[ 3 ] - 1 == s[ 2 ]
        && s[ 2 ] - 1 == s[ 1 ] && s[ 1 ] - 1 == s[ 0 ] ) {
      cout << "The hand contains a straight consisting of\n";

      for ( int j = 0; j < 5; ++j )
         cout << wFace[ wHand[ j ][ 1 ] ] << " of " << wSuit[ wHand[ j ][ 0 ] ]
              << '\n';
   }
}
```

```
The hand is:
  Ace of Diamonds        Jack of Clubs
  Ten of Clubs          Queen of Clubs
Deuce of Hearts
```

5.13 Use the functions developed in Exercise 5.12 to write a program that deals two five-card poker hands, evaluates each hand, and determines which is the better hand.

5.14 Modify the program developed in Exercise 5.13 so that it can simulate the dealer. The dealer's five-card hand is dealt "face down" so the player cannot see it. The program should then evaluate the dealer's hand, and, based on the quality of the hand, the dealer should draw one, two or three more cards to replace the corresponding number of unneeded cards in the original hand. The program should then reevaluate the dealer's hand. (*Caution:* This is a difficult problem!)

5.15 Modify the program developed in Exercise 5.14 so that it can handle the dealer's hand automatically, but the player is allowed to decide which cards of the player's hand to replace. The program should then evaluate both hands and determine who wins. Now use this new program to play 20 games against the computer. Who wins more games: you or the computer? Have one of your friends play 20 games against the computer. Who wins more games? Based on the results of these games, make appropriate modifications to refine your poker-playing program. (This, too, is a difficult problem.) Play 20 more games. Does your modified program play a better game?

5.16 In the card-shuffling and dealing program of Fig. 5.24, we intentionally used an inefficient shuffling algorithm that introduced the possibility of indefinite postponement. In this problem, you will create a high-performance shuffling algorithm that avoids indefinite postponement.

Modify Fig. 5.24 as follows. Initialize the `deck` array as shown in Fig. 5.35. Modify the `shuffle` function to loop row by row and column by column through the array touching every element once. Each element should be swapped with a randomly selected element of the array. Print the resulting array to determine if the deck is satisfactorily shuffled (as in Fig. 5.36, for example). You may want your program to call the **`shuffle`** function several times to ensure a satisfactory shuffle .

Note that although the approach in this problem improves the shuffling algorithm, the dealing algorithm still requires searching the **`deck`** array for card 1, then card 2, then card 3, and so on. Worse yet, even after the dealing algorithm locates and deals the card, the algorithm continues searching through the remainder of the deck. Modify the program of Fig. 5.24 so that once a card i s dealt, no further attempts are made to match that card number, and the program immediately proceeds with dealing the next card.

Unshuffled `deck` array

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	2	3	4	5	6	7	8	9	10	11	12	13
1	14	15	16	17	18	19	20	21	22	23	24	25	26
2	27	28	29	30	31	32	33	34	35	36	37	38	39
3	40	41	42	43	44	45	46	47	48	49	50	51	52

Fig. 5.35 Unshuffled **`deck`** array.

Sample shuffled `deck` array

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	19	40	27	25	36	46	10	34	35	41	18	2	44
1	13	28	14	16	21	30	8	11	31	17	24	7	1
2	12	33	15	42	43	23	45	3	29	32	4	47	26
3	50	38	52	39	48	51	9	5	37	49	22	6	20

Fig. 5.36 Sample shuffled **`deck`** array.

```
// Exercise 5.16 Solution
#include <iostream>

using std::cout;
using std::ios;


#include <iomanip>
using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include <cstdlib>
#include <ctime>

void shuffle( int [][ 13 ] );
void deal( const int [][ 13 ], const char *[], const char *[] );

int main()
{
   int card = 1, deck[ 4 ][ 13 ] = { 0 };
   const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
   const char *face[ 13 ] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
                                 "Seven", "Eight", "Nine", "Ten", "Jack", "Queen",
                                 "King" };

   srand( time( 0 ) );

   // initialize deck
   for ( int row = 0; row <= 3; ++row )
      for ( int column = 0; column <= 12; ++column )
         deck[ row ][ column ] = card++;

   shuffle( deck );
   deal( deck, face, suit );

   return 0;
}

void shuffle( int workDeck[][ 13 ] )
{
   int temp, randRow, randColumn;

   for ( int row = 0; row <= 3; ++row )
      for ( int column = 0; column <= 12; ++column ) {
         randRow = rand() % 4;
         randColumn = rand() % 13;
         temp = workDeck[ row ][ column ];
         workDeck[ row][ column ] = workDeck[ randRow ][ randColumn ];
         workDeck[ randRow ][ randColumn ] = temp;
      }
}

void deal( const int workDeck2[][ 13 ], const char *workFace[],
           const char *workSuit[] )
{
   for ( int card = 1; card <= 52; ++card )
      for ( int row = 0; row <= 3; ++row )
         for ( int column = 0; column <= 12; ++column )
            if ( workDeck2[ row ][ column ] == card ) {
               cout << setw( 8 ) << workFace[ column ] << " of "
                    << setiosflags( ios::left ) << setw( 8 )
                    << workSuit[ row ]
                    << ( card % 2 == 0 ? '\n' : '\t' )
                    << resetiosflags( ios::left );
               break;
            }
}
```

```
    King of Hearts          King of Diamonds
   Eight of Spades          Four of Clubs
    Five of Clubs          Queen of Hearts
   Eight of Hearts          Nine of Diamonds
     Ace of Clubs          Deuce of Clubs
     Six of Diamonds        Five of Hearts
   Seven of Hearts          Nine of Hearts
     Six of Spades          Jack of Diamonds
    Five of Spades         Queen of Diamonds
    King of Clubs           King of Spades
    Five of Diamonds       Seven of Diamonds
   Eight of Diamonds        Four of Hearts
    Four of Spades           Ten of Hearts
   Three of Spades         Three of Diamonds
     Ten of Spades          Four of Diamonds
     Ten of Diamonds       Deuce of Hearts
     Ace of Diamonds         Six of Hearts
    Nine of Clubs           Nine of Spades
   Deuce of Spades         Eight of Clubs
   Queen of Spades           Six of Clubs
   Seven of Clubs           Jack of Clubs
     Ace of Hearts         Three of Clubs
   Deuce of Diamonds       Queen of Clubs
   Seven of Spades           Ace of Spades
    Jack of Hearts          Jack of Spades
     Ten of Clubs          Three of Hearts
```

5.17 (*Simulation: The Tortoise and the Hare*) In this problem you will re-create the classic race of the tortoise and the hare. You will use random-number generation to develop a simulation of this memorable event.

Our contenders begin the race at "square 1" of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and let tuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground.

There is a clock that ticks once per second. With each tick of the clock, your program should adjust the position of the animals according to the following rules:

Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the right
	Small slip	20%	2 squares to the left

Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (i.e., the "starting gate"). If an animal slips left before square 1, move the animal back to square 1.

Generate the percentages in the preceding table by producing a random integer i in the range $1 \le i \le 10$. For the tortoise, perform a "fast plod" when $1 \le i \le 5$, a "slip" when $6 \le i \le 7$ or a "slow plod" when $8 \le i \le 10$. Use a similar technique to move the hare.

Begin the race by printing

```
BANG !!!!!
AND THEY'RE OFF !!!!!
```

For each tick of the clock (i.e., each repetition of a loop), print a 70-position line showing the letter `T` in the tortoise’s position and the letter `H` in the hare’s position. Occasionally, the contenders land on the same square. In this case, the tortoise bites the hare, and your program should print `OUCH!!!` beginning at that position. All print positions other than the `T`, the `H` or the `OUCH!!!` (in case of a tie) should be blank.

After printing each line, test if either animal has reached or passed square 70. If so, print the winner and terminate the simulation. If the tortoise wins, print `TORTOISE WINS!!! YAY!!!` If the hare wins, print `Hare wins. Yuch.` If both animals win on the same clock tick, you may want to favor the turtle (the “underdog”), or you may want to print `It's a tie`. If neither animal wins, perform the loop again to simulate the next tick of the clock. When you are ready to run your program, assemble a group of fans to watch the race. You’ll be amazed how involved the audience gets!

```
// Exercise 5.17 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <cstdlib>
#include <ctime>

#include <iomanip>

using std::setw;

const int RACE_END = 70;

void moveTortoise( int * const );
void moveHare( int * const );
void printCurrentPositions( const int * const, const int * const );

int main()
{
   int tortoise = 1, hare = 1, timer = 0;

   srand( time( 0 ) );

   cout << "ON YOUR MARK, GET SET\nBANG               !!!!"
        << "\nAND THEY'RE OFF    !!!!\n";

   while ( tortoise != RACE_END && hare != RACE_END ) {
      moveTortoise( &tortoise );
      moveHare( &hare );
      printCurrentPositions( &tortoise, &hare );
      ++timer;
   }

   if ( tortoise >= hare )
      cout << "\nTORTOISE WINS!!! YAY!!!\n";
   else
      cout << "Hare wins. Yuch.\n";

   cout << "TIME ELAPSED = " << timer << " seconds" << endl;

   return 0;
}

void moveTortoise( int * const turtlePtr )
```

```
{
   int x = 1 + rand() % 10;

   if ( x >= 1 && x <= 5 )        // fast plod
      *turtlePtr += 3;
   else if ( x == 6 || x == 7 )   // slip
      *turtlePtr -= 6;
   else                           // slow plod
      ++( *turtlePtr );

   if ( *turtlePtr < 1 )
      *turtlePtr = 1;
   else if ( *turtlePtr > RACE_END )
      *turtlePtr = RACE_END;
}

void moveHare( int * const rabbitPtr )
{
   int y = 1 + rand() % 10;

   if ( y == 3 || y == 4 )        // big hop
      *rabbitPtr += 9;
   else if ( y == 5 )             // big slip
      *rabbitPtr -= 12;
   else if ( y >= 6 && y <= 8 )   // small hop
      ++( *rabbitPtr );
   else if ( y > 8 )            // small slip
      *rabbitPtr -= 2;

   if ( *rabbitPtr < 1 )
      *rabbitPtr = 1;
   else if ( *rabbitPtr > RACE_END )
      *rabbitPtr = RACE_END;
}

void printCurrentPositions( const int * const snapperPtr,
                            const int * const bunnyPtr )
{
   if ( *bunnyPtr == *snapperPtr )
      cout << setw( *bunnyPtr ) << "OUCH!!!";
   else if ( *bunnyPtr < *snapperPtr )
      cout << setw( *bunnyPtr ) << 'H'
           << setw( *snapperPtr - *bunnyPtr ) << 'T';
   else
      cout << setw( *snapperPtr ) << 'T'
           << setw( *bunnyPtr - *snapperPtr ) << 'H';

   cout << '\n';
}
```

```
ON YOUR MARK, GET SET
BANG               !!!!
AND THEY'RE OFF    !!!!
H  T
    T    H
T                 H
 T                 H
T                   H
 T                H
T     H
T   H
T   H
T   H
   T         H
      T       H
         T             H
            T           H
               T         H
                  T               H
            T         H
               T    H
                T    H
                   T H
         H          T
       H               T
H                       T
H                        T
H                           T
H                              T
H                               T
H                                  T
 H                                    T
  H                             T
  H                                T
           H                          T
            H                            T
             H                              T
              H                       T
              H                 T
  H                                T
  H                                 T
H                             T
 H                               T
  H                                 T
H                                    T
H                                        T
H                                         T
H                                            T
H                                               T
H                                                T
         H                                        T
                  H                                  T
                           H                            T
                                    H                    T
                        H                                 T
                      H                             T
                       H                               T
                     H                                    T
                   H                                         T
                 H                                              T
               H                                                   T
               H                                                     T

TORTOISE WINS!!! YAY!!!
TIME ELAPSED = 59 seconds
```

SPECIAL SECTION: BUILDING YOUR OWN COMPUTER

In the next several problems, we take a temporary diversion away from the world of high-level-language programming. We "peel open" a computer and look at its internal structure. We introduce machine-language programming and write several machine-language programs. To make this an especially valuable experience, we then build a computer (through the technique of software-based *simulation)* on which you can execute your machine-language programs!

5.18 (*Machine-Language Programming*) Let us create a computer we will call the Simpletron. As its name implies, it is a simple machine, but, as we will soon see, a powerful one as well. The Simpletron runs programs written in the only language it directly understands; that is, Simpletron Machine Language, or SML for short.

The Simpletron contains an *accumulator*—a "special register" in which information is put before the Simpletron uses that information in calculations or examines it in various ways. All information in the Simpletron is handled in terms of*words.* A word is a signed four-digit decimal number, such as `+3364`, `-1293`, `+0007`, `-0001`, etc. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers `00`, `01`, ..., `99`.

Before running an SML program, we must *load,* or place, the program into memory. The first instruction (or statement) of every SML program is always placed in location `00`. The simulator will start executing at this location.

Each instruction written in SML occupies one word of the Simpletron's memory. (Thus, instructions are signed four-digit decimal numbers.) We shall assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron's memory may contain an instruction, a data value used by a program or an unused (and hence undefined) area of memory. The first two digits of each SML instruction are the *operation code* that specifies the operation to be performed. SML operation codes are shown in Fig. 5.37.

Operation code	Meaning
Input/output operations:	
`const int READ = 10`	Read a word from the keyboard into a specific location in memory.
`const int WRITE = 11;`	Write a word from a specific location in memory to the screen.
Load and store operations:	
`const int LOAD = 20;`	Load a word from a specific location in memory into the accumulator.
`const int STORE = 21;`	Store a word from the accumulator into a specific location in memory.
Arithmetic operations:	
`const int ADD = 30;`	Add a word from a specific location in memory to the word in the accumulator (leave result in accumulator).
`const int SUBTRACT = 31;`	Subtract a word from a specific location in memory from the word in the accumulator (leave result in accumulator).
`const int DIVIDE = 32;`	Divide a word from a specific location in memory into the word in the accumulator (leave result in accumulator).
`const int MULTIPLY = 33;`	Multiply a word from a specific location in memory by the word in the accumulator (leave result in accumulator).
Transfer of control operations:	
`const int BRANCH = 40;`	Branch to a specific location in memory.

Fig. 5.37 Simpletron Machine Language (SML) operation codes (part 1 of 2).

Operation code	Meaning
`const int BRANCHNEG = 41;`	Branch to a specific location in memory if the accumulator is negative.
`const int BRANCHZERO = 42;`	Branch to a specific location in memory if the accumulator is zero.
`const int HALT = 43;`	Halt—the program has completed its task.

Fig. 5.37 Simpletron Machine Language (SML) operation codes (part 2 of 2).

The last two digits of an SML instruction are the *operand*—the address of the memory location containing the word to which the operation applies.

Now let us consider several simple SML programs. The first SML program (Example 1) reads two numbers from the keyboard and computes and prints their sum. The instruction **`+1007`** reads the first number from the keyboard and places it into location **`07`** (which has been initialized to zero). Then instruction **`+1008`** reads the next number into location **`08`**. The *load* instruction, **`+2007`**, puts (copies) the first number into the accumulator, and the *add* instruction, **`+3008`**, adds the second number to the number in the accumulator. *All SML arithmetic instructions leave their results in the accumulator.* The *store* instruction, **`+2109`**, places (copies) the result back into memory location **`09`** from which the *write* instruction, **`+1109`**, takes the number and prints it (as a signed four-digit decimal number). The *halt* instruction, **`+4300`**, terminates execution.

Example 1 Location	Number	Instruction
`00`	`+1007`	`(Read A)`
`01`	`+1008`	`(Read B)`
`02`	`+2007`	`(Load A)`
`03`	`+3008`	`(Add B)`
`04`	`+2109`	`(Store C)`
`05`	`+1109`	`(Write C)`
`06`	`+4300`	`(Halt)`
`07`	`+0000`	`(Variable A)`
`08`	`+0000`	`(Variable B)`
`09`	`+0000`	`(Result C)`

The SML program in Example 2 reads two numbers from the keyboard and determines and prints the larger value. Note the use of the instruction **`+4107`** as a conditional transfer of control, much the same as C++'s **`if`** statement.

Example 2 Location	Number	Instruction
`00`	`+1009`	`(Read A)`
`01`	`+1010`	`(Read B)`
`02`	`+2009`	`(Load A)`
`03`	`+3110`	`(Subtract B)`
`04`	`+4107`	`(Branch negative to 07)`
`05`	`+1109`	`(Write A)`
`06`	`+4300`	`(Halt)`
`07`	`+1110`	`(Write B)`

Example 2 Location	Number	Instruction
`08`	`+4300`	`(Halt)`
`09`	`+0000`	`(Variable A)`
`10`	`+0000`	`(Variable B)`

Now write SML programs to accomplish each of the following tasks.

a) Use a sentinel-controlled loop to read positive numbers and compute and print their sum. Terminate input when a negative number is entered.

b) Use a counter-controlled loop to read seven numbers, some positive and some negative, and compute and print their average.

c) Read a series of numbers and determine and print the largest number. The first number read indicates how many numbers should be processed.

5.19 (*A Computer Simulator*) It may at first seem outrageous, but in this problem, you are going to build your own computer. No, you will not be soldering components together. Rather, you will use the powerful technique of *software-based simulation* to create a *software model* of the Simpletron. You will not be disappointed. Your Simpletron simulator will turn the computer you are using into a Simpletron, and you will actually be able to run, test and debug the SML programs you wrote in Exercise 5.18.

When you run your Simpletron simulator, it should begin by printing

```
*** Welcome to Simpletron! ***

*** Please enter your program one instruction ***
*** (or data word) at a time. I will type the ***
*** location number and a question mark (?).  ***
*** You then type the word for that location. ***
*** Type the sentinel -99999 to stop entering ***
*** your program. ***
```

Simulate the memory of the Simpletron with a single-subscripted array **`memory`** that has 100 elements. Now assume that the simulator is running, and let us examine the dialog as we enter the program of Example 2 of Exercise 5.18:

```
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
11 ? -99999

*** Program loading completed ***
*** Program execution begins  ***
```

The SML program has now been placed (or loaded) in array **`memory`**. Now the Simpletron executes your SML program. Execution begins with the instruction in location **`00`** and, like C++, continues sequentially, unless directed to some other part of the program by a transfer of control.

Use the variable **`accumulator`** to represent the accumulator register. Use the variable **`counter`** to keep track of the location in memory that contains the instruction being performed. Use variable **`operationCode`** to indicate the operation currently being performed (i.e., the left two digits of the instruction word). Use variable **`operand`** to indicate the memory location on which the current instruction operates. Thus, **`operand`** is the rightmost two digits of the instruction currently being performed. Do not execute instructions directly from memory. Rather, transfer the next instruction to be performed from memory to a variable called **`instructionRegister`**. Then "pick off" the left two digits and place them in **`operationCode`**, and "pick off" the right two digits and place them in **`operand`**. When Simpletron begins execution, the special registers are all initialized to zero.

Now let us "walk through" the execution of the first SML instruction, **`+1009`** in memory location **`00`**. This is called an *instruction execution cycle.*

The **`counter`** tells us the location of the next instruction to be performed. We *fetch* the contents of that location from **`memory`** by using the C++ statement

```
instructionRegister = memory[ counter ];
```

The operation code and operand are extracted from the instruction register by the statements

```
operationCode = instructionRegister / 100;
operand = instructionRegister % 100;
```

Now the Simpletron must determine that the operation code is actually a *read* (versus a *write*, a *load*, etc.). A **`switch`** differentiates among the twelve operations of SML.

In the **`switch`** structure, the behavior of various SML instructions is simulated as follows (we leave the others to the reader):

read:	`cin >> memory[ operand ];`
load:	`accumulator = memory[ operand ];`
add:	`accumulator += memory[ operand ];`
branch:	We will discuss the branch instructions shortly.
halt:	This instruction prints the message `*** Simpletron execution terminated ***`

It then prints the name and contents of each register, as well as the complete contents of memory. Such a printout is often called a *computer dump* (and, no, a computer dump is not a place where old computers go). To help you program your dump function, a sample dump format is shown in Fig. 5.38. Note that a dump after executing a Simpletron program would show the actual values of instructions and data values at the moment execution terminated.

Let us proceed with the execution of our program's first instruction—**`+1009`** in location **`00`**. As we have indicated, the **`switch`** statement simulates this by performing the C++ statement

```
cin >> memory[ operand ];
```

A question mark (**`?`**) should be displayed on the screen before the **`cin`** is executed to prompt the user for input. The Simpletron waits for the user to type a value and then press the *Return key.* The value is then read into location **`09`**.

At this point, simulation of the first instruction is completed. All that remains is to prepare the Simpletron to execute the next instruction. Since the instruction just performed was not a transfer of control, we need merely increment the instruction counter register as follows:

```
++counter;
```

This completes the simulated execution of the first instruction. The entire process (i.e., the instruction execution cycle) begins anew with the fetch of the next instruction to be executed.

Now let us consider how the branching instructions—the transfers of control—are simulated. All we need to do is adjust the value in the instruction counter appropriately. Therefore, the unconditional branch instruction (**`40`**) is simulated within the **`switch`** as

```
counter = operand;
```

The conditional "branch if accumulator is zero" instruction is simulated as

```
if ( accumulator == 0 )
   counter = operand;
```

At this point you should implement your Simpletron simulator and run each of the SML programs you wrote in Exercise 5.18. You may embellish SML with additional features and provide for these in your simulator.

Your simulator should check for various types of errors. During the program loading phase, for example, each number the user types into the Simpletron's **memory** must be in the range **-9999** to **+9999**. Your simulator should use a **while** loop to test that each number entered is in this range and, if not, keep prompting the user to reenter the number until the user enters a cor rect number.

```
REGISTERS:
accumulator          +0000
counter                 00
instructionRegister  +0000
operationCode           00
operand                 00

MEMORY:
       0     1     2     3     4     5     6     7     8     9
 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
50 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
```

Fig. 5.38 A sample dump.

During the execution phase, your simulator should check for various serious errors, such as attempts to divide by zero, attempts to execute invalid operation codes, accumulator overflows (i.e., arithmetic operations resulting in values larger than **+9999** or smaller than **-9999**) and the like. Such serious errors are called *fatal errors.* When a fatal error is detected, your simulator should print an error message such as

```
*** Attempt to divide by zero ***
*** Simpletron execution abnormally terminated ***
```

and should print a full computer dump in the format we have discussed previously. This will help the user locate the error in th e program.

```
// Exercise 5.19 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setfill;
using std::setw;
using std::setiosflags;
using std::resetiosflags;

const int SIZE = 100, MAX_WORD = 9999, MIN_WORD = -9999;
const long SENTINEL = -99999;
enum Commands { READ = 10, WRITE, LOAD = 20, STORE, ADD = 30, SUBTRACT,
                DIVIDE, MULTIPLY, BRANCH = 40, BRANCHNEG, BRANCHZERO, HALT };

void load( int * const );
void execute( int * const, int * const, int * const, int * const,
              int * const, int * const);
void dump( const int * const, int, int, int, int, int );
bool validWord( int );

int main()
{
   int memory[ SIZE ] = { 0 }, accumulator = 0, instructionCounter = 0,
       opCode = 0, operand = 0, instructionRegister = 0;

   load( memory );
   execute( memory, &accumulator, &instructionCounter, &instructionRegister,
            &opCode, &operand );
   dump( memory, accumulator, instructionCounter, instructionRegister,
        opCode, operand );

   return 0;
}

void load( int * const loadMemory )
{
   long instruction;
   int i = 0;

   cout << "***           Welcome to Simpletron           ***\n"
        << "*** Please enter your program one instruction ***\n"
        << "*** (or data word) at a time. I will type the ***\n"
        << "*** location number and a question mark (?).  ***\n"
        << "*** You then type the word for that location. ***\n"
        << "*** Type the sentinel -99999 to stop entering ***\n"
        << "*** your program.                             ***\n" << "00 ? ";
   cin >> instruction;

   while ( instruction != SENTINEL ) {

      if ( !validWord( instruction ) )
         cout << "Number out of range. Please enter again.\n";
      else
         loadMemory[ i++ ] = instruction;

      // function setfill sets the padding character for unused
      // field widths.
      cout << setw( 2 ) << setfill( '0' ) << i << " ? ";
      cin >> instruction;
   }
}

void execute( int * const memory, int * const acPtr, int * const icPtr,
              int * const irPtr, int * const opCodePtr, int * const opPtr )
{
   bool fatal = false;
   int temp;
   const char *messages[] = { "Accumulator overflow          ***",
                              "Attempt to divide by zero     ***",
                              "Invalid opcode detected       ***" },
        *termString = "\n*** Simpletron execution abnormally terminated ***",
        *fatalString = "*** FATAL ERROR: ";

   cout << "\n************START SIMPLETRON EXECUTION************\n\n";

   do {
      *irPtr = memory[ *icPtr ];
      *opCodePtr = *irPtr / 100;
      *opPtr = *irPtr % 100;

      switch ( *opCodePtr ) {
         case READ:
            cout << "Enter an integer: ";
            cin >> temp;

            while ( !validWord( temp ) ) {
               cout << "Number out of range. Please enter again: ";
               cin >> temp;
            }

            memory[ *opPtr ] = temp;
            ++( *icPtr );
            break;
         case WRITE:
            cout << "Contents of " << setw( 2 ) << setfill( '0' ) << *opPtr
                 << ": " << memory[ *opPtr ] << '\n';
            ++( *icPtr );
            break;
         case LOAD:
            *acPtr = memory[ *opPtr ];
            ++( *icPtr );
            break;
         case STORE:
            memory[ *opPtr ] = *acPtr;
            ++( *icPtr );
            break;
         case ADD:
            temp = *acPtr + memory[ *opPtr ];

            if ( !validWord( temp ) ) {
               cout << fatalString << messages[ 0 ] << termString << '\n';
               fatal = true;
            }
            else {
               *acPtr = temp;
               ++( *icPtr );
            }

            break;
         case SUBTRACT:
            temp = *acPtr - memory[ *opPtr ];

            if ( !validWord( temp ) ) {
               cout << fatalString << messages[ 0 ] << termString << '\n';
               fatal = true;
            }
            else {
               *acPtr = temp;
               ++( *icPtr );
            }

            break;
         case DIVIDE:
            if ( memory[ *opPtr ] == 0 ) {
               cout << fatalString << messages[ 1 ] << termString << '\n';
               fatal = true;
            }
            else {
               *acPtr /= memory[ *opPtr ];
               ++( *icPtr );
            }

            break;
         case MULTIPLY:
            temp = *acPtr * memory[ *opPtr ];

            if ( !validWord( temp ) ) {
               cout << fatalString << messages[ 0 ] << termString << '\n';
               fatal = true;
            }
            else {
               *acPtr = temp;
               ++( *icPtr );
            }
            break;
         case BRANCH:
            *icPtr = *opPtr;
            break;
         case BRANCHNEG:
            *acPtr < 0 ? *icPtr = *opPtr : ++( *icPtr );
            break;
         case BRANCHZERO:
            *acPtr == 0 ? *icPtr = *opPtr : ++( *icPtr );
            break;
         case HALT:
            cout << "*** Simpletron execution terminated ***\n";
            break;
         default:
            cout << fatalString << messages[ 2 ] << termString << '\n';
            fatal = true;
            break;
      }
   } while ( *opCodePtr != HALT && !fatal );

   cout << "\n*************END SIMPLETRON EXECUTION*************\n";
}

void dump( const int * const memory, int accumulator, int instructionCounter,
          int instructionRegister, int operationCode, int operand )
{
   void output( const char * const, int, int, bool );   // prototype

   cout << "\nREGISTERS:\n";
   output( "accumulator", 5, accumulator, true );
   output( "instructionCounter", 2, instructionCounter, false );
   output( "instructionRegister", 5, instructionRegister, true );
   output( "operationCode", 2, operationCode, false );
   output( "operand", 2, operand, false );
   cout << "\n\nMEMORY:\n";

   int i = 0;
   cout << setfill( ' ' ) << setw( 3 ) << ' ';

   // print header
   for ( ; i <= 9; ++i )
      cout << setw( 5 ) << i << ' ';

   for ( i = 0; i < SIZE; ++i ) {
      if ( i % 10 == 0 )
         cout << '\n' << setw( 2 ) << i << ' ';

      cout << setiosflags( ios::internal | ios::showpos )
           << setw( 5 ) << setfill( '0' ) << memory[ i ] << ' '
           << resetiosflags( ios::internal | ios::showpos );
   }

   cout << endl;
}

bool validWord( int word )
{
   return word >= MIN_WORD && word <= MAX_WORD;
}

void output( const char * const sPtr, int width, int value, bool sign )
{
   // format of "accumulator", etc.
   cout << setfill( ' ' ) << setiosflags( ios::left ) << setw( 20 )
        << sPtr << ' ';

   // is a +/- sign needed?
   if ( sign )
      cout << setiosflags( ios::showpos | ios::internal );

   // setup for displaying accumulator value, etc.
   cout << resetiosflags( ios::left ) << setfill( '0' );

   // determine the field widths and display value
   if ( width == 5 )
      cout << setw( width ) << value << '\n';
   else  // width is 2
      cout << setfill( ' ' ) << setw( 3 ) << ' ' << setw( width )
           << setfill( '0' ) << value << '\n';

   // disable sign if it was set
   if ( sign )
      cout << resetiosflags( ios::showpos | ios::internal );
}
```

```
***           Welcome to Simpletron           ***
*** Please enter your program one instruction ***
*** (or data word) at a time. I will type the ***
*** location number and a question mark (?).  ***
*** You then type the word for that location. ***
*** Type the sentinel -99999 to stop entering ***
*** your program.                              ***
00 ? 1099
01 ? 1098
02 ? 2099
03 ? 3398
04 ? 2150
05 ? 1150
06 ? 1199
07 ? 1198
08 ? 4300
09 ? -99999

************START SIMPLETRON EXECUTION************

Enter an integer: 4
Enter an integer: 9
Contents of 50: 36
Contents of 99: 4
Contents of 98: 9
*** Simpletron execution terminated ***

*************END SIMPLETRON EXECUTION*************

REGISTERS:
accumulator          +0036
instructionCounter      08
instructionRegister  +4300
operationCode           43
operand                 00


MEMORY:
       0     1     2     3     4     5     6     7     8     9
 0 +1099 +1098 +2099 +3398 +2150 +1150 +1199 +1198 +4300 +0000
10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
50 +0036 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0009 +0004
```

MORE POINTER EXERCISES

5.20 Modify the card-shuffling and dealing program of Fig. 5.24 so the shuffling and dealing operations are performed by the same function (`shuffleAndDeal`). The function should contain one nested looping structure that is similar to function `shuffle` in Fig. 5.24.

```
// Exercise 5.20 Solution
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setw;
using std::setiosflags;
using std::resetiosflags;

#include <cstdlib>
#include <ctime>

void shuffleAndDeal( int [][ 13 ], const char *[], const char *[] );

int main()
{
   const char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
   const char *face[ 13 ] = { "Ace", "Deuce", "Three", "Four", "Five",
                              "Six", "Seven", "Eight", "Nine", "Ten",
                              "Jack", "Queen", "King" };
   int deck[ 4 ][ 13 ] = { 0 };

   srand( time( 0 ) );
   shuffleAndDeal( deck, face, suit );

   return 0;
}

void shuffleAndDeal( int workDeck[][ 13 ], const char *workFace[],
                     const char *workSuit[] )
{
   int row, column;

   for ( int card = 1; card <= 52; ++card ) {

      do {
         row = rand() % 4;
         column = rand() % 13;
      } while ( workDeck[ row ][ column ] != 0 );

      workDeck[ row ][ column ] = card;
      cout << setw( 8 ) << workFace[ column ] << " of "
           << setiosflags( ios::left ) << setw( 8 )
           << workSuit[ row ] << resetiosflags( ios::left )
           << ( card % 2 == 0 ? '\n' : '\t' );
   }
}
```

```
    Six of Hearts           Ten of Diamonds
    Deuce of Spades          Five of Hearts
      Ace of Spades          Nine of Spades
    Seven of Diamonds        King of Spades
     Nine of Diamonds       Eight of Spades
     Five of Clubs          Seven of Spades
    Deuce of Clubs           Four of Clubs
      Ten of Hearts          Jack of Diamonds
     Nine of Clubs           Four of Diamonds
      Ace of Hearts          Five of Spades
     Jack of Hearts         Deuce of Diamonds
      Ten of Spades         Queen of Hearts
    Queen of Spades         Queen of Diamonds
     King of Clubs            Six of Spades
    Deuce of Hearts         Three of Spades
     King of Diamonds       Three of Hearts
     Four of Hearts          Jack of Spades
     Jack of Clubs          Three of Clubs
     Four of Spades         Three of Diamonds
      Ace of Diamonds        Nine of Hearts
    Seven of Clubs          Eight of Diamonds
    Eight of Clubs          Seven of Hearts
    Eight of Hearts         Queen of Clubs
      Six of Clubs            Ace of Clubs
     Five of Diamonds        King of Hearts
      Ten of Clubs            Six of Diamonds
```

5.21 What does this program do?

```
// ex05_21.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

void mystery1( char *, const char * );

int main()
{
   char string1[ 80 ], string2[ 80 ];

   cout << "Enter two strings: ";
   cin >> string1 >> string2;
   mystery1( string1, string2 );
   cout << string1 << endl;

   return 0;
}

void mystery1( char *s1, const char *s2 )
{
   while ( *s1 != '\0' )
      ++s1;

   for ( ; *s1 = *s2; s1++, s2++ )
      ;   // empty statement
}
```

```
Enter two strings: string1 string2
string1string2
```

5.22 What does this program do?

```
// ex05_22.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int mystery2( const char * );

int main()
{
   char string[ 80 ];

   cout << "Enter a string: ";
   cin >> string;
   cout << mystery2( string ) << endl;

   return 0;
}

int mystery2( const char *s )
{
   int x;

   for ( x = 0; *s != '\0'; s++ )
      ++x;

   return x;
}
```

```
Enter a string: length
6
```

5.23 Find the error in each of the following program segments. If the error can be corrected, explain how.

a)
```
int *number;
cout << number << endl;
```
ANS: Pointer `number` does not "point" to a valid address. Assigning `number` to an address would correct the problem.

b)
```
double *realPtr;
long *integerPtr;
integerPtr = realPtr;
```
ANS: A pointer of type `float` cannot be directly assigned to a pointer of type `long`.

c)
```
int * x, y;
x = y;
```
ANS: Variable `y` is not a pointer, and therefore cannot be assigned to `x`. Change the assignment statement to`x = &y;`.

d)
```
char s[] = "this is a character array";
for ( ; *s != '\0'; s++)
   cout << *s << ' ';
```
ANS: `s` is not a modifiable value. Attempting to use operator `++` is a syntax error. Changing to `[]` notation corrects the problem as in:

```
for ( int t = 0; s[ t ] != ’\0’; ++t )
   cout << s[ t ] << ’ ’;
```

e)
```
short *numPtr, result;
void *genericPtr = numPtr;
result = *genericPtr + 7;
```
ANS: A `void *` pointer cannot be dereferenced.

f)
```
double x = 19.34;
double xPtr = &x;
cout << xPtr << endl;
```
ANS: `xPtr` is not a pointer and therefore cannot be assigned an address. Place a `*` before `xPtr` (in the declaration) to correct the problem. The `cout` statement display's the address to which `xPtr` points (once the previous correction is made)—this is not an error.

g)
```
char *s;
cout << s << endl;
```
ANS: `s` does not "point" to anything. `s` should be provided with an address.

5.24 (*Quicksort*) In the examples and exercises of Chapter 4, we discussed the sorting techniques of the bubble sort, bucket sort, and selection sort. We now present the recursive sorting technique called Quicksort. The basic algorithm for a single-subscripted array of values is as follows:

a) *Partitioning Step:* Take the first element of the unsorted array and determine its final location in the sorted array (i.e., all values to the left of the element in the array are less than the element, and all values to the right of the element in the array are greater than the element). We now have one element in its proper location and two unsorted subarrays.
b) *Recursive Step:* Perform step 1 on each unsorted subarray.

Each time step 1 is performed on a subarray, another element is placed in its final location of the sorted array, and two unsorted subarrays are created. When a subarray consists of one element, it must be sorted, therefore that element is in its final location.

The basic algorithm seems simple enough, but how do we determine the final position of the first element of each subarray. As an example, consider the following set of values (the element in bold is the partitioning element—it will be placed in its final location in the sorted array):

37 2 6 4 89 8 10 12 68 45

a) Starting from the rightmost element of the array, compare each element with **37** until an element less than **37** is found. Then swap **37** and that element. The first element less than **37** is 12, so **37** and 12 are swapped. The new array is

   12 2 6 4 89 8 10 **37** 68 45

   Element 12 is in italic to indicate that it was just swapped with **37**.
b) Starting from the left of the array, but beginning with the element after 12, compare each element with **37** until an element greater than **37** is found. Then swap **37** and that element. The first element greater than **37** is 89, so **37** and 89 are swapped. The new array is

   12 2 6 4 **37** 8 10 *89* 68 45

c) Starting from the right, but beginning with the element before 89, compare each element with **37** until an element less than **37** is found. Then swap **37** and that element. The first element less than **37** is 10, so **37** and 10 are swapped. The new array is

   12 2 6 4 *10* 8 **37** 89 68 45

d) Starting from the left, but beginning with the element after 10, compare each element with **37** until an element greater than **37** is found. Then swap **37** and that element. There are no more elements greater than **37**, so when we compare **37** with itself, we know that **37** has been placed in its final location of the sorted array.

Once the partition has been applied on the array, there are two unsorted subarrays. The subarray with values less than 37 contains 12, 2, 6, 4, 10 and 8. The subarray with values greater than 37 contains 89, 68 and 45. The sort continues with both subarrays being partitioned in the same manner as the original array.

Based on the preceding discussion, write recursive function `quickSort` to sort a single-subscripted integer array. The function should receive as arguments an integer array, a starting subscript and an ending subscript. Function `partition` should be called by `quickSort` to perform the partitioning step.

```
// Exercise 5.24 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

#include <cstdlib>
#include <ctime>

const int SIZE = 10, MAX_NUMBER = 1000;

void quicksort( int * const, int, int );
void swap( int * const, int * const );

int main()
{
   int arrayToBeSorted[ SIZE ] = { 0 };
   int loop;

   srand( time( 0 ) );

   for ( loop = 0; loop < SIZE; ++loop )
      arrayToBeSorted[ loop ] = rand() % MAX_NUMBER;

   cout << "Initial array values are:\n";

   for ( loop = 0; loop < SIZE; ++loop )
       cout << setw( 4 ) << arrayToBeSorted[ loop ];

   cout << "\n\n";

   if ( SIZE == 1 )
      cout << "Array is sorted: " << arrayToBeSorted[ 0 ] << '\n';
   else {
      quicksort( arrayToBeSorted, 0, SIZE - 1 );
      cout << "The sorted array values are:\n";

      for ( loop = 0; loop < SIZE; ++loop )
         cout << setw( 4 ) << arrayToBeSorted[ loop ];

      cout << endl;
   }

   return 0;
}

void quicksort( int * const array, int first, int last )
{
   int partition( int * const, int, int );
   int currentLocation;

   if ( first >= last )
      return;

   currentLocation = partition( array, first, last );  // place an element
   quicksort( array, first, currentLocation - 1 );     // sort left side
   quicksort( array, currentLocation + 1, last );      // sort right side
}

int partition( int * const array, int left, int right )
{
   int position = left;

   while ( true ) {
      while ( array[ position ] <= array[ right ] && position != right )
         --right;

      if ( position == right )
         return position;

      if ( array[ position ] > array[ right ]) {
         swap( &array[ position ], &array[ right ] );
         position = right;
      }

      while ( array[ left ] <= array[ position ] && left != position )
         ++left;

      if ( position == left )
         return position;

      if ( array[ left ] > array[ position ] ) {
         swap( &array[ position ], &array[ left ] );
         position = left;
      }
   }
}

void swap( int * const ptr1, int * const ptr2 )
{
   int temp;

   temp = *ptr1;
   *ptr1 = *ptr2;
   *ptr2 = temp;
}
```

```
Initial array values are:
 407 766 451 328  17  50 551 620 192  35

The sorted array values are:
  17  35  50 192 328 407 451 551 620 766
```

5.25 (*Maze Traversal*) The following grid of hashes (**#**) and dots (**.**) is a double-subscripted array representation of a maze:

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

In the preceding double-subscripted array, the hashes (**#**), represent the walls of the maze and the dots represent squares in the possible paths through the maze. Moves can only be made to a location in the array that contains a dot.

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming that there is an exit). If there is not an exit, you will arrive at the starting location again. Place your right hand on the wall to your right and begin walking forward. Never remove your hand from the wall. If the maze turns to the right, you follow the wall to the right. As long as you do not remove your hand from the wall, eventually you will arrive at the exit of the maze. There may be a shorter path than the one you have taken, but you are guaranteed to get out of the maze if you follow the algorithm.

Write recursive function **mazeTraverse** to walk through the maze. The function should receive as arguments a 12-by-12 character array representing the maze and the starting location of the maze. As **mazeTraverse** attempts to locate the exit from the maze, it should place the character **X** in each square in the path. The function should display the maze after each move so the user can watch as the maze is solved.

```
// Exercise 5.25 Solution
// This solution assumes that there is only one
// entrance and one exit for a given maze, and
// these are the only two zeroes on the borders.
#include <iostream>

using std::cout;
using std::cin;

#include <cstdlib>

enum Direction { DOWN, RIGHT, UP, LEFT };
const int X_START = 2, Y_START = 0;   // starting coordinate for maze

void mazeTraversal( char [][ 12 ], int, int, int );
void printMaze( const char[][ 12 ] );
bool validMove( const char [][ 12 ], int, int );
bool coordsAreEdge( int, int );

int main()
{
   char maze[ 12 ][ 12 ] =
          { {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'},
            {'#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '#'},
            {'.', '.', '#', '.', '#', '.', '#', '#', '#', '#', '.', '#'},
            {'#', '#', '#', '.', '#', '.', '.', '.', '.', '#', '.', '#'},
            {'#', '.', '.', '.', '.', '#', '#', '#', '.', '#', '.', '.'},
            {'#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#'},
            {'#', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#'},
            {'#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#'},
            {'#', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#'},
            {'#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#'},
            {'#', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#'},
            {'#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'} };

   mazeTraversal( maze, X_START, Y_START, RIGHT );

   return 0;
}

// Assume that there is exactly 1 entrance and exactly 1 exit to the maze.
void mazeTraversal( char maze[][ 12 ], int xCoord, int yCoord, int direction )
{
   static bool flag = false;

   maze[ xCoord ][ yCoord ] = 'x';
   printMaze( maze );

   if ( coordsAreEdge(xCoord, yCoord) && xCoord != X_START &&
        yCoord != Y_START ) {
      cout << "\nMaze successfully exited!\n\n";
      return;   // maze is complete
   }
   else if ( xCoord == X_START && yCoord == X_START && flag ) {
      cout << "\nArrived back at the starting location.\n\n";
      return;
   }
   else {
      flag = true;

      for ( int move = direction, count = 0; count < 4; ++count,
            ++move, move %= 4 )
         switch( move ) {
            case DOWN:
               if ( validMove( maze, xCoord + 1, yCoord ) ) { // move down
                  mazeTraversal( maze, xCoord + 1, yCoord, LEFT );
                  return;
               }
               break;
            case RIGHT:
               if ( validMove( maze, xCoord, yCoord + 1 ) ) { // move right
                  mazeTraversal( maze, xCoord, yCoord + 1, DOWN );
                  return;
               }
               break;
            case UP:
               if ( validMove( maze, xCoord - 1, yCoord ) ) { // move up
                  mazeTraversal( maze, xCoord - 1, yCoord, RIGHT );
                  return;
               }
               break;
            case LEFT:
               if ( validMove( maze, xCoord, yCoord - 1 ) ) { // move left
                  mazeTraversal( maze, xCoord, yCoord - 1, UP );
                  return;
               }
               break;
         }
   }
}

bool validMove( const char maze[][ 12 ], int r, int c )
```

```
{
   return ( r >= 0 && r <= 11 && c >= 0 && c <= 11 && maze[ r ][ c ] != '#' );
}

bool coordsAreEdge( int x, int y )
{
   if ( ( x == 0 || x == 11 ) && ( y >= 0 && y <= 11 ) )
      return true;
   else if ( ( y == 0 || y == 11 ) && ( x >= 0 && x <= 11 ) )
      return true;
   else
      return false;
}

void printMaze( const char maze[][ 12 ] )
{
   for ( int x = 0; x < 12; ++x ) {

      for ( int y = 0; y < 12; ++y )
         cout << maze[ x ][ y ] << ' ';

      cout << '\n';
   }

   cout << "\nHit return to see next move\n";
   cin.get();
}
```

```
...
# # # # # # # # # # # #
# x x x # x x x x x x #
x x # x # x # # # # x #
# # # x # x x x x # x #
# x x x x # # # x # x .
# # # # x # . # x # x #
# x x # x # . # x # x #
# # x # x # . # x # x #
# x x x x x x x x # x #
# # # # # # x # # # x #
# x x x x x x # x x x #
# # # # # # # # # # # #

Hit return to see next move

# # # # # # # # # # # #
# x x x # x x x x x x #
x x # x # x # # # # x #
# # # x # x x x x # x #
# x x x x # # # x # x x
# # # # x # . # x # x #
# x x # x # . # x # x #
# # x # x # . # x # x #
# x x x x x x x x # x #
# # # # # # x # # # x #
# x x x x x x # x x x #
# # # # # # # # # # # #

Hit return to see next move


Maze successfully exited!
```

5.26 (*Generating Mazes Randomly*) Write a function **`mazeGenerator`** that takes as an argument a double-subscripted 12-by-12 character array and randomly produces a maze. The function should also provide the starting and ending locations of the maze. Try your function **`mazeTraverse`** from Exercise 5.25 using several randomly generated mazes.

```cpp
// Exercise 5.26 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <cstdlib>
#include <ctime>

enum Direction { DOWN, RIGHT, UP, LEFT };
const int MAX_DOTS = 100;  // maximum possible dots for maze

void mazeTraversal( char [][ 12 ], const int, const int, int, int, int );
void mazeGenerator( char [][ 12 ], int *, int * );
void printMaze( const char[][ 12 ] );
bool validMove( const char [][ 12 ], int, int );
bool coordsAreEdge( int, int );

int main()
{
   char maze[ 12 ][ 12 ];
   int xStart, yStart, x, y;

   srand( time( 0 ) );

   for ( int loop = 0; loop < 12; ++loop )
      for ( int loop2 = 0; loop2 < 12; ++loop2 )
         maze[ loop ][ loop2 ] = '#';

   mazeGenerator( maze, &xStart, &yStart );

   x = xStart;  // starting row
   y = yStart;  // starting col

   mazeTraversal( maze, xStart, yStart, x, y, RIGHT );

   return 0;
}

// Assume that there is exactly 1 entrance and exactly 1 exit to the maze.
void mazeTraversal( char maze[][ 12 ], const int xCoord, const int yCoord,
                    int row, int col, int direction )
{
   static bool flag = false;   // starting position flag

   maze[ row ][ col ] = 'x';  // insert X at current location
   printMaze( maze );

   if ( coordsAreEdge( row, col ) && row != xCoord && col != yCoord ) {
      cout << "Maze successfully exited!\n\n";
      return;   // maze is complete
   }
   else if ( row == xCoord && col == yCoord && flag ) {
      cout << "Arrived back at the starting location.\n\n";
      return;
   }
   else {
      flag = true;

      for ( int move = direction, count = 0; count < 4;
            ++count, ++move, move %= 4 )

         switch( move ) {
            case DOWN:
               if ( validMove( maze, row + 1, col ) ) { // move down
                  mazeTraversal( maze, xCoord, yCoord, row + 1, col, LEFT );
                  return;
               }
               break;
            case RIGHT:
               if ( validMove( maze, row, col + 1 ) ) { // move right
                  mazeTraversal( maze, xCoord, yCoord, row, col + 1, DOWN );
                  return;
               }
               break;
            case UP:
               if ( validMove( maze, row - 1, col ) ) { // move up
                  mazeTraversal( maze, xCoord, yCoord, row - 1, col, RIGHT );
                  return;
               }
               break;
            case LEFT:
               if ( validMove( maze, row, col - 1 ) ) { // move left
```

```
                  mazeTraversal( maze, xCoord, yCoord, row, col - 1, UP );
                  return;
               }
               break;
         }
   }
}

bool validMove( const char maze[][ 12 ], int r, int c )
{
   return ( r >= 0 && r <= 11 && c >= 0 && c <= 11 && maze[ r ][ c ] != '#' );
}

bool coordsAreEdge( int x, int y )
{
   if ( ( x == 0 || x == 11 ) && ( y >= 0 && y <= 11 ) )
      return true;
   else if ( ( y == 0 || y == 11 ) && ( x >= 0 && x <= 11 ) )
      return true;
   else
      return false;
}

void printMaze( const char maze[][ 12 ] )
{
   for ( int x = 0; x < 12; ++x ) {

      for ( int y = 0; y < 12; ++y )
         cout << maze[ x ][ y ] << ' ';

      cout << '\n';
   }

   cout << "Hit return to see next move";
   cin.get();
}

void mazeGenerator(char maze[][ 12 ], int *xPtr, int *yPtr )
{
   int a, x, y, entry, exit;

   do {
      entry = rand() % 4;
      exit = rand() % 4;
   } while ( entry == exit );

   // Determine entry position

   if ( entry == 0 ) {
      *xPtr = 1 + rand() % 10;    // avoid corners
      *yPtr = 0;
      maze[ *xPtr ][ 0 ] = '.';
   }
   else if ( entry == 1 ) {
      *xPtr = 0;
      *yPtr = 1 + rand() % 10;
      maze[ 0 ][ *yPtr ] = '.';
   }
   else if ( entry == 2 ) {
      *xPtr = 1 + rand() % 10;
      *yPtr = 11;
      maze[ *xPtr ][ 11 ] = '.';
   }
   else {
      *xPtr = 11;
      *yPtr = 1 + rand() % 10;
      maze[ 11 ][ *yPtr ] = '.';
   }

   // Determine exit location

   if ( exit == 0 ) {
      a = 1 + rand() % 10;
      maze[ a ][ 0 ] = '.';
   }
   else if ( exit == 1 ) {
      a = 1 + rand() % 10;
      maze[ 0 ][ a ] = '.';
   }
   else if ( exit == 2 ) {
      a = 1 + rand() % 10;
      maze[ a ][ 11 ] = '.';
   }
   else {
      a = 1 + rand() % 10;
      maze[ 11 ][ a ] = '.';
   }

   for ( int loop = 1; loop < MAX_DOTS; ++loop ) {   // add dots randomly
      x = 1 + rand() % 10;
      y = 1 + rand() % 10;
      maze[ x ][ y ] = '.';
   }
}
```

```
 ...
# # # # # # # # # # # #
# # . # # # x # # # # #
# # . . . # x x x # x #
# . # # # x x x x x x #
# . . # # # # x . x x #
# # . # x x x x # x # .
# # . # x x x # # x x #
# # # x x # x . . . x #
# # x x x # x x x x x #
# # # # # x # x x # x #
# . # x x x x x # # # #
# # # # # # # x # # # #
Hit return to see next move
# # # # # # # # # # # #
# # . # # # x # # # # #
# # . . . # x x x # x #
# . # # # x x x x x x #
# . . # # # # x . x x #
# # . # x x x x # x # .
# # . # x x x # # x x #
# # # x x # x . . . x #
# # x x x # x x x x x #
# # # # # x # x x # x #
# . # x x x x x # # # #
# # # # # # # x # # # #
Hit return to see next move
Arrived back at the starting location.
```

5.27 (*Mazes of Any Size*) Generalize functions **mazeTraverse** and **mazeGenerator** of Exercises 5.25 and 5.26 to process mazes of any width and height.

```
// Exercise 5.27 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>

enum Direction { DOWN, RIGHT, UP, LEFT };
const int ROWS = 15, COLS = 30;

void mazeTraversal( char [][ COLS ], const int, const int, int, int, int );
void mazeGenerator( char [][ COLS ], int *, int * );
void printMaze( const char[][ COLS ] );
bool validMove( const char [][ COLS ], int, int );
bool coordsAreEdge( int, int );

int main()
{
   char maze[ ROWS ][ COLS ];
   int xStart, yStart, x, y;

   srand( time( 0 ) );

   for ( int loop = 0; loop < ROWS; ++loop )
      for ( int loop2 = 0; loop2 < COLS; ++loop2 )
```

```
         maze[ loop ][ loop2 ] = '#';

   mazeGenerator( maze, &xStart, &yStart );

   x = xStart;  // starting row
   y = yStart;  // starting col

   mazeTraversal( maze, xStart, yStart, x, y, RIGHT );

   return 0;
}

// Assume that there is exactly 1 entrance and exactly 1 exit to the maze.
void mazeTraversal( char maze[][ COLS ], const int xCoord, const int yCoord,
                    int row, int col, int direction )
{
   static bool flag = false;   // starting position flag

   maze[ row ][ col ] = 'x';  // insert x at current location
   printMaze( maze );

   if ( coordsAreEdge( row, col ) && row != xCoord && col != yCoord ) {
      cout << endl << "Maze successfully exited!\n\n";
      return;   // maze is complete
   }
   else if ( row == xCoord && col == yCoord && flag ) {
      cout << "\nArrived back at the starting location.\n\n";
      return;
   }
   else {
      flag = true;

      for ( int move = direction, count = 0; count < 4;
            ++count, ++move, move %= 4 )
         switch( move ) {
            case DOWN:
               if ( validMove( maze, row + 1, col ) ) { // move down
                  mazeTraversal( maze, xCoord, yCoord, row + 1, col, LEFT );
                  return;
               }
               break;
            case RIGHT:
               if ( validMove( maze, row, col + 1 ) ) { // move right
                  mazeTraversal( maze, xCoord, yCoord, row, col + 1, DOWN );
                  return;
               }
               break;
            case UP:
               if ( validMove( maze, row - 1, col ) ) { // move up
                  mazeTraversal( maze, xCoord, yCoord, row - 1, col, RIGHT );
                  return;
               }
               break;
            case LEFT:
               if ( validMove( maze, row, col - 1 ) ) { // move left
                  mazeTraversal( maze, xCoord, yCoord, row, col - 1, UP );
                  return;
               }
               break;
         }
   }
}
```

```

bool validMove( const char maze[][ COLS ], int r, int c )
{
   return ( r >= 0 && r <= ROWS - 1 && c >= 0 && c <= COLS - 1 &&
            maze[ r ][ c ] != '#' );  // a valid move
}

bool coordsAreEdge( int x, int y )
{
   if ( ( x == 0 || x == ROWS - 1 ) && ( y >= 0 && y <= COLS - 1 ) )
      return true;
   else if ( ( y == 0 || y == COLS - 1 ) && ( x >= 0 && x <= ROWS - 1 ) )
      return true;
   else
      return false;
}

void printMaze( const char maze[][ COLS ] )
{
   for ( int x = 0; x < ROWS; ++x ) {

      for ( int y = 0; y < COLS; ++y )
         cout << maze[ x ][ y ] << ' ';

      cout << '\n';
   }

   cout << "\nHit return to see next move";
   cin.get();
}

void mazeGenerator( char maze[][ COLS ], int *xPtr, int *yPtr )
{
   int a, x, y, entry, exit;

   do {
      entry = rand() % 4;
      exit = rand() % 4;
   } while ( entry == exit );

   // Determine entry position
   if ( entry == 0 ) {
      *xPtr = 1 + rand() % ( ROWS - 2 );    // avoid corners
      *yPtr = 0;
      maze[ *xPtr ][ *yPtr ] = '.';
   }
   else if ( entry == 1 ) {
      *xPtr = 0;
      *yPtr = 1 + rand() % ( COLS - 2 );
      maze[ *xPtr ][ *yPtr ] = '.';
   }
   else if ( entry == 2 ) {
      *xPtr = 1 + rand() % ( ROWS - 2 );
      *yPtr = COLS - 1;
      maze[ *xPtr ][ *yPtr ] = '.';
   }
   else {
      *xPtr = ROWS - 1;
      *yPtr = 1 + rand() % ( COLS - 2 );
      maze[ *xPtr ][ *yPtr ] = '.';
   }

   // Determine exit location
   if ( exit == 0 ) {
      a = 1 + rand() % ( ROWS - 2 );
      maze[ a ][ 0 ] = '.';
   }
   else if ( exit == 1 ) {
      a = 1 + rand() % ( COLS - 2 );
      maze[ 0 ][ a ] = '.';
   }
   else if ( exit == 2 ) {
      a = 1 + rand() % ( ROWS - 2 );
      maze[ a ][ COLS - 1 ] = '.';
   }
   else {
      a = 1 + rand() % ( COLS - 2 );
      maze[ ROWS - 1 ][ a ] = '.';
   }

   for ( int loop = 1; loop < ( ROWS - 2 ) * ( COLS - 2 ); ++loop ) {
      x = 1 + rand() % ( ROWS - 2 );    // add dots to maze
      y = 1 + rand() % ( COLS - 2 );
      maze[ x ][ y ] = '.';
   }
}
```

```
...
Hit return to see next move
# # # # # # # # # # # # # # . # # # # # # # # # # # # # # #
# . . # . . . . . . # . . # x x x # x x x x x x # . # . # #
# . . . . # # . # # # # # . . . x x x # # x x x x # . . # #
# . # . # . # # . # # . # # . . # . # . # x # # # . . # # #
# . # # . . . # # . . # # . . . . # . . x x x x x # . # # #
# . # # # . . . # . # . # # . . . . # . x # # x x x # . # #
# # # . # . . . . . # . # # . . # # # . x # . # x # . . # #
# . x x x x x x . # # . . . . . # . # . x x # x # . # . # #
# x x # # x # x . . . . # # . . # # . # # x x x x # . . . #
# x # # x x # x x . . . . . . . . # . . # . # # x # . . . #
x x # x x x x # x x x . . x x x x x x x . x x x x x # # . #
# # x x x # # x # # x . # x # x x # # x x x # x # x # # # #
# # x x # . # x # x x x x x x # x # # # # # x x x # # . . #
# . # x # . # x x x x # x # # # x x x # # x x # x # . . . #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

Hit return to see next move
# # # # # # # # # # # # # # x # # # # # # # # # # # # # # #
# . . # . . . . . . # . . # x x x # x x x x x x # . # . # #
# . . . . # # . # # # # # . . . x x x # # x x x x # . . # #
# . # . # . # # . # # . # # . . # . # . # x # # # . . # # #
# . # # . . . # # . . # # . . . . # . . x x x x x # . # # #
# . # # # . . . # . # . # # . . . . # . x # # x x x # . # #
# # # . # . . . . . # . # # . . # # # . x # . # x # . . # #
# . x x x x x x . # # . . . . . # . # . x x # x # . # . # #
# x x # # x # x . . . . # # . . # # . # # x x x x # . . . #
# x # # x x # x x . . . . . . . . # . . # . # # x # . . . #
x x # x x x x # x x x . . x x x x x x x . x x x x x # # . #
# # x x x # # x # # x . # x # x x # # x x x # x # x # # # #
# # x x # . # x # x x x x x x # x # # # # # x x x # # . . #
# . # x # . # x x x x # x # # # x x x # # x x # x # . . . #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

Hit return to see next move

Maze successfully exited!
```

5.28 (*Arrays of Pointers to Functions*) Rewrite the program of Fig. 4.23 to use a menu-driven interface. The program should offer the user five options as follows (these should be displayed on the screen):

```
Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
```

One restriction on using arrays of pointers to functions is that all the pointers must have the same type. The pointers must be to functions of the same return type that receive arguments of the same type. For this reason, the functions in Fig. 4.23 must be modified so they each return the same type and take the same parameters. Modify functions **`minimum`** and **`maximum`** to print the minimum or maximum value and return nothing. For option 3, modify function **`average`** of Fig. 4.23 to output the average for each student (not a specific student). Function **`average`** should return nothing and take the same parameters as **`printArray`**, **`minimum`** and **`maximum`**. Store the pointers to the four functions in array **`processGrades`**, and use the choice made by the user as the subscript into the array for calling each function.

```
// Exercise 5.28 Solution
#include <iostream>

using std::cout;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

const int STUDENTS = 3, EXAMS = 4;

void minimum( const int [][ EXAMS ], int, int );
void maximum( const int [][ EXAMS ], int, int );
void average( const int [][ EXAMS ], int, int );
void printArray( const int [][ EXAMS ], int, int );
void printMenu( void );
int test( bool ( * )( int, int ), int, int, int, const int [][ EXAMS ] );

int main()
{
   void ( *processGrades[ 4 ] )( const int [][ EXAMS ], int, int )
                     = { printArray, minimum, maximum, average };

   int choice = 0,
       studentGrades[ STUDENTS ][ EXAMS ] = { { 77, 68, 86, 73 },
                                              { 96, 87, 89, 78 },
                                              { 70, 90, 86, 81 } };

   while ( choice != 4 ) {

      do {
         printMenu();
         cin >> choice;
      } while ( choice < 0 || choice > 4 );

      if ( choice != 4 )
         ( *processGrades[ choice ] )( studentGrades, STUDENTS, EXAMS );
      else
         cout << "Program Ended.\n";
   }

   return 0;
}

void minimum( const int grades[][ EXAMS ], int pupils, int tests )
{
   bool smaller( int, int );

   cout << "\n\tThe lowest grade is "
        << test( smaller, pupils, tests, 100, grades ) << '\n';
}

void maximum( const int grades[][ EXAMS ], int pupils, int tests )
{
   bool greater( int, int );

   cout << "\n\tThe highest grade is "
        << test( greater, pupils, tests, 0, grades ) << '\n';
}

int test( bool ( *ptr )( int, int ), int p, int t, int value,
          const int g[][ EXAMS ] )
{
    for ( int i = 0; i < p; ++i )
      for ( int j = 0; j < t; ++j )
         if ( ( *ptr )( g[ i ][ j ], value ) )
            value = g[ i ][ j ];

    return value;   // return max/min value
}

void average( const int grades[][ EXAMS ], int pupils, int tests )
{
   int total;

   cout << setiosflags( ios::fixed | ios::showpoint ) << '\n';

   for ( int i = 0; i < pupils; ++i ) {
      total = 0;  // reset total

      for ( int j = 0; j < tests; ++j )
         total += grades[ i ][ j ];

      cout << "\tThe average for student " << pupils + 1 << " is "
           << setprecision( 1 )
           << static_cast< double > ( total ) / tests << '\n';
   }

   cout << resetiosflags( ios::fixed | ios::showpoint );
}

void printArray( const int grades[][ EXAMS ], int pupils, int tests )
{
   cout << "\n                  [0]  [1]  [2]  [3]";

   for ( int i = 0; i < pupils; ++i ) {
      cout << "\nstudentGrades[" << i << ']';

      for ( int j = 0; j < tests; ++j )
         cout << setw( 5 ) << grades[ i ][ j ];
   }

   cout << '\n';
}

void printMenu( void )
{
   cout << "\nEnter a choice:\n"
        << "  0  Print the array of grades\n"
        << "  1  Find the minimum grade\n"
        << "  2  Find the maximum grade\n"
        << "  3  Print the average on all tests for each student\n"
        << "  4  End program\n? ";
}

bool greater( int a, int b )
{
   return a > b;
}

bool smaller( int a, int b )
{
   return a < b;
}
```

```
Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
? 0

                    [0]  [1]  [2]  [3]
studentGrades[0]    77   68   86   73
studentGrades[1]    96   87   89   78
studentGrades[2]    70   90   86   81

Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
? 1

        The lowest grade is 68

Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
? 2

        The highest grade is 96

Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
? 3

        The average for student 4 is 76.0
        The average for student 4 is 87.5
        The average for student 4 is 81.8

Enter a choice:
  0  Print the array of grades
  1  Find the minimum grade
  2  Find the maximum grade
  3  Print the average on all tests for each student
  4  End program
? 4
Program Ended.
```

5.29 (*Modifications to the Simpletron Simulator*) In Exercise 5.19, you wrote a software simulation of a computer that executes programs written in Simpletron Machine Language (SML). In this exercise, we propose several modifications and enhancements to the Simpletron Simulator. In Exercises 15.26 and 15.27, we propose building a compiler that converts programs written in a high-level programming language (a variation of BASIC) to SML. Some of the following modifications and enhancements may be required to execute the programs produced by the compiler. (*Note:* Some modifications may conflict with others and therefore must be done separately.)

a) Extend the Simpletron Simulator's memory to contain 1000 memory locations to enable the Simpletron to handle larger programs.
b) Allow the simulator to perform modulus calculations. This requires an additional Simpletron Machine Language instruction.
c) Allow the simulator to perform exponentiation calculations. This requires an additional Simpletron Machine Language instruction.
d) Modify the simulator to use hexadecimal values rather than integer values to represent Simpletron Machine Language instructions.
e) Modify the simulator to allow output of a newline. This requires an additional Simpletron Machine Language instruction.
f) Modify the simulator to process floating-point values in addition to integer values.
g) Modify the simulator to handle string input. [*Hint:* Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII decimal equivalent of a character. Add a machine-language instruction that will input a string and store the string beginning at a specific Simpletron memory location. The first half of the word at that location will be a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction converts each character into its ASCII equivalent and assigns it to a half-word.]
h) Modify the simulator to handle output of strings stored in the format of part (g). [*Hint:* Add a machine language instruction that will print a string beginning at a certain Simpletron memory location. The first half of the word at that location is a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction checks the length and prints the string by translating each two-digit number into its equivalent character.]
i) Modify the simulator to include instruction **SML_DEBUG** that prints a memory dump after each instruction is executed. Give **SML_DEBUG** an operation code of **44**. The word **+4401** turns on debug mode and **+4400** turns off debug mode.

5.30 What does this program do?

```
// ex05_30.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

bool mystery3( const char *, const char * );

int main()
{
   char string1[ 80 ], string2[ 80 ];

   cout << "Enter two strings: ";
   cin >> string1 >> string2;
   cout << "The result is "
        << mystery3( string1, string2 ) << endl;

   return 0;
}

bool mystery3( const char *s1, const char *s2 )
{
   for ( ; *s1 != '\0' && *s2 != '\0'; s1++, s2++ )

      if ( *s1 != *s2 )
         return false;

   return true;
}
```

ANS: Function **mystery3** compares two strings for equality.

STRING MANIPULATION EXERCISES

5.31 Write a program that uses function **strcmp** to compare two strings input by the user. The program should state whether the first string is less than, equal to or greater than the second string.

```
// Exercise 5.31 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

const int SIZE = 20;

int main()
{
   char string1[ SIZE ], string2[ SIZE ];
   int result;

   cout << "Enter two strings: ";
   cin >> string1 >> string2;

   result = strcmp( string1, string2 );

   if ( result > 0 )
      cout << '\"' << string1 << '\"' << " is greater than \""
           << string2 << '\"' << endl;
   else if ( result == 0 )
      cout << '\"' << string1 << '\"' << " is equal to \"" << string2
           << '\"' << endl;
   else
      cout << '\"' << string1 << '\"' << " is less than \"" << string2
           << '\"' << endl;

   return 0;
}
```

```
Enter two strings: green leaf
"green" is less than "leaf"
```

5.32 Write a program that uses function**strncmp** to compare two strings input by the user. The program should input the number of characters to be compared. The program should state whether the first string is less than, equal to or greater than the s econd string.

```
// Exercise 5.32 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

const int SIZE = 20;

int main()
{
   char string1[ SIZE ], string2[ SIZE ];
   int result, compareCount;

   cout << "Enter two strings: ";
   cin >> string1 >> string2;
   cout << "How many characters should be compared: ";
   cin >> compareCount;

   result = strncmp( string1, string2, compareCount );

   if ( result > 0 )
      cout << '\"' << string1 << "\" is greater than \"" << string2
           << "\" up to " << compareCount << " characters\n";
   else if ( result == 0 )
      cout << '\"' << string1 << "\" is equal to \"" << string2
           << "\" up to " << compareCount << " characters\n";
   else
      cout << '\"' << string1 << "\" is less than \"" << string2
           << "\" up to " << compareCount << " characters\n";

   cout << endl;

   return 0;
}
```

```
Enter two strings: sand sandpaper
How many characters should be compared: 4
"sand" is equal to "sandpaper" up to 4 characters
```

5.33 Write a program that uses random-number generation to create sentences. The program should use four arrays of pointers to **char** called **article**, **noun**, **verb** and **preposition**. The program should create a sentence by selecting a word at random from each array in the following order: **article**, **noun**, **verb**, **preposition**, **article** and **noun**. As each word is picked, it should be concatenated to the previous words in an array that is large enough to hold the entire sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The program should generate 20 such sentences.

The arrays should be filled as follows: the **article** array should contain the articles **"the"**, **"a"**, **"one"**, **"some"** and **"any"**; the **noun** array should contain the nouns **"boy"**, **"girl"**, **"dog"**, **"town"** and **"car"**; the **verb** array should contain the verbs **"drove"**, **"jumped"**, **"ran"**, **"walked"** and **"skipped"**; the **preposition** array should contain the prepositions **"to"**, **"from"**, **"over"**, **"under"** and **"on"**.

After the preceding program is written and working, modify the program to produce a short story consisting of several of these sentences. (How about the possibility of a random term paper writer!)

```
// Exercise 5.33 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <cstdlib>
#include <ctime>

#include <cstring>
#include <cctype>

const int SIZE = 100;

int main()
{
   const char *article[] = { "the", "a", "one", "some", "any" },
        *noun[] = { "boy", "girl", "dog", "town", "car" },
        *verb[] = { "drove", "jumped", "ran", "walked", "skipped" },
        *preposition[] = { "to", "from", "over", "under", "on" };
   char sentence[ SIZE ] = "";

   for ( int i = 1; i <= 20; ++i ) {
      strcat( sentence, article[ rand() % 5 ] );
      strcat( sentence, " " );
      strcat( sentence, noun[ rand() % 5 ] );
      strcat( sentence, " " );
      strcat( sentence, verb[ rand() % 5 ] );
      strcat( sentence, " " );
      strcat( sentence, preposition[ rand() % 5 ] );
      strcat( sentence, " " );
      strcat( sentence, article[ rand() % 5 ] );
      strcat( sentence, " " );
      strcat( sentence, noun[ rand() % 5 ] );
      cout << static_cast< char > ( toupper( sentence[ 0 ] ) )
           << &sentence[ 1 ] << ".\n";
      sentence[ 0 ] = '\0';
   }

   cout << endl;

   return 0;
}
```

```
A dog skipped to any car.
Some town ran on the boy.
A dog jumped from the dog.
One girl jumped on one town.
One dog jumped from some boy.
One girl jumped under any dog.
One car drove on some girl.
One town walked on a girl.
Some town ran on one dog.
One car walked from any town.
A boy drove over some girl.
The dog skipped under a boy.
The car drove to a girl.
Some town skipped under any car.
A boy jumped from a town.
Any car jumped under one town.
Some dog skipped from some boy.
Any town skipped to one girl.
Some girl jumped to any dog.
The car ran under one dog.
```

5.34 *(Limericks)* A limerick is a humorous five-line verse in which the first and second lines rhyme with the fifth, and the third line rhymes with the fourth. Using techniques similar to those developed in Exercise 5.33, write a C++ program that produces random limericks. Polishing this program to produce good limericks is a challenging problem, but the result will be worth the effort!

5.35 Write a program that encodes English language phrases into pig Latin. Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following algorithm:

To form a pig-Latin phrase from an English-language phrase, tokenize the phrase into words with function `strtok`. To translate each English word into a pig-Latin word, place the first letter of the English word at the end of the English word, and add the letters "`ay`." Thus the word "`jump`" becomes "`umpjay`," the word "`the`" becomes "`hetay`" and the word "`computer`" becomes "`omputercay`." Blanks between words remain as blanks. Assume that the: the English phrase consists of words separated by blanks, there are no punctuation marks and all words have two or more letters. Function `printLatinWord` should display each word. (*Hint:* Each time a token is found in a call to `strtok`, pass the token pointer to function `printLatinWord`, and print the pig Latin word.)

```
// Exercise 5.35 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

const int SIZE = 80;

void printLatinWord( char const * const );

int main()
{
   char sentence[ SIZE ], *tokenPtr;

   cout << "Enter a sentence:\n";
   cin.getline( sentence, SIZE );

   cout << "\nThe sentence in Pig Latin is:\n";
   tokenPtr = strtok( sentence, " .,;" );

   while ( tokenPtr ) {
      printLatinWord( tokenPtr );
      tokenPtr = strtok( 0, " .,;" );

      if ( tokenPtr )
         cout << ' ';
   }

   cout << '.' << endl;

   return 0;
}

void printLatinWord( char const * const wordPtr )
{
   int len = strlen( wordPtr );
   for (int i = 1; i < len; ++i )
      cout << *( wordPtr + i );

   cout << *wordPtr << "ay";
}
```

```
Enter a sentence:
mirror mirror on the wall

The sentence in Pig Latin is:
irrormay irrormay noay hetay allway.
```

5.36 Write a program that inputs a telephone number as a string in the form **(555) 555-5555**. The program should use function **strtok** to extract the area code as a token, the first three digits of the phone number as a token, and the last four digits of the phone number as a token. The seven digits of the phone number should be concatenated into one string. The program should convert the area code string to **int** and convert the phone number string to **long**. Both the area code and the phone number should be printed.

```
// Exercise 5.36 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>
#include <cstdlib>

int main()
{
   const int SIZE1 = 20, SIZE2 = 10;
   char p[ SIZE1 ],
        phoneNumber[ SIZE2 ] = { '\0' }, *tokenPtr;
   int  areaCode;
   long phone;

   cout << "Enter a phone number in the form (555) 555-5555:\n";
   cin.getline( p, SIZE1 );

   areaCode = atoi( strtok( p, "()" ) );

   tokenPtr = strtok( 0, "-" );
   strcpy( phoneNumber, tokenPtr );
   tokenPtr = strtok( 0, "" );
   strcat( phoneNumber, tokenPtr );
   phone = atol( phoneNumber );

   cout << "\nThe integer area code is " << areaCode
        << "\nThe long integer phone number is " << phone << endl;

   return 0;
}
```

```
Enter a phone number in the form (555) 555-5555:
(555) 492-4195

The integer area code is 555
The long integer phone number is 4924195
```

5.37 Write a program that inputs a line of text, tokenizes the line with function **strtok** and outputs the tokens in reverse order.

```
// Exercise 5.37 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

void reverseTokens( char * const );
```

```

int main()
{
   const int SIZE = 80;
   char text[ SIZE ];

   cout << "Enter a line of text:\n";
   cin.getline( text, SIZE );
   reverseTokens( text );
   cout << endl;

   return 0;
}

void reverseTokens( char * const sentence )
{
   char *pointers[ 50 ], *temp;
   int count = 0;

   temp = strtok( sentence, " " );

   while ( temp ) {
      pointers[ count++ ] = temp;
      temp = strtok( 0, " " );
   }

   cout << "\nThe tokens in reverse order are:\n";

   for ( int i = count - 1; i >= 0; --i )
      cout << pointers[ i ] << ' ';
}
```

```
Enter a line of text:
twinkle twinkle little star

The tokens in reverse order are:
star little twinkle twinkle
```

5.38 Use the string comparison functions discussed in Section 5.12.2 and the techniques for sorting arrays developed in Chapter 4 to write a program that alphabetizes a list of strings. Use the names of 10 or 15 towns in your area as data for your program.

```
// Exercise 5.38 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

const int SIZE = 50;
void bubbleSort( char [][ SIZE ] );

int main()
{
   char array[ 10 ][ SIZE ];
   int i;

   for ( i = 0; i < 10; ++i ) {
      cout << "Enter a string: ";
      cin >> &array[ i ][ 0 ];
   }

   bubbleSort( array );
   cout << "\nThe strings in sorted order are:\n";

   for ( i = 0; i < 10; ++i )
      cout << &array[ i ][ 0 ] << endl;

   return 0;
}

void bubbleSort( char a[][ SIZE ] )
{
   char temp[ SIZE ];

   for ( int i = 0; i <= 8; ++i )
      for ( int j = 0; j <= 8; ++j )
         if ( strcmp( &a[ j ][ 0 ], &a[ j + 1 ][ 0 ] ) > 0 ) {
            strcpy( temp, &a[ j ][ 0 ] );
            strcpy( &a[ j ][ 0 ], &a[ j + 1 ][ 0 ] );
            strcpy(&a[ j + 1 ][ 0 ], temp );
         }
}
```

```
Enter a string: Windsor
Enter a string: Pittsford
Enter a string: Warren
Enter a string: Killington
Enter a string: Marlboro
Enter a string: Grafton
Enter a string: Middlebury
Enter a string: Barre
Enter a string: Montpelier
Enter a string: Wolcott

The strings in sorted order are:
Barre
Grafton
Killington
Marlboro
Middlebury
Montpelier
Pittsford
Warren
Windsor
Wolcott
```

5.39 Write two versions of each of the string copy and string concatenation functions in Fig. 5.29. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

```
// Exercise 5.39 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

```

```
char *stringCopy1( char *, const char * );
char *stringCopy2( char *, const char * );
char *stringNCopy1( char *, const char *, unsigned );
char *stringNCopy2( char *, const char *, unsigned );
char *stringCat1( char *, const char * );
char *stringCat2( char *, const char * );
char *stringNCat1( char *, const char *, unsigned );
char *stringNCat2( char *, const char *, unsigned );

int main()
{
   int n = 4;
   char string1[ 100 ], string2[ 100 ];

   cout << "Enter a string: ";
   cin >> string2;

   cout << "Copied string returned from stringCopy1 is "
        << stringCopy1( string1, string2 )
        << "\nCopied string returned from stringCopy2 is "
        << stringCopy2( string1, string2 );
   cout << "\nCopied " << n << " elements returned from stringNCopy1 is "
        << stringNCopy1( string1, string2, n );
   cout << "\nCopied " << n << " elements returned from stringNCopy2 is "
        << stringNCopy2(string1, string2, n);
   cout << "\nConcatenated string returned from stringCat1 is "
        << stringCat1( string1, string2 );
   cout << "\nConcatenated string returned from stringCat2 is "
        << stringCat2( string1, string2 );
   cout << "\nConcatenated string returned from stringNCat1 is "
        << stringNCat1( string1, string2, n );
   cout << "\nConcatenated string returned from stringNCat2 is "
        << stringNCat2( string1, string2, n ) << endl;

   return 0;
}

char *stringCopy1( char *s1, const char *s2 )
{
   for ( int sub = 0; s1[ sub ] = s2[ sub ]; ++sub )
      ; // empty body

   return s1;
}

char *stringCopy2( char *s1, const char *s2 )
{
   char *ptr = s1;

   for ( ; *s1 = *s2; ++s1, ++s2 )
      ; // empty body

   return ptr;
}

char *stringNCopy1( char *s1, const char *s2, unsigned n )
{
   unsigned c;

   for ( c = 0; c < n && ( s1[ c ] = s2[ c ] ); ++c )
      ; // empty body

   s1[ c ] = '\0';
   return s1;
}

char *stringNCopy2( char *s1, const char *s2, unsigned n )
{
   char *ptr = s1;

   for ( unsigned c = 0; c < n; ++c, ++s1, ++s2 )
      *s1 = *s2;

   *s1 = '\0';
   return ptr;
}

char *stringCat1( char *s1, const char *s2 )
{
   int x;

   for ( x = 0; s1[ x ] != '\0'; ++x )
      ; // empty body

   for ( int y = 0; s1[ x ] = s2[ y ]; ++x, ++y )
      ; // empty body

   return s1;
}

char *stringCat2( char *s1, const char *s2 )
{
   char *ptr = s1;

   for ( ; *s1 != '\0'; ++s1 )
      ; // empty body

   for ( ; *s1 = *s2; ++s1, ++s2 )
      ; // empty body

   return ptr;
}

char *stringNCat1( char *s1, const char *s2, unsigned n )
{
   int x;

   for ( x = 0; s1[ x ] != '\0'; ++x )
      ; // empty body

   for ( unsigned y = 0; y < n && ( s1[ x ] = s2[ y ] ); ++x, ++y )
      ; // empty body

   s1[ x ] = '\0';
   return s1;
}

char *stringNCat2( char *s1, const char *s2, unsigned n )
{
   char *ptr = s1;

   for ( ; *s1 != '\0'; ++s1 )
      ; // empty body

   for ( unsigned c = 0 ; c < n && ( *s1 = *s2 ); ++s1, ++s2 )
      ; // empty body

   *s1 = '\0';
   return ptr;
}
```

```
Enter a string: coo
Copied string returned from stringCopy1 is coo
Copied string returned from stringCopy2 is coo
Copied 4 elements returned from stringNCopy1 is coo
Copied 4 elements returned from stringNCopy2 is coo
Concatenated string returned from stringCat1 is coocoo
Concatenated string returned from stringCat2 is coocoocoo
Concatenated string returned from stringNCat1 is coocoocoocoo
Concatenated string returned from stringNCat2 is coocoocoocoocoo
```

5.40 Write two versions of each string comparison function in Fig. 5.29. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

```
// Exercise 5.40 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int stringCompare1( const char *, const char * );
int stringCompare2( const char *, const char * );
int stringNCompare1( const char *, const char *, unsigned );
int stringNCompare2( const char *, const char *, unsigned );

int main()
{
   char string1[ 100 ], string2[ 100 ];
   unsigned n = 3;  // number of characters to be compared

   cout << "Enter two strings: ";
   cin >> string1 >> string2;

   cout << "The value returned from stringCompare1(\"" << string1
        << "\", \"" << string2 << "\") is "
        << stringCompare1(string1, string2)
        << "\nThe value returned from stringCompare2(\"" << string1
        << "\", \"" << string2 << "\") is "
        << stringCompare2(string1, string2) << '\n';

   cout << "\nThe value returned from stringNCompare1(\"" << string1
        << "\", \"" << string2 << "\", " << n << ") is "
        << stringNCompare1(string1, string2, n)
        << "\nThe value returned from stringNCompare2(\"" << string1
        << "\", \"" << string2 << "\", " << n << ") is "
        << stringNCompare2( string1, string2, n ) << endl;

   return 0;
}

int stringCompare1( const char *s1, const char *s2 )
{
   int sub;

   // array subscript notation
   for ( sub = 0; s1[ sub ] == s2[ sub ]; ++sub )
      ; // empty statement

   --sub;

   if ( s1[ sub ] == '\0' && s2[ sub ] == '\0')
      return 0;
   else if ( s1[ sub] < s2[ sub ] )
      return -1;
   else
      return 1;
}

int stringCompare2( const char *s1, const char *s2 )
{
   // pointer notation
   for ( ; *s1 == *s2; s1++, s2++ )
      ; // empty statement

   --s1;
   --s2;

   if ( *s1 == '\0' && *s2 == '\0' )
      return 0;
   else if ( *s1 < *s2 )
      return -1;
   else
      return 1;
}

int stringNCompare1( const char *s1, const char *s2, unsigned n )
{
   unsigned sub;

   // array subscript notation
   for ( sub = 0; sub < n && ( s1[ sub ] == s2[ sub ] ); sub++ )
      ; // empty body

   --sub;

   if ( s1[ sub ] == s2[ sub ] )
      return 0;
   else if ( s1[ sub ] < s2[ sub ] )
      return -1;
   else
      return 1;
}

int stringNCompare2( const char *s1, const char *s2, unsigned n )
{
   // pointer notation
   for ( unsigned c = 0; c < n && (*s1 == *s2); c++, s1++, s2++ )
      ; // empty statement

   --s1;
   --s2;

   if ( *s1 == *s2 )
      return 0;
   else if ( *s1 < *s2 )
      return -1;
   else
      return 1;
}
```

```
Enter two strings: tommy tomato
The value returned from stringCompare1("tommy", "tomato") is 1
The value returned from stringCompare2("tommy", "tomato") is 1

The value returned from stringNCompare1("tommy", "tomato", 3) is 0
The value returned from stringNCompare2("tommy", "tomato", 3) is 0
```

5.41 Write two versions of function `strlen` in Fig. 5.29. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

```
// Exercise 5.41 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

unsigned long stringLength1( const char * );
unsigned long stringLength2( const char * );

int main()
{
   char string[ 100 ];

   cout << "Enter a string: ";
   cin >> string;

   cout << "\nAccording to stringLength1 the string length is: "
           << stringLength1( string )
           << "\nAccording to stringLength2 the string length is: "
           << stringLength2( string ) << endl;

   return 0;
}

unsigned long stringLength1( const char *sPtr )
{
   // array subscript notation
   for ( int length = 0; sPtr[ length ] != '\0'; ++length )
      ; // empty body

   return length;
}

unsigned long stringLength2( const char *sPtr )
{
   // pointer notation
   for ( int length = 0; *sPtr != '\0'; ++sPtr, ++length )
      ; // empty body

   return length;
}
```

```
Enter a string: howLongCanThisNameWithoutQuestionPossiblyBe?

According to stringLength1 the string length is: 44
According to stringLength2 the string length is: 44
```

SPECIAL SECTION: ADVANCED STRING MANIPULATION EXERCISES

The preceding exercises are keyed to the text and designed to test the reader's understanding of fundamental string manipulation concepts. This section includes a collection of intermediate and advanced string manipulation exercises. The reader should find these problems challenging, yet enjoyable. The problems vary considerably in difficulty. Some require an hour or two of program writing and implementation. Others are useful for lab assignments that might require two or three weeks of study and implementation. Some are challenging term projects.

5.42 *(Text Analysis)* The availability of computers with string manipulation capabilities has resulted in some rather interesting approaches to analyzing the writings of great authors. Much attention has been focused on whether William Shakespeare ever lived Some scholars believe there is substantial evidence indicating that Christopher Marlowe or other authors actually penned the masterpieces attributed to Shakespeare. Researchers have used computers to find similarities in the writings of these two authors. This exercise examines three methods for analyzing texts with a computer.

a) Write a program that reads several lines of text from the keyboard and prints a table indicating the number of occurrences of each letter of the alphabet in the text. For example, the phrase

```
To be, or not to be: that is the question:
```

contains one “a,” two “b’s,” no “c’s,” etc.

b) Write a program that reads several lines of text and prints a table indicating the number of one-letter words, two-letter words, three-letter words, etc., appearing in the text. For example, the phrase

```
Whether 'tis nobler in the mind to suffer
```

contains

Word length	Occurrences
1	0
2	2
3	1
4	2 (including 'tis)
5	0
6	2
7	1

c) Write a program that reads several lines of text and prints a table indicating the number of occurrences of each different word in the text. The first version of your program should include the words in the table in the same order in which they appear in the text. For example, the lines

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
```

contain the words “to” three times, the word “be” two times, the word “or” once, etc. A more interesting (and useful) printout should then be attempted in which the words are sorted alphabetically.

```
// Exercise 5.42 Part A Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;

#include <cctype>

const int SIZE = 80;

int main()
{
   char letters[ 26 ] = { 0 }, text[ 3 ][ SIZE ], i;

   cout << "Enter three lines of text:\n";

   for ( i = 0; i <= 2; ++i )
      cin.getline( &text[ i ][ 0 ], SIZE );

   for ( i = 0; i <= 2; ++i )
      for ( int j = 0; text[ i ][ j ] != '\0'; ++j )
         if ( isalpha( text[ i ][ j ] ) )
            ++letters[ tolower( text[ i ][ j ] ) - 'a' ];

   cout << "\nTotal letter counts:\n";

   for ( i = 0; i <= 25; ++i )
      cout << setw( 3 ) << static_cast< char > ( 'a' + i ) << ':' << setw( 3 )
           << static_cast< int > ( letters[ i ] ) << endl;

   return 0;
}
```

```
Enter three lines of text:
when the cats are away the mice will play
still waters run deep
out of sight out of mind

Total letter counts:
  a:  6
  b:  0
  c:  2
  d:  2
  e:  8
  f:  2
  g:  1
  h:  4
  i:  5
  j:  0
  k:  0
  l:  5
  m:  2
  n:  3
  o:  4
  p:  2
  q:  0
  r:  3
  s:  4
  t:  8
  u:  3
  v:  0
  w:  4
  x:  0
  y:  2
  z:  0
```

```cpp
// Exercise 5.42 Part B Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

int main()
{
   const int SIZE = 80;
   char text[ 3 ][ SIZE ], *temp;
   int lengths[ 20 ] = { 0 }, i;

   cout << "Enter three lines of text:\n";

   for ( i = 0; i <= 2; ++i )
      cin.getline( &text[ i ][ 0 ], SIZE );

   for ( i = 0; i <= 2; ++i ) {
      temp = strtok( &text[ i ][ 0 ], ". \n" );

      while ( temp ) {
         ++lengths[ strlen( temp ) ];
         temp = strtok( 0, ". \n" );
      }
   }

   cout << '\n';

   for ( i = 1; i <= 19; ++i )
      if ( lengths[ i ] )
         cout << lengths[ i ] << " word(s) of length " << i << endl;

   return 0;
}
```

```
Enter three lines of text:
the five time super bowl champion dallas cowboys
americas team the dallas cowboys
the once and future champion dallas cowboys

4 word(s) of length 3
5 word(s) of length 4
1 word(s) of length 5
4 word(s) of length 6
3 word(s) of length 7
3 word(s) of length 8
```

```
// Exercise 5.42 Part C Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>

const int SIZE = 80;

int main()
{
   char text[ 3 ][ SIZE ], *temp, words[ 100 ][ 20 ] = { "" };
   int count[ 100 ] = { 0 }, i;

   cout << "Enter three lines of text:\n";

   for ( i = 0; i <= 2; ++i )
      cin.getline( &text[ i ][ 0 ], SIZE );

   for ( i = 0; i <= 2; ++i ) {
      temp = strtok( &text[ i ][ 0 ], ". \n" );

      while ( temp ) {
         int j;

         for ( j = 0; words[ j ][ 0 ] &&
                  strcmp( temp, &words[ j ][ 0 ] ) != 0; ++j )
            ;  // empty body

         ++count[ j ];

         if ( !words[ j ][ 0 ] )
            strcpy( &words[ j ][ 0 ], temp );

         temp = strtok( 0, ". \n" );
      }
   }

   cout << '\n';

   for ( int k = 0; words[ k ][ 0 ] != '\0' && k <= 99; ++k )
      cout << "\"" << &words[ k ][ 0 ] << "\" appeared " << count[ k ]
           << " time(s)\n";

   cout << endl;

   return 0;
}
```

```
Enter three lines of text:
lovebirds make great pets
the dallas cowboys are a great organization
peach faced lovebirds are small birds

"lovebirds" appeared 2 time(s)
"make" appeared 1 time(s)
"great" appeared 2 time(s)
"pets" appeared 1 time(s)
"the" appeared 1 time(s)
"dallas" appeared 1 time(s)
"cowboys" appeared 1 time(s)
"are" appeared 2 time(s)
"a" appeared 1 time(s)
"organization" appeared 1 time(s)
"peach" appeared 1 time(s)
"faced" appeared 1 time(s)
"small" appeared 1 time(s)
"birds" appeared 1 time(s)
```

5.43 *(Word Processing)* One important function in word-processing systems is *type justification*—the alignment of words to both the left and right margins of a page. This generates a professional-looking document that gives the appearance of being set in type rather than prepared on a typewriter. Type justification can be accomplished on computer systems by inserting blank characters between each of the words in a line so that the rightmost word aligns with the right margin.

Write a program that reads several lines of text and prints this text in type-justified format. Assume that the text is to be printed on 8-1/2-inch-wide paper, and that one-inch margins are to be allowed on both the left and right sides of the printed page. Assume that the computer prints 10 characters to the horizontal inch. Therefore, your program should print 6 1/2 inches of text, or 65 characters per line.

5.44 *(Printing Dates in Various Formats)* Dates are commonly printed in several different formats in business correspondence. Two of the more common formats are

07/21/1955 and **July 21, 1955**

Write a program that reads a date in the first format and prints that date in the second format.

```
// Exercise 5.44 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   const char *months[ 13 ] = { "", "January", "February", "March", "April",
                                "May", "June", "July", "August", "September",
                                "October", "November", "December" };
   int m, d, y;

   cout << "Enter a date in the form mm/dd/yy: \n";
   cin >> m;
   cin.ignore();
   cin >> d;
   cin.ignore();
   cin >> y;

   cout << "The date is: " << months[ m ] << ' ' << d << ' '
        << ( ( y < 50 ) ? y + 2000 : y + 1900 ) << endl;

   return 0;
}
```

```
Enter a date in the form mm/dd/yy:
6/7/00
The date is: June 7 2000
```

5.45 *(Check Protection)* Computers are frequently employed in check-writing systems such as payroll and accounts payable applications. Many strange stories circulate regarding weekly paychecks being printed (by mistake) for amounts in excess of $1 million. Weird amounts are printed by computerized check-writing systems, because of human error or machine failure. Systems designers build controls into their systems to prevent such erroneous checks from being issued.

Another serious problem is the intentional alteration of a check amount by someone who intends to cash a check fraudulently. To prevent a dollar amount from being altered, most computerized check-writing systems employ a technique called *check protection.*

Checks designed for imprinting by computer contain a fixed number of spaces in which the computer may print an amount. Suppose a paycheck contains eight blank spaces in which the computer is supposed to print the amount of a weekly paycheck. If the amount is large, then all eight of those spaces will be filled, for example,

```
1,230.60    (check amount)
--------
12345678    (position numbers)
```

On the other hand, if the amount is less than $1000, then several of the spaces would ordinarily be left blank. For example,

```
   99.87
--------
12345678
```

contains three blank spaces. If a check is printed with blank spaces, it is easier for someone to alter the amount of the check. To prevent a check from being altered, many check-writing systems insert *leading asterisks* to protect the amount as follows:

```
***99.87
--------
12345678
```

Write a program that inputs a dollar amount to be printed on a check and then prints the amount in check-protected format with leading asterisks if necessary. Assume that nine spaces are available for printing an amount.

```
// Exercise 5.45 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setfill;
using std::setw;
using std::setprecision;
using std::setiosflags;

int main()
{
   double amount, base = 100000.0;
   int i;

   cout << "Enter check amount: ";
   cin >> amount;
   cout << "The protected amount is $";

   for ( i = 0; amount < base; ++i )
      base /= 10;

   for ( int j = 1; j <= i; ++j )
      cout << '*';

   cout << setiosflags( ios::fixed | ios::showpoint )
        << setw( 9 - i ) << setfill( '*' )
        << setprecision( 2 ) << amount << endl;

   return 0;
}
```

```
Enter check amount: 22.88
The protected amount is $****22.88
```

5.46 *(Writing the Word Equivalent of a Check Amount)* Continuing the discussion of the previous example, we reiterate the importance of designing check-writing systems to prevent alteration of check amounts. One common security method requires that the check amount be written both in numbers, and "spelled out" in words as well. Even if someone is able to alter the numerical amount of the check, it is extremely difficult to change the amount in words.

Many computerized check-writing systems do not print the amount of the check in words. Perhaps the main reason for this omission is the fact that most high-level languages used in commercial applications do not contain adequate string manipulation features. Another reason is that the logic for writing word equivalents of check amounts is somewhat involved.

Write a program that inputs a numeric check amount and writes the word equivalent of the amount. For example, the amount 112.43 should be written as

```
ONE HUNDRED TWELVE and 43/100
```

```
// Exercise 5.46 Solution
// NOTE: THIS PROGRAM ONLY HANDLES VALUES UP TO $99.99
// The program is easily modified to process larger values
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

int main()
{
   const char *digits[ 10 ] = { "", "ONE", "TWO", "THREE", "FOUR", "FIVE",
                                "SIX", "SEVEN", "EIGHT", "NINE" };
   const char *teens[ 10 ] = { "TEN", "ELEVEN", "TWELVE", "THIRTEEN",
                               "FOURTEEN", "FIFTEEN", "SIXTEEN",
                               "SEVENTEEN", "EIGHTEEN", "NINETEEN"};
   const char *tens[ 10 ] = { "", "TEN", "TWENTY", "THIRTY", "FORTY", "FIFTY",
                              "SIXTY", "SEVENTY", "EIGHTY", "NINETY" };
   int dollars, cents, digit1, digit2;

   cout << "Enter the check amount (0.00 to 99.99): ";
   cin >> dollars;
   cin.ignore();
   cin >> cents;
   cout << "The check amount in words is:\n";

   if ( dollars < 10 )
      cout << digits[ dollars ] << ' ';
   else if ( dollars < 20 )
      cout << teens[ dollars - 10 ] << ' ';
   else {
      digit1 = dollars / 10;
      digit2 = dollars % 10;

      if ( !digit2 )
         cout << tens[ digit1 ] << ' ';
      else
         cout << tens[ digit1 ] << "-" << digits[ digit2 ] << ' ';
   }

   cout << "Dollars and " << cents << "/100" << endl;

   return 0;
}
```

```
Enter the check amount (0.00 to 99.99): 72.68
The check amount in words is:
SEVENTY-TWO Dollars and 68/100
```

5.47 *(Morse Code)* Perhaps the most famous of all coding schemes is the Morse code, developed by Samuel Morse in 1832 for use with the telegraph system. The Morse code assigns a series of dots and dashes to each letter of the alphabet, each digit and a few special characters (such as period, comma, colon and semicolon). In sound-oriented systems, the dot represents a short sound and the dash represents a long sound. Other representations of dots and dashes are used with light-oriented systems and signal-flag systems.

Separation between words is indicated by a space, or, quite simply, the absence of a dot or dash. In a sound-oriented system, a space is indicated by a short period of time during which no sound is transmitted. The international version of the Morse code appears in Fig. 5.39.

Write a program that reads an English-language phrase and encodes the phrase into Morse code. Also write a program that reads a phrase in Morse code and converts the phrase into the English-language equivalent. Use one blank between each Morse-coded letter and three blanks between each Morse-coded word.

Character	Code	Character	Code
A	`.-`	T	`-`
B	`-...`	U	`..-`
C	`-.-.`	V	`...-`
D	`-..`	W	`.--`
E	`.`	X	`-..-`
F	`..-.`	Y	`-.--`
G	`--.`	Z	`--..`
H	`....`		
I	`..`	Digits	
J	`.---`	1	`.----`
K	`-.-`	2	`..---`
L	`.-..`	3	`...--`
M	`--`	4	`....-`
N	`-.`	5	`.....`
O	`---`	6	`-....`
P	`.--.`	7	`--...`
Q	`--.-`	8	`---..`
R	`.-.`	9	`----.`
S	`...`	0	`-----`

Fig. 5.39 The letters of the alphabet as expressed in international Morse code.

5.48 *(A Metric Conversion Program)* Write a program that will assist the user with metric conversions. Your program should allow the user to specify the names of the units as strings (i.e., centimeters, liters, grams, etc. for the metric system and inches, quarts, pounds, etc., for the English system) and should respond to simple questions such as

```
"How many inches are in 2 meters?"
"How many liters are in 10 quarts?"
```

Your program should recognize invalid conversions. For example, the question

```
"How many feet in 5 kilograms?"
```

is not meaningful, because **`"feet"`** are units of length, while **`"kilograms"`** are units of weight.

A CHALLENGING STRING MANIPULATION PROJECT

5.49 *(A Crossword Puzzle Generator)* Most people have worked a crossword puzzle, but few have ever attempted to generate one. Generating a crossword puzzle is a difficult problem. It is suggested here as a string manipulation project requiring substantial sophistication and effort. There are many issues the programmer must resolve to get even the simplest crossword puzzle generator program working. For example, how does one represent the grid of a crossword puzzle inside the computer? Should one use a series of strings, or should double-subscripted arrays be used? The programmer needs a source of words (i.e., a computerized dictionary) that can be directly referenced by the program. In what form should these words be stored to facilitate the complex manipulations required by the program? The really ambitious reader will want to generate the "clues" portion of the puzzle in which the brief hints for each "across" word and each "down" word are printed for the puzzle worker. Merely printing a version of the blank puzzle itself is not a simple problem.

6

Classes and Data Abstraction Solutions

Solutions

6.3 What is the purpose of the scope resolution operator?

ANS: The scope resolution operator is used to specify the class to which a function belongs. It also resolves the ambiguity caused by multiple classes having member functions of the same name.

6.4 Compare and contrast the notions of `struct` and `class` in C++.

ANS: In C++, the keywords `struct` and `class` are used to define types containing data members (and member functions for `class`es). The differences occur in the default access privileges for each. The default access for `class` members is `private` and the default access for `struct` members is `public`. The access privileges for `class` and `struct` can be specified explicitly.

6.5 Provide a constructor that is capable of using the current time from the `time()` function—declared in the C Standard Library header `ctime`—to initialize an object of the `Time` class.

```
// P6_05.H
#ifndef p6_05_H
#define p6_05_H
#include <iostream>

class Time {
public:
   Time();
   void setHour( int h ) { hour = ( h >= 0 && h < 24 ) ? h : 0; }
   void setMinute( int m ) { minute = ( m >= 0 && m < 60 ) ? m : 0; }
   void setSecond( int s ) { second = ( s >= 0 && s < 60 ) ? s : 0; }
   int getHour( void ) { return hour; }
   int getMinute( void ) { return minute; }
   int getSecond( void ) { return second; }
   void printStandard( void );
private:
   int hour;
   int minute;
   int second;
};

#endif
```

```cpp
// P6_5M.cpp
// member function definitions for p6_05.cpp
#include <iostream>

using std::cout;

#include <ctime>

#include "p6_05.h"

Time::Time()
{
   long int totalTime;             // time in seconds since 1970
   int currentYear = 1998 - 1970; // current year
   double totalYear;               // current time in years
   double totalDay;                // days since beginning of year
   double day;                     // current time in days
   double divisor;                 // conversion divisor
   int timeShift = 7;              // time returned by time() is
                                   // given as the number of seconds
                                   // elapsed since 1/1/70 GMT.
                                   // Depending on the time zone
                                   // you are in, you must shift
                                   // the time by a certain
                                   // number of hours. For this
                                   // problem, 7 hours is the
                                   // current shift for EST.
   double tempMinute;              // Used in conversion to seconds.
   double tempSecond;              // Used to set seconds.

   totalTime = time( 0 );
   divisor = ( 60.0 * 60.0 * 24.0 * 365.0 );
   totalYear = totalTime / divisor - currentYear;
   totalDay = 365 * totalYear;    // leap years ignored
   day = totalDay - static_cast< int >( totalDay );
   tempMinute = totalDay * 24 * 60;
   setHour( day * 24 + timeShift );
   setMinute( ( day * 24 - static_cast< int >( day * 24 ) ) * 60 );
   tempMinute -= static_cast< int > ( tempMinute ) * 60;
   tempSecond = tempMinute;
   setSecond( tempSecond );
}

void Time::printStandard()
{
   cout << ( ( hour % 12 == 0 ) ? 12 : hour % 12 ) << ':'
        << ( minute < 10 ? "0" : "" ) << minute << ':'
        << ( second < 10 ? "0" : "" ) << second << '\n';
}
```

```cpp
// driver for p6_05.cpp
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;
#include "p6_05.h"

int main()
{
   Time t;

   t.printStandard();

   return 0;
}
```

```
11:26:00
```

6.6 Create a class called **Complex** for performing arithmetic with complex numbers. Write a driver program to test your class. Complex numbers have the form

```
realPart + imaginaryPart * i
```

where ***i*** is

$$\sqrt{-1}$$

Use **double** variables to represent the **private** data of the class. Provide a constructor function that enables an object of this class to be initialized when it is declared. The constructor should contain default values in case no initializers are provided.Provide **public** member functions for each of the following:

a) Addition of two **Complex** numbers: The real parts are added together and the imaginary parts are added together.
b) Subtraction of two **Complex** numbers: The real part of the right operand is subtracted from the real part of the left operand and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.
c) Printing **Complex** numbers in the form **(a, b)** where **a** is the real part and **b** is the imaginary part.

```
// P6_06.H
#ifndef p6_06_H
#define p6_06_H

class Complex {
public:
   Complex( double = 0.0, double = 0.0 ); // default constructor
   void addition( const Complex & );
   void subtraction( const Complex & );
   void printComplex( void );
   void setComplexNumber( double, double );
private:
   double realPart;
   double imaginaryPart;
};

#endif
```

```
// p6_06M.cpp
// member function definitions for p6_06.cpp
#include <iostream>

using std::cout;

#include "p6_06.h"

Complex::Complex( double real, double imaginary )
  { setComplexNumber( real, imaginary ); }

void Complex::addition( const Complex &a )
{
   realPart += a.realPart;
   imaginaryPart += a.imaginaryPart;
}

void Complex::subtraction( const Complex &s )
{
   realPart -= s.realPart;
   imaginaryPart -= s.imaginaryPart;
}

void Complex::printComplex( void )
  { cout << '(' << realPart << ", " << imaginaryPart << ')'; }

void Complex::setComplexNumber( double rp, double ip )
{
   realPart = rp;
   imaginaryPart = ip;
}
```

```
// driver for p6_06.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_06.h"

int main()
{
   Complex b( 1, 7 ), c( 9, 2 );

   b.printComplex();
   cout << " + ";
   c.printComplex();
   cout << " = ";
   b.addition( c );
   b.printComplex();

   cout << '\n';
   b.setComplexNumber( 10, 1 ); // reset realPart and imaginaryPart
   c.setComplexNumber( 11, 5 );
   b.printComplex();
   cout << " - ";
   c.printComplex();
   cout << " = ";
   b.subtraction( c );
   b.printComplex();
   cout << endl;

   return 0;
}
```

```
(1, 7) + (9, 2) = (10, 9)
(10, 1) - (11, 5) = (-1, -4)
```

6.7 Create a class called **`Rational`** for performing arithmetic with fractions. Write a driver program to test your class.

Use integer variables to represent the **`private`** data of the class—the numerator and the denominator. Provide a constructor function that enables an object of this class to be initialized when it is declared. The constructor should contain default valu es in case no initializers are provided and should store the fraction in reduced form (i.e., the fraction

$$\frac{2}{4}$$

would be stored in the object as 1 in the numerator and 2 in the denominator). Provide **public** member functions for each of the following:

a) Addition of two **Rational** numbers. The result should be stored in reduced form.
b) Subtraction of two **Rational** numbers. The result should be stored in reduced form.
c) Multiplication of two **Rational** numbers. The result should be stored in reduced form.
d) Division of two **Rational** numbers. The result should be stored in reduced form.
e) Printing **Rational** numbers in the form **a/b** where **a** is the numerator and **b** is the denominator.
f) Printing **Rational** numbers in double floating-point format.

```
// P6_07.H
#ifndef P6_07_H
#define P6_07_H

class Rational {
public:
   Rational( int = 0, int = 1 );  // default constructor
   Rational addition( const Rational & );
   Rational subtraction( const Rational & );
   Rational multiplication( const Rational & );
   Rational division( Rational & );
   void printRational( void );
   void printRationalAsDouble( void );
private:
   int numerator;
   int denominator;
   void reduction( void );   // utility function
};

#endif
```

```
// P6_07M.cpp
// member function definitions for p6_07.cpp
#include <iostream>

using std::cout;

#include "p6_07.h"

Rational::Rational( int n, int d )
{
   numerator = n;
   denominator = d;
}

Rational Rational::addition( const Rational &a )
{
   Rational t;

   t.numerator = a.numerator * denominator;
   t.numerator += a.denominator * numerator;
   t.denominator = a.denominator * denominator;
   t.reduction();

   return t;
}

Rational Rational::subtraction( const Rational &s )
```

```
{
   Rational t;

   t.numerator = s.denominator * numerator;
   t.numerator -= denominator * s.numerator;
   t.denominator = s.denominator * denominator;
   t.reduction();

   return t;
}

Rational Rational::multiplication( const Rational &m )
{
   Rational t;

   t.numerator = m.numerator * numerator;
   t.denominator = m.denominator * denominator;
   t.reduction();

   return t;
}

Rational Rational::division( Rational &v )
{
   Rational t;

   t.numerator = v.denominator * numerator;
   t.denominator = denominator * v.numerator;
   t.reduction();

   return t;
}

void Rational::printRational( void )
{
   if ( denominator == 0 )
      cout << "\nDIVIDE BY ZERO ERROR!!!" << '\n';
   else if ( numerator == 0 )
      cout << 0;
   else
      cout << numerator << '/' << denominator;
}

void Rational::printRationalAsDouble( void )
   {  cout << static_cast< double >( numerator ) / denominator; }

void Rational::reduction( void )
{
   int largest;
   largest = numerator > denominator ? numerator : denominator;

   int gcd = 0;  // greatest common divisor

   for ( int loop = 2; loop <= largest; ++loop )
       if ( numerator % loop == 0 && denominator % loop == 0 )
          gcd = loop;

   if (gcd != 0) {
      numerator /= gcd;
      denominator /= gcd;
   }
}
```

```
// driver for P6_07.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_07.h"

int main()
{
   Rational c( 1, 3 ), d( 7, 8 ), x;

   c.printRational();
   cout << " + ";
   d.printRational();
   x = c.addition( d );
   cout << " = ";
   x.printRational();
   cout << '\n';
   x.printRational();
   cout << " = ";
   x.printRationalAsDouble();
   cout << "\n\n";

   c.printRational();
   cout << " - ";
   d.printRational();
   x = c.subtraction( d );
   cout << " = ";
   x.printRational();
   cout << '\n';
   x.printRational();
   cout << " = ";
   x.printRationalAsDouble();
   cout << "\n\n";

   c.printRational();
   cout << " x ";
   d.printRational();
   x = c.multiplication( d );
   cout << " = ";
   x.printRational();
   cout << '\n';
   x.printRational();
   cout << " = ";
   x.printRationalAsDouble();
   cout << "\n\n";

   c.printRational();
   cout << " / ";
   d.printRational();
   x = c.division( d );
   cout << " = ";
   x.printRational();
   cout << '\n';
   x.printRational();
   cout << " = ";
   x.printRationalAsDouble();
   cout << endl;

   return 0;
}
```

```
1/3 + 7/8 = 29/24
29/24 = 1.20833

1/3 - 7/8 = -13/24
-13/24 = -0.541667

1/3 x 7/8 = 7/24
7/24 = 0.291667

1/3 / 7/8 = 8/21
8/21 = 0.380952
```

6.8 Modify the **Time** class of Fig. 6.10 to include a **tick** member function that increments the time stored in a **Time** object by one second. The **Time** object should always remain in a consistent state. Write a driver program that tests the **tick** member function in a loop that prints the time in standard format during each iteration of the loop to illustrate that the **tick** member function works correctly. Be sure to test the following cases:

a) Incrementing into the next minute.
b) Incrementing into the next hour.
c) Incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

```
// P6_08.H
#ifndef p6_08_H
#define p6_08_H

class Time {
public:
   Time( int = 0, int = 0, int = 0 );
   void setTime( int, int, int );
   void setHour( int );
   void setMinute( int );
   void setSecond( int );
   int getHour( void );
   int getMinute( void );
   int getSecond( void );
   void printStandard( void );
   void tick( void );
private:
   int hour;
   int minute;
   int second;
};

#endif
```

```
// P6_08M.cpp
// member function definitions for p6_08.cpp
#include <iostream>

using std::cout;

#include "p6_08.h"

Time::Time( int hr, int min, int sec ) { setTime( hr, min, sec ); }

void Time::setTime( int h, int m, int s )
{
   setHour( h );
   setMinute( m );
```

```cpp
   setSecond( s );
}

void Time::setHour( int h ) { hour = ( h >= 0 && h < 24 ) ? h : 0; }

void Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0;
}

void Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

int Time::getHour( void ) { return hour; }

int Time::getMinute( void ) { return minute; }

int Time::getSecond( void ) { return second; }

void Time::printStandard( void )
{
   cout << ( ( hour % 12 == 0 ) ? 12 : hour % 12 ) << ':'
         << ( minute < 10 ? "0" : "" ) << minute << ':'
         << ( second < 10 ? "0" : "" ) << second
         << ( hour < 12 ? " AM" : " PM" );
}

void Time::tick( void )
{
   setSecond( getSecond() + 1 );

   if ( getSecond() == 0 ) {
      setMinute( getMinute() + 1 );

      if ( getMinute() == 0 )
         setHour( getHour() + 1 );
   }
}
```

```cpp
// driver for p6_08.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_08.h"

const int MAX_TICKS = 3000;

main()
{
   Time t;

   t.setTime( 23, 59, 57 );

   for ( int ticks = 1; ticks < MAX_TICKS; ++ticks ) {
      t.printStandard();
      cout << endl;
      t.tick();
   }

   return 0;
}
```

```
11:59:57 PM
.
.
.
.
12:49:50 AM
12:49:51 AM
12:49:52 AM
12:49:53 AM
12:49:54 AM
12:49:55 AM
```

6.9 Modify the **Date** class of Fig. 6.12 to perform error checking on the initializer values for data members **month**, **day** and **year**. Also, provide a member function **nextDay** to increment the day by one. The **Date** object should always remain in a consistent state. Write a driver program that tests the **nextDay** function in a loop that prints the date during each iteration of the loop to illustrate that the **nextDay** function works correctly. Be sure to test the following cases:

a) Incrementing into the next month.
b) Incrementing into the next year.

```
// P6_09.H
#ifndef p6_09_H
#define p6_09_H

class Date {
public:
   Date( int = 1, int = 1, int = 1900 );  // default constructor
   void print( void );
   void setDate( int, int, int );
   void setMonth( int );
   void setDay( int );
   void setYear( int );
   int getMonth( void );
   int getDay( void );
   int getYear( void );
   bool leapYear( void );
   int monthDays( void );
   void nextDay( void );
private:
   int month;
   int day;
   int year;
};

#endif
```

```
// p6_09M.cpp
// member function definitions for p6_09.cpp
#include <iostream>

using std::cout;

#include "p6_09.h"

```

```
Date::Date( int m, int d, int y ) { setDate( m, d, y ); }

int Date::getDay() { return day; }

int Date::getMonth() { return month; }

int Date::getYear() { return year; }

void Date::setDay( int d )
{
   if ( month == 2 && leapYear() )
      day = ( d <= 29 && d >= 1 ) ? d : 1;
   else
      day = ( d <= monthDays() && d >= 1 ) ? d : 1;
}

void Date::setMonth( int m ) { month = m <= 12 && m >= 1 ? m : 1; }

void Date::setYear( int y )
{
   year = y <= 2000 && y >= 1900 ? y : 1900;
}

void Date::setDate( int mo, int dy, int yr )
{
   setMonth( mo );
   setDay( dy );
   setYear( yr );
}

void Date::print()
   { cout << month << '-' << day << '-' << year << '\n'; }

void Date::nextDay()
{
   setDay( day + 1 );

   if ( day == 1 ) {
      setMonth( month + 1 );

      if ( month == 1 )
         setYear( year + 1 );
   }
}

bool Date::leapYear( void )
{
   if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
         return true;
      else
         return false;    // not a leap year
}

int Date::monthDays( void )
{
   const int days[ 12 ] = { 31, 28, 31, 30, 31, 30,
                            31, 31, 30, 31, 30, 31 };

   return month == 2 && leapYear() ? 29 : days[ month - 1 ];
}
```

```
// driver for p6_09.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_09.h"

int main()
{
   const int MAXDAYS = 160;
   Date d( 9, 2, 1998 );

   for ( int loop = 1; loop <= MAXDAYS; ++loop ) {
      d.print();
      d.nextDay();
   }

   cout << endl;

   return 0;
}
```

```
9-2-1998
9-3-1998
.
.
.
2-5-1999
2-6-1999
2-7-1999
2-8-1999
```

6.10 Combine the modified **Time** class of Exercise 6.8 and the modified **Date** class of Exercise 6.9 into one class called **DateAndTime** (in Chapter 9 we will discuss inheritance, which will enable us to accomplish this task quickly without modifying the existing class definitions). Modify the **tick**s function to call the **nextDay** function if the time is incremented into the next day. Modify function **printStandard** and **printMilitary** to output the date in addition to the time. Write a driver program to test the new class **DateAndTime**. Specifically, test incrementing the time into the next day.

```
// P6_10.H
#ifndef p6_10_H
#define p6_10_H

class DateAndTime {
public:
   DateAndTime( int = 1, int = 1, int = 1900,
                int = 0, int = 0, int = 0 );
   void setDate( int, int, int );
   void setMonth( int );
   void setDay( int );
   void setYear( int );
   int getMonth( void );
   int getDay( void );
   int getYear( void );
   void nextDay( void );
   void setTime( int, int, int );
   void setHour( int );
   void setMinute( int );
   void setSecond( int );
   int getHour( void );
   int getMinute( void );
   int getSecond( void );
   void printStandard( void );
   void printMilitary( void );
   int monthDays( void );
   void tick( void );
   bool leapYear( void );
private:
   int month;
   int day;
   int year;
   int hour;
   int minute;
   int second;
};

#endif
```

```
// P6_10M.cpp
// member function definitions for p6_10.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_10.h"

DateAndTime::DateAndTime( int m, int d, int y, int hr,
                          int min, int sec )
{
   setDate( m, d, y );
   setTime( hr, min, sec );
}

void DateAndTime::setDate( int mo, int dy, int yr )
{
   setMonth( mo );
   setDay( dy );
   setYear( yr );
}

int DateAndTime::getDay( void ) { return day; }

int DateAndTime::getMonth( void ) { return month; }

int DateAndTime::getYear( void ) { return year; }

void DateAndTime::setDay( int d )
{
   if ( month == 2 && leapYear() )
      day = ( d <= 29 && d >= 1 ) ? d : 1;
   else
      day = ( d <= monthDays() && d >= 1 ) ? d : 1;
}

void DateAndTime::setMonth( int m )
   { month = m <= 12 && m >= 1 ? m : 1; }

void DateAndTime::setYear( int y )
   { year = y <= 2000 && y >= 1900 ? y : 1900; }
```

```

void DateAndTime::nextDay( void )
{
   setDay( day + 1 );

   if ( day == 1 ) {
      setMonth( month + 1 );

      if ( month == 1 )
         setYear( year + 1 );
   }
}

void DateAndTime::setTime( int hr, int min, int sec )
{
   setHour( hr );
   setMinute( min );
   setSecond( sec );
}

void DateAndTime::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
}

void DateAndTime::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0;
}

void DateAndTime::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

int DateAndTime::getHour( void ) { return hour; }

int DateAndTime::getMinute( void ) { return minute; }

int DateAndTime::getSecond( void ) { return second; }

void DateAndTime::printStandard( void )
{
   cout << ( ( hour % 12 == 0 ) ? 12 : hour % 12 ) << ':'
           << ( minute < 10 ? "0" : "" ) << minute << ':'
           << ( second < 10 ? "0" : "" ) << second
           << ( hour < 12 ? " AM " : " PM " )
           << month << '-' << day << '-' << year << endl;
}

void DateAndTime::printMilitary( void )
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ':'
           << ( minute < 10 ? "0" : "" ) << minute << ':'
           << ( second < 10 ? "0" : "" ) << second << "     "
           << month << '-' << day << '-' << year << endl;
}

void DateAndTime::tick( void )
{
   setSecond( second + 1 );

   if ( second == 0 ) {
      setMinute( minute + 1 );

      if ( minute == 0 ) {
         setHour( hour + 1 );

         if ( hour == 0 )
            nextDay();
      }
   }
}

bool DateAndTime::leapYear( void )
{
   if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
         return true;
      else
         return false;    // not a leap year
}

int DateAndTime::monthDays( void )
{
   const int days[ 12 ] = { 31, 28, 31, 30, 31, 30,
                            31, 31, 30, 31, 30, 31 };

   return ( month == 2 && leapYear() ) ? 29 : days[ ( month - 1 ) ];
}
```

```
// driver for p6_10.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_10.h"

int main()
{
   const int MAXTICKS = 3000;

   DateAndTime d( 3, 2, 1998, 23, 50, 0 );

   for ( int ticks = 1; ticks <= MAXTICKS; ++ticks ) {
      cout << "Military time: ";
      d.printMilitary();
      cout << "Standard time: ";
      d.printStandard();
      d.tick();
   }

   cout << endl;

   return 0;
}
```

```
Military time: 23:50:00    3-2-1998
Standard time: 11:50:00 PM 3-2-1998
.
.
.
Standard time: 12:39:56 AM 3-3-1998
Military time: 00:39:57    3-3-1998
Standard time: 12:39:57 AM 3-3-1998
Military time: 00:39:58    3-3-1998
Standard time: 12:39:58 AM 3-3-1998
Military time: 00:39:59    3-3-1998
Standard time: 12:39:59 AM 3-3-1998
```

6.11 Modify the *set* functions in the program of Fig. 6.10 to return appropriate error values if an attempt is made to *set* a data member of an object of class **Time** to an invalid value.

```
// P6_11.H
#ifndef P6_11_H
#define P6_11_H

class Time {
public:
   Time( int = 0, int = 0, int = 0 );
   void setTime( int, int, int );
   void setHour( int );
   void setMinute( int );
   void setSecond( int );
   void setInvalidTime( int t ) { invalidTime = t; }
   int getHour( void ) { return hour; }
   int getMinute( void ) { return minute; }
   int getSecond( void ) { return second; }
   int getInvalidTime( void ) { return invalidTime; }
   void printMilitary( void );
   void printStandard( void );
private:
   int hour;
   int minute;
   int second;
   int invalidTime;   // set if an invalid time is attempted
};

#endif
```

```
// P6_11M.cpp
// member function defintions for p6_11.cpp
#include <iostream>

using std::cout;

#include "p6_11.h"

Time::Time( int hr, int min, int sec )
   { setTime( hr, min, sec ); }

void Time::setTime( int h, int m, int s )
{
   setHour( h );
   setMinute( m );
   setSecond( s );
}

void Time::setHour( int hr )
{
   if ( hr >= 0 && hr < 24 ) {
      hour = hr;
      setInvalidTime( 1 );  // hour is valid
   }
   else {
      hour = 0;
      setInvalidTime( 0 );  // hour is invalid
   }
}

void Time::setMinute( int min )
{
   if ( min >= 0 && min < 60 ) {
      minute = min;
      setInvalidTime( 1 );  // minute is valid
   }
   else {
      minute = 0;
      setInvalidTime( 0 );  // minute is invalid
   }
}

void Time::setSecond( int sec )
{
   if ( sec >= 0 && sec < 60 ) {
      second = sec;
      setInvalidTime( 1 );  // second is valid
   }
   else {
      second = 0;
      setInvalidTime( 0 );  // second is invalid
   }
}

void Time::printMilitary( void )
{
   cout << ( hour < 10 ? "0" : "" ) << hour << ':'
        << ( minute < 10 ? "0" : "" ) << minute << ':'
        << ( second < 10 ? "0" : "" ) << second;
}

void Time::printStandard( void )
{
   cout << ( ( hour % 12 == 0 ) ? 12 : hour % 12 ) << ':'
        << ( minute < 10 ? "0": "" ) << minute << ':'
        << ( second < 10 ? "0": "" ) << second
        << ( hour < 12 ? " AM" : " PM" );
}
```

```
// driver for p6_11.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p6_11.h"

```

```
int main()
{
   Time t1( 17, 34, 25 ), t2( 99, 345, -897 );

   // all t1 object's times are valid
   if ( !t1.getInvalidTime() )
      cout << "Error: invalid time setting(s) attempted." << '\n'
           << "Invalid setting(s) changed to zero." << '\n';

   t1.printStandard();

   // object t2 has invalid time settings
   if ( !t2.getInvalidTime() )
      cout << "\nError: invalid time setting(s) attempted.\n"
           << "Invalid setting(s) changed to zero.\n";

   t2.printMilitary();
   cout << endl;

   return 0;
}
```

```
5:34:25 PM
Error: invalid time setting(s) attempted.
Invalid setting(s) changed to zero.
00:00:00
```

6.12 Create a class **`Rectangle`**. The class has attributes **`length`** and **`width`**, each of which defaults to 1. It has member functions that calculate the **`perimeter`** and the **`area`** of the rectangle. It has *set* and *get* functions for both **`length`** and **`width`**. The *set* functions should verify that **`length`** and **`width`** are each floating-point numbers larger than 0.0 and less than 20.0.

```
// P6_12.H
#ifndef P6_12_H
#define P6_12_H

class Rectangle {
public:
   Rectangle( double = 1.0, double = 1.0 );
   double perimeter( void );
   double area( void );
   void setWidth( double w );
   void setLength( double l );
   double getWidth( void );
   double getLength( void );
private:
   double length;
   double width;
};

#endif
```

```
// P6_12M.cpp
// member function definitions for p6_12.cpp
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;


#include "p6_12.h"

Rectangle::Rectangle( double w, double l )
{
   setWidth(w);
   setLength(l);
}

double Rectangle::perimeter( void )
{
   return 2 * ( width + length );
}

double Rectangle::area( void )
{
   return width * length;
}

void Rectangle::setWidth( double w )
{
   width = w > 0 && w < 20.0 ? w : 1.0;
}

void Rectangle::setLength( double l )
{
   length = l > 0 && l < 20.0 ? l : 1.0;
}

double Rectangle::getWidth( void ) { return width; }

double Rectangle::getLength( void ) { return length; }
```

```
// driver for p6_12.cpp

#include "p6_12.h"

int main()
{
   Rectangle a, b( 4.0, 5.0 ), c( 67.0, 888.0 );

   cout << setiosflags( ios::fixed | ios::showpoint );
   cout << setprecision( 1 );

   // output Rectangle a
   cout << "a: length = " << a.getLength()
           << "; width = " << a.getWidth()
           << "; perimeter = " << a.perimeter() << "; area = "
           << a.area() << '\n';

   // output Rectangle b
   cout << "b: length = " << b.getLength()
           << "; width = " << b.getWidth()
           << "; perimeter = " << b.perimeter() << "; area = "
           << b.area() << '\n';

   // output Rectangle c; bad values attempted
   cout << "c: length = " << c.getLength()
           << "; width = " << c.getWidth()
           << "; perimeter = " << c.perimeter() << "; area = "
           << c.area() << endl;

   return 0;
}
```

```
a: length = 1.0; width = 1.0; perimeter = 4.0; area = 1.0
b: length = 5.0; width = 4.0; perimeter = 18.0; area = 20.0
c: length = 1.0; width = 1.0; perimeter = 4.0; area = 1.0
```

6.13 Create a more sophisticated **Rectangle** class than the one you created in Exercise 6.12. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* function that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x* or *y* coordinate larger than 20.0. The *set* function also verifies that the supplied coordinates do, in fact, specify a rectangle. Member functions calculate the **length**, **width**, **perimeter** and **area**. The length is the larger of the two dimensions. Include a predicate function **square** that determines if the rectangle is a square.

```
// P6_13.H
#ifndef P6_13_H
#define P6_13_H

class Rectangle {
public:
   Rectangle( double *, double *, double *, double * );
   void setCoord( double *, double *, double *, double * );
   void perimeter( void );
   void area( void );
   void square( void );
private:
   double point1[ 2 ];
   double point2[ 2 ];
   double point3[ 2 ];
   double point4[ 2 ];
};

#endif
```

```
// P6_13M.cpp
// member function definitions for p6_13.cpp
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <cmath>

#include "p6_13.h"

Rectangle::Rectangle( double *a, double *b,
                      double *c, double *d )
   { setCoord( a, b, c, d ); }

void Rectangle::setCoord( double *p1, double *p2,
                          double *p3, double *p4 )
```

```
{
   // Arrangement of points
   // p4.........p3
   //  .         .
   //  .         .
   // p1.........p2

   const int x = 0, y = 1;  // added for clarity

   // validate all points
   point1[ x ] = ( p1[ x ] > 20.0 || p1[ x ] < 0.0 )? 0.0 : p1[ x ];
   point1[ y ] = ( p1[ y ] > 20.0 || p1[ y ] < 0.0 )? 0.0 : p1[ y ];
   point2[ x ] = ( p2[ x ] > 20.0 || p2[ x ] < 0.0 )? 0.0 : p2[ x ];
   point2[ y ] = ( p2[ y ] > 20.0 || p2[ y ] < 0.0 )? 0.0 : p2[ y ];
   point3[ x ] = ( p3[ x ] > 20.0 || p3[ x ] < 0.0 )? 0.0 : p3[ x ];
   point3[ y ] = ( p3[ y ] > 20.0 || p3[ y ] < 0.0 )? 0.0 : p3[ y ];
   point4[ x ] = ( p4[ x ] > 20.0 || p4[ x ] < 0.0 )? 0.0 : p4[ x ];
   point4[ y ] = ( p4[ y ] > 20.0 || p4[ y ] < 0.0 )? 0.0 : p4[ y ];

   // verify that points form a rectangle
   if ( p1[ y ] == p2[ y ] && p1[ x ] ==
        p4[ x ] && p2[ x ] == p3[ x ] && p3[ y ] == p4[ y ] )
   {

      perimeter();
      area();
      square();
   }
   else
      cout << "Coordinates do not form a rectangle!\n";
}

void Rectangle::perimeter( void )
{
   double l = fabs( point4[ 1 ] - point1[ 1 ] ),
          w = fabs( point2[ 0 ] - point1[ 0 ] );

   cout << setiosflags( ios::fixed | ios::showpoint )
           << "length = " << setprecision( 1 ) << ( l > w ? l : w )
           << '\t' << "width = " << ( l > w ? w : l )
           << "\nThe perimeter is: " << 2 * ( w + l ) << '\n'
           << resetiosflags( ios::fixed | ios::showpoint );
}

void Rectangle::area( void )
{
   double l = fabs( point4[ 1 ] - point1[ 1 ] ),
          w = fabs( point2[ 0 ] - point1[ 0 ] );

   cout << setiosflags( ios::fixed | ios::showpoint )
           << "The area is: " << setprecision( 1 ) << w * l
           << resetiosflags( ios::fixed | ios::showpoint )
           << "\n\n" ;
}

void Rectangle::square( void )
{
   const int x = 0, y = 1;   // added for clarity

   if ( fabs( point4[ y ] - point1[ y ] ) ==
        fabs( point2[ x ] - point1[ x ] ) )
      cout << "The rectangle is a square.\n\n";
}
```

```
// driver for p6_13.cpp

#include "p6_13.h"

int main()
{
   double w[ 2 ] = { 1.0, 1.0 }, x[ 2 ] = { 5.0, 1.0 },
          y[ 2 ] = { 5.0, 3.0 }, z[ 2 ] = { 1.0, 3.0 },
          j[ 2 ] = { 0.0, 0.0 }, k[ 2 ] = { 1.0, 0.0 },
          m[ 2 ] = { 1.0, 1.0 }, n[ 2 ] = { 0.0, 1.0 },
          v[ 2 ] = { 99.0, -2.3 };
   Rectangle a( z, y, x, w ), b( j, k, m, n ),
             c( w, x, m, n ), d( v, x, y, z );

   return 0;
}
```

```
length = 4.0    width = 2.0
The perimeter is: 12.0
The area is: 8.0

length = 1.0    width = 1.0
The perimeter is: 4.0
The area is: 1.0

The rectangle is a square.

Coordinates do not form a rectangle!
Coordinates do not form a rectangle!
```

6.14 Modify the **Rectangle** class of Exercise 6.13 to include a **draw** function that displays the rectangle inside a 25-by-25 box enclosing the portion of the first quadrant in which the rectangle resides. Include a **setFillCharacter** function to specify the character out of which the body of the rectangle will be drawn. Include a **setPerimeterCharacter** function to specify the character that will be used to draw the border of the rectangle. If you feel ambitious, you might include functions to scale the size of the rectangle, rotate it, and move it around within the designated portion of the first quadrant.

```
// P6_14.H
#ifndef P6_14_H
#define P6_14_H

class Rectangle {
public:
   Rectangle( double *, double *, double *, double *, char, char );
   void setCoord( double *, double *, double *, double * );
   void perimeter( void );
   void area( void );
   void draw( void );
   void square( void );
   void setFillCharacter( char c ) { fillChar = c; }
   void setPerimeterCharacter( char c ) { periChar = c;}
   bool isValid( void ) { return valid; }
   void setValid( bool v ) { valid = v; }
private:
   double point1[ 2 ];
   double point2[ 2 ];
   double point3[ 2 ];
   double point4[ 2 ];
   char fillChar;
   char periChar;
   bool valid;
};

#endif
```

```
// P6_14M.cpp
// member function definitions for p6_14.cpp
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include <cmath>

#include "p6_14.h"

Rectangle::Rectangle( double *a, double *b, double *c, double *d,
                      char x, char y )
{
   setCoord( a, b, c, d );
   setFillCharacter( x );
   setPerimeterCharacter( y );
}

void Rectangle::setCoord( double *p1, double *p2,
                           double *p3, double *p4 )
{
   // Arrangement of points
   // p4.........p3
   //  .          .
   //  .          .
   // p1.........p2

   const int x = 0, y = 1;  // added for clarity

   // validate all points
   point1[ x ] = ( p1[ x ] > 20.0 || p1[ x ] < 0.0 )? 0.0 : p1[ x ];
   point1[ y ] = ( p1[ y ] > 20.0 || p1[ y ] < 0.0 )? 0.0 : p1[ y ];
   point2[ x ] = ( p2[ x ] > 20.0 || p2[ x ] < 0.0 )? 0.0 : p2[ x ];
   point2[ y ] = ( p2[ y ] > 20.0 || p2[ y ] < 0.0 )? 0.0 : p2[ y ];
   point3[ x ] = ( p3[ x ] > 20.0 || p3[ x ] < 0.0 )? 0.0 : p3[ x ];
   point3[ y ] = ( p3[ y ] > 20.0 || p3[ y ] < 0.0 )? 0.0 : p3[ y ];
   point4[ x ] = ( p4[ x ] > 20.0 || p4[ x ] < 0.0 )? 0.0 : p4[ x ];
   point4[ y ] = ( p4[ y ] > 20.0 || p4[ y ] < 0.0 )? 0.0 : p4[ y ];

   // verify that points form a rectangle
   if (point1[ y ] == point2[ y ] && point1[ x ] == point4[ x ] &&
       point2[ x ] == point3[ x ] && point3[ y ] == point4[ y ]) {

      perimeter();
      area();
      square();
      setValid( true );   // valid set of points
   }
   else {
      cout << "Coordinates do not form a rectangle!\n";
      setValid( false );   // invalid set of points
   }
}

void Rectangle::perimeter( void )
{
   double l = fabs( point4[ 1 ] - point1[ 1 ] ),
          w = fabs( point2[ 0 ] - point1[ 0 ] );

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "length = " << setprecision( 1 ) << ( l > w ? l : w )
        << "\twidth = " << ( l > w ? w : l )
        << "\nThe perimeter is: " << 2 * ( w + l ) << '\n'
        << resetiosflags( ios::fixed | ios::showpoint );
}
void Rectangle::area( void )
{
   double l = fabs( point4[ 1 ] - point1[ 1 ] ),
          w = fabs( point2[ 0 ] - point1[ 0 ] );

   cout << setiosflags( ios::fixed | ios::showpoint )
        << "The area is: " << setprecision( 1 ) << w * l
        << resetiosflags( ios::fixed | ios::showpoint )
        << "\n\n";
}

void Rectangle::square( void )
{
   const int x = 0, y = 1;   // added for clarity

   if ( fabs( point4[ y ] - point1[ y ] ) == fabs( point2[ x ] - point1[ x ] ) )
      cout << "The rectangle is a square.\n\n";
}

void Rectangle::draw( void )
{
   for ( double y = 25.0; y >= 0.0; --y ) {
      for ( double x = 0.0; x <= 25.0; ++x ) {
         if ( ( point1[ 0 ] == x && point1[ 1 ] == y ) ||
             ( point4[ 0 ] == x && point4[ 1 ] == y ) ) {

            // print horizontal perimeter of rectangle
            while ( x <= point2[ 0 ] ) {
               cout << periChar;
               ++x;
            }

            // print remainder of quadrant
            cout << '.';
         }
         // prints vertical perimeter of rectangle
         else if ( ( ( x <= point4[ 0 ] && x >= point1[ 0 ] ) ) &&
                     point4[ 1 ] >= y && point1[ 1 ] <= y ) {
            cout << periChar;

            // fill inside of rectangle
            for ( x++; x < point2[ 0 ]; ) {
               cout << fillChar;
               ++x;
            }

            cout << periChar;
         }
         else
            cout << '.';  // print quadrant background
      }

      cout << '\n';
   }
}
```

```
// driver for p6_14.cpp

#include "p6_14.h"

int main()
{
   double xy1[ 2 ] = { 12.0, 12.0 }, xy2[ 2 ] = { 18.0, 12.0 },
          xy3[ 2 ] = { 18.0, 20.0 }, xy4[ 2 ] = { 12.0, 20.0 };
   Rectangle a( xy1, xy2, xy3, xy4, '?', '*' );

   if ( a.isValid() )
      a.draw();

   return 0;
}
```

```
length = 8.0   width = 6.0
The perimeter is:  28.0
The area is: 48.0



..........................
..........................
..........................
............*******.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*******.......
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
```

6.15 Create a class **HugeInteger** that uses a 40-element array of digits to store integers as large as 40-digits each. Provide member functions **inputHugeInteger**, **outputHugeInteger**, **addHugeIntegers** and **substractHugeIntegers**. For comparing **HugeInteger** objects, provide functions **isEqualTo**, **isNotEqualTo**, **isGreaterThan**, **isLessThan**, **IsGreaterThanOrEqualTo** and **isLessThanOrEqualTo**—each of these is a "predicate" function that simply returns **true** if the relationship holds between the two huge integers and returns **false** if the relationship does not hold. Provide a predicate function **isZero**. If you feel ambitious, also provide member functions **multiplyHugeIntegers**, **divideHugeIntegers** and **modulusHugeIntegers.**

6.16 Create a class **TicTacToe** that will enable you to write a complete program to play the game of tic-tac-toe. The class contains as **private** data a 3-by-3 double array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square; place a 2 wherever the second player movesaEh move must be to an empty square. After each move, determine if the game has been won or if the game is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players automatically. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play thr ee-dimensional tic-tac-toe on a 4-by-4-by-4 board (Caution: This is an extremely challenging project that could take many weeks of effort!).

```
// p6.16_H
#ifndef P6_16_H
#define P6_16_H

class TicTacToe {
private:
   enum Status { WIN, DRAW, CONTINUE };
   int board[ 3 ][ 3 ];
public:
   TicTacToe();
   void makeMove( void );
   void printBoard( void );
   bool validMove( int, int );
   bool xoMove( int );
   Status gameStatus( void );
};

#endif
```

```
// P6_16M.cpp
// member function definitions for p6_16.cpp
#include <iostream>

using std::cout;
using std::cin;

#include <iomanip>

using std::setw;

#include "p6_16.h"

TicTacToe::TicTacToe()
{
   for ( int j = 0; j < 3; ++j )    // initialize board
      for ( int k = 0; k < 3; ++k )
         board[ j ][ k ] = ' ';
}

bool TicTacToe::validMove( int r, int c )
{
   return r >= 0 && r < 3 && c >= 0 && c < 3 && board[ r ][ c ] == ' ';
}
```

```

// must specify that type Status is part of the TicTacToe class.
// See Chapter 21 for a discussion of namespaces.
TicTacToe::Status TicTacToe::gameStatus( void )
{
   int a;

   // check for a win on diagonals
   if ( board[ 0 ][ 0 ] != ' ' && board[ 0 ][ 0 ] == board[ 1 ][ 1 ] &&
       board[ 0 ][ 0 ] == board[ 2 ][ 2 ] )
      return WIN;
   else if ( board[ 2 ][ 0 ] != ' ' && board[ 2 ][ 0 ] ==
   board[ 1 ][ 1 ] && board[ 2 ][ 0 ] == board[ 0 ][ 2 ] )
      return WIN;

   // check for win in rows
   for ( a = 0; a < 3; ++a )
      if ( board[ a ][ 0 ] != ' ' && board[ a ][ 0 ] ==
      board[ a ][ 1 ] && board[ a ][ 0 ] == board[ a ][ 2 ] )
         return WIN;

   // check for win in columns
   for ( a = 0; a < 3; ++a )
      if ( board[ 0 ][ a ] != ' ' && board[ 0 ][ a ] ==
      board[ 1 ][ a ] && board[ 0 ][ a ] == board[ 2 ][ a ] )
         return WIN;

   // check for a completed game
   for ( int r = 0; r < 3; ++r )
      for ( int c = 0; c < 3; ++c )
         if ( board[ r ][ c ] == ' ' )
            return CONTINUE; // game is not finished

   return DRAW; // game is a draw
}

void TicTacToe::printBoard( void )
{
   cout << "   0    1    2\n\n";

   for ( int r = 0; r < 3; ++r ) {
      cout << r;

      for ( int c = 0; c < 3; ++c ) {
         cout << setw( 3 ) << static_cast< char > ( board[ r ][ c ] );

         if ( c != 2 )
            cout << " |";
      }

      if ( r != 2 )
         cout << "\n ____|____|____"
              << "\n     |    |    \n";
   }

   cout << "\n\n";
}

void TicTacToe::makeMove( void )
{
   printBoard();

   while ( true ) {
      if ( xoMove( 'X' ) )
         break;
      else if ( xoMove( 'O' ) )
         break;
   }
}

bool TicTacToe::xoMove( int symbol )
{
   int x, y;

   do {
      cout << "Player " << static_cast< char >( symbol )
                << " enter move: ";
      cin >> x >> y;
      cout << '\n';
   } while ( !validMove( x, y ) );

   board[ x ][ y ] = symbol;
   printBoard();
   Status xoStatus = gameStatus();

   if ( xoStatus == WIN ) {
      cout << "Player " << static_cast< char >( symbol )
           << " wins!\n";
      return true;
   }
   else if ( xoStatus == DRAW ) {
      cout << "Game is a draw.\n";
      return true;
   }
   else // CONTINUE
      return false;
}
```

```
// driver for p6_16.cpp
#include "p6_16.h"

int main()
{
   TicTacToe g;
   g.makeMove();

   return 0;
}
```

```
   0    1    2

0    |    |
 ____|____|____
     |    |
1    |    |
 ____|____|____
     |    |
2    |    |

Player X enter move: 2 0

   0    1    2

0    |    |
 ____|____|____
     |    |
1    |    |
 ____|____|____
     |    |
2  X |    |

Player O enter move: 2 2

   0    1    2

0    |    |
 ____|____|____
     |    |
1    |    |
 ____|____|____
     |    |
2  X |    |  O

Player X enter move: 1 1
...
Player X enter move: 0 2

   0    1    2

0    |    |  X
 ____|____|____
     |    |
1    |  X |  O
 ____|____|____
     |    |
2  X |    |  O

Player X wins!Player X wins!
```

7

Classes: Part II Solutions

Solutions

7.3 Compare and contrast dynamic memory allocation using the C++ operators `new` and `delete`, with dynamic memory allocation using the C Standard Library functions `malloc` and `free`.

ANS: In C, dynamic memory allocation requires function calls to `malloc` and `free`. Also, `malloc` must be told the exact number of bytes to allocate (normally this is accomplished with the `sizeof` operator), then it returns a `void` pointer. C++ uses operators `new` and `delete`. The `new` operator automatically determines the number of bytes to allocate and returns a pointer to the appropriate type. The `delete` operator guarantees a call to the destructor for the object(s) being deleted.

7.4 Explain the notion of friendship in C++. Explain the negative aspects of friendship as described in the text.

ANS: Functions that are declared as `friend`s of a class have access to that class's `private` and `protected` members. Some people in the object-oriented programming community prefer not to use `friend` functions because they break the encapsulation of a class—i.e., they allow direct access to a class's impementation details that are supposed to be hidden.

7.5 Can a correct `Time` class definition include both of the following constructors? If not, explain why not.

```
Time( int h = 0, int m = 0, int s = 0 );
Time();
```

ANS: No, because there is ambiguity between the two constructors. When a call is made to the default constructor, the compiler cannot determine which one to use because they can both be called with no arguments.

7.6 What happens when a return type, even `void`, is specified for a constructor or destructor?

ANS: A compiler syntax error occurs. No return types can be specified for constructors.

7.7 Create a `Date` class with the following capabilities:

a) Output the date in multiple formats such as

```
DDD YYYY
MM/DD/YY
June 14, 1992
```

b) Use overloaded constructors to create `Date` objects initialized with dates of the formats in part (a).

c) Create a `Date` constructor that reads the system date using the standard library functions of the `ctime` header and sets the `Date` members.

In Chapter 8, we will be able to create operators for testing the equality of two dates and for comparing dates to determine if one date is prior to, or after, another.

```
// P7_07.H
#ifndef p7_07_H
#define p7_07_H

#include <ctime>
#include <cstring>

class Date {
public:
   Date();
   Date( int, int );
   Date( int, int, int );
   Date( char *, int, int );
   void setMonth( int );
   void setDay( int );
   void setYear( int );
   void printDateSlash( void ) const;
   void printDateMonth( void ) const;
   void printDateDay( void ) const;
   const char *monthName( void ) const;
   bool leapYear( void ) const;
   int daysOfMonth( void ) const;
   void convert1( int );
   int convert2( void ) const;
   void convert3( const char * const );
   const char *monthList( int ) const;
   int days( int ) const;
private:
   int day;
   int month;
   int year;
};

#endif
```

```
// P7_07M.cpp
// member function definitions for p7_07.cpp
#include <iostream>

using std::cout;

#include <cstring>
#include <ctime>

#include "p7_07.h"

// Date constructor that uses functions from ctime
Date::Date()
{
   struct tm *ptr;             // pointer of type struct tm
                               // which holds calendar time components
   time_t t = time( 0 );       // determine the current calendar time
                               // which is assigned to timePtr
   ptr = localtime( &t );      // convert the current calendar time
                               // pointed to by timePtr into
                               // broken down time and assign it to ptr
   day = ptr->tm_mday;         // broken down day of month
   month = 1 + ptr->tm_mon;    // broken down month since January
   year = ptr->tm_year + 1900; // broken down year since 1900
}

// Date constructor that uses day of year and year
Date::Date( int ddd, int yyyy )
{
   setYear( yyyy );
   convert1( ddd );  // convert to month and day
}

// Date constructor that uses month, day and year
Date::Date( int mm, int dd, int yy )
{
   setYear( yy + 1900 );
   setMonth( mm );
   setDay( dd );
}

// Date constructor that uses month name, day and year
Date::Date( char *mPtr, int dd, int yyyy )
{
   setYear( yyyy );
   convert3( mPtr );
   setDay( dd );
}

// Set the day
void Date::setDay( int d )
   { day = d >= 1 && d <= daysOfMonth() ? d : 1; }

// Set the month
void Date::setMonth( int m ) { month = m >= 1 && m <= 12 ? m : 1; }

// Set the year
void Date::setYear( int y ) { year = y >= 1900 && y <= 1999 ? y : 1900; }

// Print Date in the form: mm/dd/yyyy
void Date::printDateSlash( void ) const
   { cout << month << '/' << day << '/' << year << '\n'; }

// Print Date in the form: monthname dd, yyyy
void Date::printDateMonth( void ) const
   { cout << monthName() << ' ' << day << ", " << year << '\n'; }

// Print Date in the form: ddd yyyy
void Date::printDateDay( void ) const
   { cout << convert2() << ' ' << year << '\n'; }

// Return the month name
const char *Date::monthName( void ) const { return monthList( month - 1 ); }

// Return the number of days in the month
int Date::daysOfMonth( void ) const
  { return leapYear() && month == 2 ? 29 : days( month ); }

// Test for a leap year
bool Date::leapYear( void ) const
{
   if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
      return true;
   else
      return false;
```

```
}

// Convert ddd to mm and dd
void Date::convert1( int ddd )  // convert to mm / dd / yyyy
{
   int dayTotal = 0;

   if ( ddd < 1 || ddd > 366 )  // check for invalid day
      ddd = 1;

   setMonth( 1 );

   for ( int m = 1; m < 13 && ( dayTotal + daysOfMonth() ) < ddd; ++m ) {
      dayTotal += daysOfMonth();
      setMonth( m + 1 );
   }

   setDay( ddd - dayTotal );
   setMonth( m );
}

// Convert mm and dd to ddd
int Date::convert2( void ) const   // convert to a ddd yyyy format
{
   int ddd = 0;

   for ( int m = 1; m < month; ++m )
      ddd += days( m );

   ddd += day;
   return ddd;
}

// Convert from month name to month number
void Date::convert3( const char * const mPtr )   // convert to mm / dd / yyyy
{
   bool flag = false;

   for ( int subscript = 0; subscript < 12; ++subscript )
      if ( !strcmp( mPtr, monthList( subscript ) ) ) {
         setMonth( subscript + 1 );
         flag = true; // set flag
         break;    // stop checking for month
      }

   if ( !flag )
      setMonth( 1 ); // invalid month default is january
}

// Return the name of the month
const char *Date::monthList( int mm ) const
{
   char *months[] = { "January", "February", "March", "April", "May",
                      "June", "July", "August", "September", "October",
                      "November", "December" };

   return months[ mm ];
}

// Return the days in the month
int Date::days( int m ) const
{
   const int monthDays[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

   return monthDays[ m - 1 ];
}
```

```
// driver for p7_07.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p7_07.h"

int main()
{
   Date d1( 7, 4, 98 ), d2( 86, 1999 ),
        d3, d4( "September", 1, 1998 );

   d1.printDateSlash();   // format m / dd / yy
   d2.printDateSlash();
   d3.printDateSlash();
   d4.printDateSlash();
   cout << '\n';

   d1.printDateDay();     // format ddd yyyy
   d2.printDateDay();
   d3.printDateDay();
   d4.printDateDay();
   cout << '\n';

   d1.printDateMonth();   // format "month" d, yyyy
   d2.printDateMonth();
   d3.printDateMonth();
   d4.printDateMonth();
   cout << endl;

   return 0;
}
```

```
7/4/1998
3/27/1999
6/16/2000
9/1/1998

185 1998
86 1999
167 2000
244 1998

July 4, 1998
March 27, 1999
June 16, 2000
September 1, 1998
```

7.8 Create a **SavingsAccount** class. Use a **static** data member to contain the **annualInterestRate** for each of the savers. Each member of the class contains a **private** data member **savingsBalance** indicating the amount the saver currently has on deposit. Provide a **calculateMonthlyInterest** member function that calculates the monthly interest by multiplying the **balance** by **annualInterestRate** divided by 12; this interest should be added to **savingsBalance**. Provide a **stat-**

ic member function **modifyInterestRate** that sets the **static annualInterestRate** to a new value. Write a driver program to test class **SavingsAccount**. Instantiate two different **savingsAccount** objects, **saver1** and **saver2**, with balances of $2000.00 and $3000.00, respectively. Set **annualInterestRate** to 3%, then calculate the monthly interest and print the new balances for each of the savers. Then set the **annualInterestRate** to 4% and calculate the next month's interest and print the new balances for each of the savers.

```
// P7_08.H
#ifndef P7_08_H
#define P7_08_H

class SavingsAccount {
public:
   SavingsAccount( double b ) { savingsBalance = b >= 0 ? b : 0; }
   void calculateMonthlyInterest( void );
   static void modifyInterestRate( double );
   void printBalance( void ) const;
private:
   double savingsBalance;
   static double annualInterestRate;
};

#endif
```

```
// P7.08M.cpp
// Member function defintions for p7_08.cpp
#include "p7_08.h"
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

// initialize static data member
double SavingsAccount::annualInterestRate = 0.0;

void SavingsAccount::calculateMonthlyInterest( void )
   { savingsBalance += savingsBalance * ( annualInterestRate / 12.0 ); }

void SavingsAccount::modifyInterestRate( double i )
   { annualInterestRate = ( i >= 0 && i <= 1.0 ) ? i : 0.03; }

void SavingsAccount::printBalance( void ) const
{
   cout << setiosflags( ios::fixed | ios::showpoint )
        << '$' << setprecision( 2 ) << savingsBalance
        << resetiosflags( ios::fixed | ios::showpoint );
}
```

```
// driver for p7_08.cpp
#include <iostream>

using std::cout;
using std::endl;

```

```
#include <iomanip>

using std::setw;

#include "p7_08.h"

int main()
{
   SavingsAccount saver1( 2000.0 ), saver2( 3000.0 );

   SavingsAccount::modifyInterestRate( .03 );

   cout << "\nOutput monthly balances for one year at .03"
        << "\nBalances: Saver 1 ";
   saver1.printBalance();
   cout << "\tSaver 2 ";
   saver2.printBalance();

   for ( int month = 1; month <= 12; ++month ) {
      saver1.calculateMonthlyInterest();
      saver2.calculateMonthlyInterest();

      cout << "\nMonth" << setw( 3 ) << month << ": Saver 1 ";
      saver1.printBalance();
      cout << "\tSaver 2 ";
      saver2.printBalance();
   }

   SavingsAccount::modifyInterestRate( .04 );
   saver1.calculateMonthlyInterest();
   saver2.calculateMonthlyInterest();
   cout << "\nAfter setting interest rate to .04"
        << "\nBalances: Saver 1 ";
   saver1.printBalance();
   cout << "\tSaver 2 ";
   saver2.printBalance();
   cout << endl;

   return 0;
}
```

```
 Output monthly balances for one year at .03
Balances: Saver 1 $2000.00      Saver 2 $3000.00
Month  1: Saver 1 $2005.00      Saver 2 $3007.50
Month  2: Saver 1 $2010.01      Saver 2 $3015.02
Month  3: Saver 1 $2015.04      Saver 2 $3022.56
Month  4: Saver 1 $2020.08      Saver 2 $3030.11
Month  5: Saver 1 $2025.13      Saver 2 $3037.69
Month  6: Saver 1 $2030.19      Saver 2 $3045.28
Month  7: Saver 1 $2035.26      Saver 2 $3052.90
Month  8: Saver 1 $2040.35      Saver 2 $3060.53
Month  9: Saver 1 $2045.45      Saver 2 $3068.18
Month 10: Saver 1 $2050.57      Saver 2 $3075.85
Month 11: Saver 1 $2055.69      Saver 2 $3083.54
Month 12: Saver 1 $2060.83      Saver 2 $3091.25
After setting interest rate to .04
Balances: Saver 1 $2067.70      Saver 2 $3101.55
```

7.9 Create a class called `IntegerSet`. Each object of class `IntegerSet` can hold integers in the range 0 through 100. A set is represented internally as an array of ones and zeros. Array element `a[ i ]` is 1 if integer *i* is in the set. Array element `a[ j ]` is 0 if integer *j* is not in the set. The default constructor initializes a set to the so-called “empty set,” i.e., a set whose array representation contains all zeros.

Provide member functions for the common set operations. For example, provide a `unionOfIntegerSets` member function that creates a third set which is the set-theoretic union of two existing sets (i.e., an element of the third set’s array is set to 1 if that element is 1 in either or both of the existing sets, and an element of the third set’s array is set to 0 if that element is 0 in each of the existing sets).

Provide an `intersectionOfIntegerSets` member function that creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set’s array is set to 0 if that element is 0 in either or both of the e xisting sets, and an element of the third set’s array is set to 1 if that element is 1 in each of the existing sets).

Provide an `insertElement` member function that inserts a new integer *k* into a set (by setting `a[k]` to 1). Provide a `deleteElement` member function that deletes integer *m* (by setting `a[m]` to 0).

Provide a `setPrint` member function that prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set (i.e., their position in the array has a value of 1). Print `---` for an empty set.

Provide an `isEqualTo` member function that determines if two sets are equal.

Provide an additional constructor to take five integer arguments which can be used to initialize a set object. If you want to provide fewer than five elements in the set, use default arguments of
-1 for the others.

Now write a driver program to test your `IntegerSet` class. Instantiate several `IntegerSet` objects. Test that all your member functions work properly.

```cpp
P7_09.H
#ifndef P7_09_H
#define P7_09_H

class IntegerSet {
public:
   IntegerSet() { emptySet(); }
   IntegerSet( int, int = -1, int = -1, int = -1, int = -1 );
   IntegerSet unionOfIntegerSets( const IntegerSet& );
   IntegerSet intersectionOfIntegerSets( const IntegerSet& );
   void emptySet( void );
   void inputSet( void );
   void insertElement( int );
   void deleteElement( int );
   void setPrint( void ) const;
   bool isEqualTo( const IntegerSet& ) const;
private:
   int set[ 101 ];  // range of 0 - 100
   int validEntry( int x ) const { return x >= 0 && x <= 100; }
};

#endif
```

```cpp
//P7_09M.cpp
//Member function definitions for p7_09.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::cerr;

#include <iomanip>

```

```
using std::setw;

#include "p7_09.h"

IntegerSet::IntegerSet( int a, int b, int c, int d, int e )
{
   emptySet();

   if ( validEntry( a ) )
      insertElement( a );

   if ( validEntry( b ) )
      insertElement( b );

   if ( validEntry( c ) )
      insertElement( c );

   if ( validEntry( d ) )
      insertElement( d );

   if ( validEntry( e ) )
      insertElement( e );
}

void IntegerSet::emptySet( void )
{
   for ( int y = 0; y < 101; ++y )
      set[ y ] = 0;
}

void IntegerSet::inputSet( void )
{
   int number;

   do {
      cout << "Enter an element (-1 to end): ";
      cin >> number;

      if ( validEntry( number ) )
         set[ number ] = 1;
      else if ( number != -1 )
         cerr << "Invalid Element\n";
   } while ( number != -1 );

   cout << "Entry complete\n";
}

void IntegerSet::setPrint( void ) const
{
   int x;
   bool empty = true;  // assume set is empty

   cout << '{';

   for (int u = 0; u < 101; ++u )
      if ( set[ u ] ) {
         cout << setw( 4 ) << u << ( x % 10 == 0 ? "\n" : "" );
         empty = false; // set is not empty
         ++x;
      }

   if ( empty )
```

```
      cout << setw( 4 ) << "---";  // display an empty set

   cout << setw( 4 ) << "}" << '\n';
}

IntegerSet IntegerSet::unionOfIntegerSets( const IntegerSet &r )
{
   IntegerSet temp;

   for ( int n = 0; n < 101; ++n )
      if ( set[ n ] == 1 || r.set[ n ] == 1 )
         temp.set[ n ] = 1;

   return temp;
}

IntegerSet IntegerSet::intersectionOfIntegerSets( const IntegerSet &r )
{
   IntegerSet temp;

   for ( int w = 0; w < 101; ++w )
      if ( set[ w ] == 1 && r.set[ w ] == 1 )
         temp.set[ w ] = 1;

   return temp;
}

void IntegerSet::insertElement( int k )
{
   if ( validEntry( k ) )
      set[ k ] = 1;
   else
      cerr << "Invalid insert attempted!\n";
}

void IntegerSet::deleteElement( int m )
{
   if ( validEntry( m ) )
      set[ m ] = 0;
   else
      cerr << "Invalid delete attempted!\n";
}

bool IntegerSet::isEqualTo( const IntegerSet &r ) const
{
   for ( int v = 0; v < 101; ++v )
      if ( set[ v ] != r.set[ v ] )
         return false;   // sets are not-equal

   return true;   // sets are equal
}
```

```
// driver for p7_09.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p7_09.h"

int main()
```

```
{

   IntegerSet a, b, c, d, e( 8, 5, 7 );

   cout << "Enter set A:\n";
   a.inputSet();
   cout << "\nEnter set B:\n";
   b.inputSet();
   c = a.unionOfIntegerSets( b );
   d = a.intersectionOfIntegerSets( b );
   cout << "\nUnion of A and B is:\n";
   c.setPrint();
   cout << "Intersection of A and B is:\n";
   d.setPrint();

   if ( a.isEqualTo( b ) )
      cout << "Set A is equal to set B\n";
   else
      cout << "Set A is not equal to set B\n";

   cout << "\nInserting 77 into set A...\n";
   a.insertElement( 77 );
   cout << "Set A is now:\n";
   a.setPrint();

   cout << "\nDeleting 77 from set A...\n";
   a.deleteElement( 77 );
   cout << "Set A is now:\n";
   a.setPrint();

   cout << "\nSet e is:\n";
   e.setPrint();

   cout << endl;

   return 0;
}
```

```
Enter set A:
Enter an element (-1 to end): 1
Enter an element (-1 to end): 2
Enter an element (-1 to end): 3
Enter an element (-1 to end): 4
Enter an element (-1 to end): -1
Entry complete

Enter set B:
Enter an element (-1 to end): 3
Enter an element (-1 to end): 4
Enter an element (-1 to end): 5
Enter an element (-1 to end): 9
Enter an element (-1 to end): 22
Enter an element (-1 to end): -1
Entry complete

Union of A and B is:
{   1   2   3   4   5   9  22   }
Intersection of A and B is:
{   3   4   }
Set A is not equal to set B

Inserting 77 into set A...
Set A is now:
{   1   2   3   4  77   }

Deleting 77 from set A...
Set A is now:
{   1   2   3   4   }

Set e is:
{   5   7   8   }
```

7.10 It would be perfectly reasonable for the **`Time`** class of Fig. 7.8 to represent the time internally as the number of seconds since midnight rather than the three integer values **`hour`**, **`minute`** and **`second`**. Clients could use the same **`public`** methods and get the same results. Modify the **`Time`** class of Fig. 7.8 to implement the **`Time`** as the number of seconds since midnight and show that there is no visible change in functionality to the clients of the class.

8

Operator Overloading Solutions

Solutions

8.6 Give as many examples as you can of operator overloading implicit in C++. Give a reasonable example of a situation in which you might want to overload an operator explicitly in C++.

ANS: In C, the operators **`+`**, **`-`**, **`*`**, and **`&`** are overloaded. The context of these operators determines how they are used. It can be argued that the arithmetic operators are all overloaded, because they can be used to perform operations on more than one type of data. In C++, the same operators as in C are overloaded, as well as **`<<`** and **`>>`**.

8.7 The C++ operators that cannot be overloaded are __________, __________, __________,__________ and __________.

ANS: **`sizeof`**, **`.`**, **`?:`**, **`.*`**, and **`::`**.

8.8 String concatenation requires two operands—the two strings that are to be concatenated. In the text we showed how to implement an overloaded concatenation operator that concatenates the second **`String`** object to the right of the first **`String`** object, thus modifying the first **`String`** object. In some applications, it is desirable to produce a concatenated **`String`** object without modifying the **`String`** arguments. Implement **`operator+`** to allow operations such as

```
string1 = string2 + string3;
```

```
// P8_08.H
#ifndef p8_08_H
#define p8_08_H

#include <cstring>
#include <cassert>

class String {
   friend ostream &operator<<( ostream &output, const String &s );
public:
   String( const char * const = "" ); // conversion constructor
   String( const String & );    // copy constructor
   ~String();                   // destructor
   const String &operator=( const String & );
   String operator+( const String & );
private:
   char *sPtr;
   int length;
};
```

```

#endif
```

```
// P8_08M.cpp
// member function definitions for p8_08.cpp
// class String
#include <iostream>

using std::cout;
using std::ostream;


#include <cstring>
#include "p8_08.h"

// Conversion constructor: Convert a char * to String
String::String( const char * const zPtr )
{
   length = strlen( zPtr );          // compute length
   sPtr = new char[ length + 1 ]; // allocate storage
   assert( sPtr != 0 );  // terminate if memory not allocated
   strcpy( sPtr, zPtr );             // copy literal to object
}

// Copy constructor
String::String( const String &copy )
{
   length = copy.length;        // copy length
   sPtr = new char[ length + 1 ]; // allocate storage
   assert( sPtr != 0 );           // ensure memory allocated
   strcpy( sPtr, copy.sPtr );     // copy string
}

// Destructor
String::~String() { delete [] sPtr; }  // reclaim string

// Overloaded = operator; avoids self assignment
const String &String::operator=( const String &right )
{
   if ( &right != this ) {           // avoid self assignment
      delete [] sPtr;               // prevents memory leak
      length = right.length;        // new String length
      sPtr = new char[ length + 1 ]; // allocate memory
      assert( sPtr != 0 );           // ensure memory allocated
      strcpy( sPtr, right.sPtr );    // copy string
   }
   else
      cout << "Attempted assignment of a String to itself\n";

   return *this;   // enables concatenated assignments
}


// Concatenate right operand and this object and
// store in temp object.
String String::operator+( const String &right )
{
   String temp;

   temp.length = length + right.length;
   temp.sPtr = new char[ temp.length + 1 ]; // create space
   assert( sPtr != 0 );   // terminate if memory not allocated
   strcpy( temp.sPtr, sPtr );       // left part of new String
   strcat( temp.sPtr, right.sPtr ); // right part of new String
   return temp;                     // enables concatenated calls
}
ostream & operator<<( ostream &output, const String &s )
{
   output << s.sPtr;
   return output;   // enables concatenation
}
```

```
// driver for p8_08.cpp
#include <iostream>

using std::cout;
using std::endl;
using std::ostream;

// Overloaded output operator


#include "p8_08.h"

int main()
{
   String string1, string2( "The date is" );
   String string3( " August 1, 1993" );

   cout << "string1 = string2 + string3\n";
   string1 = string2 + string3;
   cout << string1 << " = " << string2 << " + "
        << string3 << endl;

   return 0;
}
```

```
string1 = string2 + string3
The date is August 1, 1993 = The date is +  August 1, 1993
```

8.9 *(Ultimate operator overloading exercise)* To appreciate the care that should go into selecting operators for overloading, list each of C++'s overloadable operators, and for each, list a possible meaning (or several, if appropriate) for each of several cla sses you have studied in this course. We suggest you try:

a) Array
b) Stack
c) String

After doing this, comment on which operators seem to have meaning for a wide variety of classes. Which operators seem to be of little value for overloading? Which operators seem ambiguous?

8.10 Now work the process described in the previous problem in reverse. List each of C++'s overloadable operators. For each, list what you feel is perhaps the "ultimate operation" the operator should be used to represent. If there are several excellent operations, list them all.

8.11 *(Project)* C++ is an evolving language, and new languages are always being developed. What additional operators would you recommend adding to C++ or to a future language like C++ that would support both procedural programming and object-oriented programming? Write a careful justification. You might consider sending your suggestions to the ANSI C++ Committee or the newsgroup `comp.std.c++`.

8.12 One nice example of overloading the function call operator `()` is to allow the more common form of double-array subscripting. Instead of saying

```
chessBoard[ row ][ column ]
```

for an array of objects, overload the function call operator to allow the alternate form

```
chessBoard( row, column )
```

```
// P8_12.H
#ifndef P8_12_H
#define P8_12_H

class CallOperator {
public:
   CallOperator();
   int operator()( int, int ); // overloaded function call operator
private:
   int chessBoard[ 8 ][ 8 ];
};

#endif
```

```
// P8_12M.CPP
// member function definitions for p8_12.cpp
#include "p8_12.h"

CallOperator::CallOperator()
{
   for ( int loop = 0; loop < 8; ++loop )
      for ( int loop2 = 0; loop2 < 8; ++loop2 )
         chessBoard[ loop ][ loop2 ] = loop2;
}

int CallOperator::operator()( int r, int c ) { return chessBoard[ r ][ c ]; }
```

```
// driver for p8_12.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p8_12.h"

int main()
{
   CallOperator board;

   cout << "board[2][5] is " << board( 2, 5 ) << endl;

   return 0;
}
```

```
board[2][5] is 5
```

8.13 Create a class **DoubleSubscriptedArray** that has similar features to class **Array** in Fig. 8.4. At construction time, the class should be able to create an array of any number of rows and any number of columns. The class should supply **operator()** to perform double-subscripting operations. For example, in a 3-by-5 **DoubleSubscriptedArray** called **a**, the user could write **a(1, 3)** to access the element at row **1** and column **3**. Remember that **operator()** can receive any number of arguments (see class **String** in Fig. 18.5 for an example of **operator()**). The underlying representation of the double-subscripted array should be a single-subscripted array of integers with *rows* * *columns* number of elements. Function **`operator()`** should perform the proper pointer arithmetic to access each element of the array. There should be two versions of **`operator()`**—one that returns **`int &`** so an element of a **`DoubleSubscriptedArray`** can be used as an *lvalue* and one that returns **`const int &`** so an element of a **`const DoubleSubscriptedArray`** can be used as an *rvalue*. The class should also provide the following operators: **`==`**, **`!=`**, **`=`**, **`<<`** (for outputting the array in row and column format) and **`>>`** (for inputting the entire array contents).

8.14 Overload the subscript operator to return the largest element of a collection, the second largest, the third largest, etc.

8.15 Consider class **`Complex`** shown in Fig. 8.7. The class enables operations on so-called *complex numbers*. These are numbers of the form **`realPart + imaginaryPart *`** *i* where *i* has the value:

$$\sqrt{-1}$$

a) Modify the class to enable input and output of complex numbers through the overloaded **`>>`** and **`<<`** operators, respectively (you should remove the **`print`** function from the class).
b) Overload the multiplication operator to enable multiplication of two complex numbers as in algebra.
c) Overload the **`==`** and **`!=`** operators to allow comparisons of complex numbers

```
// Fig. 8.7: complex1.h
// Definition of class Complex
#ifndef COMPLEX1_H
#define COMPLEX1_H

class Complex {
public:
   Complex( double = 0.0, double = 0.0 );       // constructor
   Complex operator+( const Complex & ) const;  // addition
   Complex operator-( const Complex & ) const;  // subtraction
   const Complex &operator=( const Complex & ); // assignment
   void print() const;                          // output
private:
   double real;       // real part
   double imaginary;  // imaginary part
};

#endif
```

```
// Fig. 8.7: complex1.cpp
// Member function definitions for class Complex
#include <iostream>

using std::cout;

#include "complex1.h"

// Constructor
Complex::Complex( double r, double i )
   : real( r ), imaginary( i ) { }

// Overloaded addition operator
Complex Complex::operator+( const Complex &operand2 ) const
{
   return Complex( real + operand2.real,
                   imaginary + operand2.imaginary );
}

// Overloaded subtraction operator
Complex Complex::operator-( const Complex &operand2 ) const
{
   return Complex( real - operand2.real,
                   imaginary - operand2.imaginary );
}

// Overloaded = operator
const Complex& Complex::operator=( const Complex &right )
{
   real = right.real;
   imaginary = right.imaginary;
   return *this;   // enables cascading
}

// Display a Complex object in the form: (a, b)
void Complex::print() const
   { cout << '(' << real << ", " << imaginary << ')'; }
```

```
// Fig. 8.7: fig08_07.cpp
// Driver for class Complex
#include <iostream>

using std::cout;
using std::endl;

#include "complex1.h"

int main()
{
   Complex x, y( 4.3, 8.2 ), z( 3.3, 1.1 );

   cout << "x: ";
   x.print();
   cout << "\ny: ";
   y.print();
   cout << "\nz: ";
   z.print();

   x = y + z;
   cout << "\n\nx = y + z:\n";
   x.print();
   cout << " = ";
   y.print();
   cout << " + ";
   z.print();

   x = y - z;
   cout << "\n\nx = y - z:\n";
   x.print();
   cout << " = ";
   y.print();
   cout << " - ";
   z.print();
   cout << endl;

   return 0;
}
```

Solution:

```
// P8_15.H
#ifndef P8_15_H
#define P8_15_H
#include <iostream>

using std::ostream;
using std::istream;

class Complex {
   friend ostream &operator<<( ostream &, const Complex & );
   friend istream &operator>>( istream &, Complex & );
public:
   Complex( double = 0.0, double = 0.0 );     // constructor
   Complex operator+( const Complex& ) const; // addition
   Complex operator-( const Complex& ) const; // subtraction
   Complex operator*( const Complex& ) const; // multiplication
   Complex& operator=( const Complex& );      // assignment
   bool operator==( const Complex& ) const;
   bool operator!=( const Complex& ) const;
private:
   double real;       // real part
   double imaginary;  // imaginary part
};

#endif
```

```
// P8_15M.cpp
// member function definitions for p8_15.cpp
#include "p8_15.h"
#include <iostream>

using std::ostream;
using std::istream;

// Constructor
Complex::Complex( double r, double i )
{
   real = r;
   imaginary = i;
}

// Overloaded addition operator
Complex Complex::operator+( const Complex &operand2 ) const
{
   Complex sum;

   sum.real = real + operand2.real;
   sum.imaginary = imaginary + operand2.imaginary;
   return sum;
}

// Overloaded subtraction operator
Complex Complex::operator-( const Complex &operand2 ) const
{
   Complex diff;

   diff.real = real - operand2.real;
   diff.imaginary = imaginary - operand2.imaginary;
   return diff;
}

// Overloaded multiplication operator
Complex Complex::operator*( const Complex &operand2 ) const
{
   Complex times;

   times.real = real * operand2.real + imaginary * operand2.imaginary;
   times.imaginary = real * operand2.imaginary + imaginary * operand2.real;
   return times;
}

// Overloaded = operator
Complex& Complex::operator=( const Complex &right )
{
   real = right.real;
   imaginary = right.imaginary;
   return *this;   // enables concatenation
}

bool Complex::operator==( const Complex &right ) const
   { return right.real == real && right.imaginary == imaginary ? true : false; }

bool Complex::operator!=( const Complex &right ) const
   { return !( *this == right ); }

ostream& operator<<( ostream &output, const Complex &complex )
{
   output << complex.real << " + " << complex.imaginary << 'i';
   return output;
}

istream& operator>>( istream &input, Complex &complex )
{
   input >> complex.real;
   input.ignore( 3 );       // skip spaces and +
   input >> complex.imaginary;
   input.ignore( 2 );

   return input;
}
```

```
// driver for p8_15.cpp
#include <iostream>

using std::cout;
using std::cin;

#include "p8_15.h"

int main()
{
   Complex x, y( 4.3, 8.2 ), z( 3.3, 1.1 ), k;

   cout << "Enter a complex number in the form: a + bi\n? ";
   cin >> k;

   cout << "x: " << x << "\ny: " << y << "\nz: " << z << "\nk: "
```

```
         << k << '\n';

   x = y + z;
   cout << "\nx = y + z:\n" << x << " = " << y << " + " << z << '\n';

   x = y - z;
   cout << "\nx = y - z:\n" << x << " = " << y << " - " << z << '\n';

   x = y * z;
   cout << "\nx = y * z:\n" << x << " = " << y << " * " << z << "\n\n";

   if ( x != k )
      cout << x << " != " << k << '\n';

   cout << '\n';

   x = k;

   if ( x == k )
      cout << x << " == " << k << '\n';

   return 0;
}
```

```
Enter a complex number in the form: a + bi
? 0 + 0i
x: 0 + 0i
y: 4.3 + 8.2i
z: 3.3 + 1.1i
k: 0 + 0i

x = y + z:
7.6 + 9.3i = 4.3 + 8.2i + 3.3 + 1.1i

x = y - z:
1 + 7.1i = 4.3 + 8.2i - 3.3 + 1.1i

x = y * z:
23.21 + 31.79i = 4.3 + 8.2i * 3.3 + 1.1i

23.21 + 31.79i != 0 + 0i

0 + 0i == 0 + 0i
```

8.16 A machine with 32-bit integers can represent integers in the range of approximately –2 billion to +2 billion. This fixed-size restriction is rarely troublesome. But there are applications in which we would like to be able to use a much wider range of integers. This is what C++ was built to do, namely create powerful new data types. Consider class **`HugeInt`** of Fig. 8.8. Study the class carefully, then

a) Describe precisely how it operates.
b) What restrictions does the class have?
c) Overload the **`*`** multiplication operator.
d) Overload the **`/`** division operator.

e) Overload all the relational and equality operators.

```
// Fig. 8.8: hugeint1.h
// Definition for class HugeInt
#ifndef HUGEINT1_H
#define HUGEINT1_H

#include <iostream>

using std::ostream;

class HugeInt {
   friend ostream &operator<<( ostream &, const HugeInt & );
public:
   HugeInt( long = 0 );       // conversion/default constructor
   HugeInt( const char * );            // conversion constructor
   HugeInt operator+( const HugeInt & ); // add another HugeInt
   HugeInt operator+( int );           // add an int
   HugeInt operator+( const char * ); // add an int in a char *
private:
   short integer[ 30 ];
};

#endif
```

```
// Fig. 8.8: hugeint1.cpp
// Member and friend function definitions for class HugeInt
#include <cstring>
#include "hugeint1.h"

// Conversion constructor
HugeInt::HugeInt( long val )
{
   int i;

   for ( i = 0; i <= 29; i++ )
      integer[ i ] = 0;   // initialize array to zero

   for ( i = 29; val != 0 && i >= 0; i-- ) {
      integer[ i ] = val % 10;
      val /= 10;
   }
}

HugeInt::HugeInt( const char *string )
{
   int i, j;

   for ( i = 0; i <= 29; i++ )
      integer[ i ] = 0;

   for ( i = 30 - strlen( string ), j = 0; i <= 29; i++, j++ )
      if ( isdigit( string[ j ] ) )
         integer[ i ] = string[ j ] - '0';
}

// Addition
```

```
HugeInt HugeInt::operator+( const HugeInt &op2 )
{
   HugeInt temp;
   int carry = 0;

   for ( int i = 29; i >= 0; i-- ) {
      temp.integer[ i ] = integer[ i ] +
                          op2.integer[ i ] + carry;

      if ( temp.integer[ i ] > 9 ) {
         temp.integer[ i ] %= 10;
         carry = 1;
      }
      else
         carry = 0;
   }

   return temp;
}

// Addition
HugeInt HugeInt::operator+( int op2 )
   { return *this + HugeInt( op2 ); }

// Addition
HugeInt HugeInt::operator+( const char *op2 )
   { return *this + HugeInt( op2 ); }

ostream& operator<<( ostream &output, const HugeInt &num )
{
   int i;

   for ( i = 0; ( num.integer[ i ] == 0 ) && ( i <= 29 ); i++ )
      ; // skip leading zeros

   if ( i == 30 )
      output << 0;
   else
      for ( ; i <= 29; i++ )
         output << num.integer[ i ];

   return output;
}
```

```
// Fig. 8.8: fig08_08.cpp
// Test driver for HugeInt class
#include <iostream>

using std::cout;
using std::endl;

#include "hugeint1.h"

int main()
{
   HugeInt n1( 7654321 ), n2( 7891234 ),
           n3( "99999999999999999999999999999" ),
           n4( "1" ), n5;

   cout << "n1 is " << n1 << "\nn2 is " << n2
        << "\nn3 is " << n3 << "\nn4 is " << n4
```

```
          << "\nn5 is " << n5 << "\n\n";

   n5 = n1 + n2;
   cout << n1 << " + " << n2 << " = " << n5 << "\n\n";

   cout << n3 << " + " << n4 << "\n= " << ( n3 + n4 )
        << "\n\n";

   n5 = n1 + 9;
   cout << n1 << " + " << 9 << " = " << n5 << "\n\n";

   n5 = n2 + "10000";
   cout << n2 << " + " << "10000" << " = " << n5 << endl;

   return 0;
}
```

```
n1 is 7654321
n2 is 7891234
n3 is 99999999999999999999999999999
n4 is 1
n5 is 0

7654321 + 7891234 = 15545555

99999999999999999999999999999 + 1
= 100000000000000000000000000000

7654321 + 9 = 7654330

7891234 + 10000 = 7901234
```

8.17 Create a class **RationalNumber** (fractions) with the following capabilities:

a) Create a constructor that prevents a 0 denominator in a fraction, reduces or simplifies fractions that are not in reduced form and avoids negative denominators.

b) Overload the addition, subtraction, multiplication and division operators for this class.

c) Overload the relational and equality operators for this class.

```
// P8_17.H
#ifndef P8_17_H
#define P8_17_H

class RationalNumber {
public:
   RationalNumber( int = 0, int = 1 ); // default constructor
   RationalNumber operator+( const RationalNumber& );
   RationalNumber operator-( const RationalNumber& );
   RationalNumber operator*( const RationalNumber& );
   RationalNumber operator/( RationalNumber& );
   bool operator>( const RationalNumber& ) const;
   bool operator<( const RationalNumber& ) const;
   bool operator>=( const RationalNumber& ) const;
   bool operator<=( const RationalNumber& ) const;
   bool operator==( const RationalNumber& ) const;
   bool operator!=( const RationalNumber& ) const;
   void printRational( void ) const;
private:
   int numerator;
   int denominator;
   void reduction( void );
};

#endif
```

```
// P8_17M.cpp
// member function definitions for p8_17.cpp
#include <cstdlib>

#include <iostream>

using std::cout;
using std::endl;

#include "p8_17.h"

RationalNumber::RationalNumber( int n, int d )
{
   numerator = n;
   denominator = d;
   reduction();
}

RationalNumber RationalNumber::operator+( const RationalNumber &a )
{
   RationalNumber sum;

   sum.numerator = numerator * a.denominator + denominator * a.numerator;
   sum.denominator = denominator * a.denominator;
   sum.reduction();
   return sum;
}

RationalNumber RationalNumber::operator-( const RationalNumber &s )
{
   RationalNumber sub;

   sub.numerator = numerator * s.denominator - denominator * s.numerator;
   sub.denominator = denominator * s.denominator;
   sub.reduction();
   return sub;
}

RationalNumber RationalNumber::operator*( const RationalNumber &m )
{
   RationalNumber multiply;

   multiply.numerator = numerator * m.numerator;
   multiply.denominator = denominator * m.denominator;
   multiply.reduction();
   return multiply;
}

RationalNumber RationalNumber::operator/( RationalNumber &d )
{
   RationalNumber divide;

   if ( d.numerator != 0 ) {   // check for a zero in numerator
      divide.numerator = numerator * d.denominator;
      divide.denominator = denominator * d.numerator;
      divide.reduction();
   }
   else {
      cout << "Divide by zero error: terminating program" << endl;
      exit( 1 );   // stdlib function
   }

   return divide;
}

bool RationalNumber::operator>( const RationalNumber &gr ) const
{
   if ( static_cast<double>( numerator ) / denominator >
        static_cast<double>( gr.numerator ) / gr.denominator )
      return true;
   else
      return false;
}

bool RationalNumber::operator<(const RationalNumber &lr) const
{
   if ( static_cast<double>( numerator ) / denominator <
        static_cast<double>( lr.numerator ) / lr.denominator )
      return true;
   else
      return false;
}

bool RationalNumber::operator>=( const RationalNumber &ger ) const
   { return *this == ger || *this > ger; }

bool RationalNumber::operator<=( const RationalNumber &ler ) const
   { return *this == ler || *this < ler; }

bool RationalNumber::operator==( const RationalNumber &er ) const
{
   if ( numerator == er.numerator && denominator == er.denominator )
      return true;
   else
      return false;
}

bool RationalNumber::operator!=( const RationalNumber &ner ) const
   { return !( *this == ner ); }

void RationalNumber::printRational( void ) const
{
   if ( numerator == 0 )          // print fraction as zero
      cout << numerator;
   else if ( denominator == 1 )  // print fraction as integer
      cout << numerator;
   else
      cout << numerator << '/' << denominator;
}

void RationalNumber::reduction( void )
{
   int largest, gcd = 1;  // greatest common divisor;

   largest = ( numerator > denominator ) ? numerator: denominator;

   for ( int loop = 2; loop <= largest; ++loop )
       if ( numerator % loop == 0 && denominator % loop == 0 )
```

```
         gcd = loop;

   numerator /= gcd;
   denominator /= gcd;
}
```

```
// driver for p8_17.cpp
#include "p8_17.h"
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   RationalNumber c( 7, 3 ), d( 3, 9 ), x;

   c.printRational();
   cout << " + " ;
   d.printRational();
   cout << " = ";
   x = c + d;
   x.printRational();

   cout << '\n';
   c.printRational();
   cout << " - " ;
   d.printRational();
   cout << " = ";
   x = c - d;
   x.printRational();

   cout << '\n';
   c.printRational();
   cout << " * " ;
   d.printRational();
   cout << " = ";
   x = c * d;
   x.printRational();

   cout << '\n';
   c.printRational();
   cout << " / " ;
   d.printRational();
   cout << " = ";
   x = c / d;
   x.printRational();

   cout << '\n';
   c.printRational();
   cout << " is:\n";

   cout << ( ( c > d ) ? "  > " : "  <= " );
   d.printRational();
   cout << " according to the overloaded > operator\n";

   cout << ( ( c < d ) ? "  < " : "  >= " );
   d.printRational();
   cout << " according to the overloaded < operator\n";

   cout << ( ( c >= d ) ? "  >= " : "  < " );
   d.printRational();
   cout << " according to the overloaded >= operator\n";

   cout << ( ( c <= d ) ? "  <= " : "  > " );
   d.printRational();
   cout << " according to the overloaded <= operator\n";

   cout << ( ( c == d ) ? "  == " : "  != " );
   d.printRational();
   cout << " according to the overloaded == operator\n";

   cout << ( ( c != d ) ? "  != " : "  == " );
   d.printRational();
   cout << " according to the overloaded != operator" << endl;

   return 0;
}
```

```
7/3 + 1/3 = 8/3
7/3 - 1/3 = 2
7/3 * 1/3 = 7/9
7/3 / 1/3 = 7
7/3 is:
 > 1/3 according to the overloaded > operator
 >= 1/3 according to the overloaded < operator
 >= 1/3 according to the overloaded >= operator
 > 1/3 according to the overloaded <= operator
 != 1/3 according to the overloaded == operator
 != 1/3 according to the overloaded != operator
```

8.18 Study the C string-handling library functions and implement each of the functions as part of the **String** class. Then, use these functions to perform text manipulations.

8.19 Develop class **Polynomial**. The internal representation of a **Polynomial** is an array of terms. Each term contains a coefficient and an exponent. The term

$2x^4$

has a coefficient of 2 and an exponent of 4. Develop a full class containing proper constructor and destructor functions as well as *set* and *get* functions. The class should also provide the following overloaded operator capabilities:

a) Overload the addition operator (**+**) to add two **Polynomials**.
b) Overload the subtraction operator (**-**) to subtract two **Polynomials**.
c) Overload the assignment operator (=) to assign one **Polynomial** to another.
d) Overload the multiplication operator (*****) to multiply two **Polynomials**.
e) Overload the addition assignment operator (**+=**), the subtraction assignment operator (**-=**), and the multiplication assignment operator (***=**).

```
// P8_19P.H
#ifndef P8_19P_H
#define P8_19P_H

class Polynomial {
public:
   Polynomial();
   Polynomial operator+( const Polynomial& ) const;
   Polynomial operator-( const Polynomial& ) const;
   Polynomial operator*( const Polynomial& );
   const Polynomial operator=( const Polynomial&);
   Polynomial& operator+=( const Polynomial& );
   Polynomial& operator-=( const Polynomial& );
   Polynomial& operator*=( const Polynomial& );
   void enterTerms( void );
   void printPolynomial( void ) const;
private:
   int exponents[ 100 ];
   int coefficients[ 100 ];
   void polynomialCombine( Polynomial& );  // combine common terms
};

#endif
```

```
// P8_19M.cpp
// member function definitions for p8_19.cpp
// NOTE: The assignment operator does not need to be overloaded,
// because default member-wise copy can be used
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setiosflags;
using std::resetiosflags;

#include "p8_19p.h"

Polynomial::Polynomial()
{
   for ( int t = 0; t < 100; ++t ) {
      coefficients[ t ] = 0;
      exponents[ t ] = 0;
   }
}

void Polynomial::printPolynomial( void ) const
{
   int start;
   bool zero = false;

   if ( coefficients[ 0 ] ) {        // output constants
      cout << coefficients[ 0 ];
      start = 1;
      zero = true;    // at least one term exists
   }
   else {

      if ( coefficients[ 1 ] ) {
         cout << coefficients[ 1 ] << 'x';  // constant does not exist
                                            // so output first term
                                            // without a sign
         if ( ( exponents[ 1 ] != 0 ) && ( exponents[ 1 ] != 1 ) )
            cout << '^' << exponents[ 1 ];

         zero = true;  // at least one term exists
      }

      start = 2;
   }

   // output remaining polynomial terms
   for ( int x = start; x < 100; ++x ) {
      if ( coefficients[ x ] != 0 ) {
         cout << setiosflags( ios::showpos ) << coefficients[ x ]
              << resetiosflags( ios::showpos ) << 'x';

         if ( ( exponents[ x ] != 0 ) && ( exponents[ x ] != 1 ) )
            cout << '^' << exponents[ x ];

         zero = true;  // at least one term exists
      }
   }

   if ( !zero )   // no terms exist in the polynomial
      cout << '0';

   cout << endl;
}

const Polynomial Polynomial::operator=( const Polynomial& r )
{
   exponents[ 0 ] = r.exponents[ 0 ];
   coefficients[ 0 ] = r.coefficients[ 0 ];

   for ( int s = 1; ( s < 100 ); ++s ) {
      if ( r.exponents[ s ] != 0 ) {
         exponents[ s ] = r.exponents[ s ];
         coefficients[ s ] = r.coefficients[ s ];
      }
      else {
         if ( exponents[ s ] == 0 )
            break;
         exponents[ s ] = 0;
         coefficients[ s ] = 0;
      }
   }

   return *this;
}

Polynomial Polynomial::operator+( const Polynomial& r ) const
{
   Polynomial temp;
   bool exponentExists;

   // process element with a zero exponent
   temp.coefficients[ 0 ] = coefficients[ 0 ] + r.coefficients[ 0 ];

   // copy right arrays into temp object s will be used to keep
   // track of first open coefficient element
   for ( int s = 1; ( s < 100 ) && ( r.exponents[ s ] != 0 ); ++s ) {
      temp.coefficients[ s ] = r.coefficients[ s ];
      temp.exponents[ s ] = r.exponents[ s ];
   }

   for ( int x = 1; x < 100; ++x ) {
      exponentExists = false; // assume exponent will not be found

      for ( int t = 1; ( t < 100 ) && ( !exponentExists ); ++t )
         if ( exponents[ x ] == temp.exponents[ t ] ) {
            temp.coefficients[ t ] += coefficients[ x ];
            exponentExists = true;  // exponent found
         }

      // exponent was not found, insert into temp
      if ( !exponentExists ) {
         temp.exponents[ s ] = exponents[ x ];
         temp.coefficients[ s ] += coefficients[ x ];
         ++s;
      }
   }

   return temp;
}

Polynomial &Polynomial::operator+=( const Polynomial &r )
{
   *this = *this + r;
   return *this;
}

Polynomial Polynomial::operator-( const Polynomial& r ) const
{
   Polynomial temp;
   bool exponentExists;

   // process element with a zero exponent
   temp.coefficients[ 0 ] = coefficients[ 0 ] - r.coefficients[ 0 ];

   // copy left arrays into temp object s will be used to keep
   // track of first open coefficient element
   for ( int s = 1; ( s < 100 ) && ( exponents[ s ] != 0 ); ++s ) {
      temp.coefficients[ s ] = coefficients[ s ];
      temp.exponents[ s ] = exponents[ s ];
   }

   for ( int x = 1; x < 100; ++x) {
      exponentExists = false; // assume exponent will not be found

      for ( int t = 1; ( t < 100 ) && ( !exponentExists ); ++t )
         if ( r.exponents[ x ] == temp.exponents[ t ] ) {
            temp.coefficients[ t ] -= r.coefficients[ x ];
            exponentExists = true;  // exponent found
         }

      // exponent was not found, insert into temp
      if ( !exponentExists ) {
         temp.exponents[ s ] = r.exponents[ x ];
         temp.coefficients[ s ] -= r.coefficients[ x ];
         ++s;
      }
   }

   return temp;
}

Polynomial &Polynomial::operator-=( const Polynomial& r )
{
   *this = *this - r;
   return *this;
}

Polynomial Polynomial::operator*( const Polynomial& r )
```

```
{
   Polynomial temp;
   int s = 1;    // subscript location for temp coefficients and exponents

   for ( int x = 0; ( x < 100 ) && ( x == 0 || coefficients[ x ] != 0 ); ++x )
      for ( int y = 0; ( y < 100 ) && ( y == 0 || r.coefficients[ y ] != 0 ); ++y )
         if ( coefficients[ x ] * r.coefficients[ y ] )

            if ( ( exponents[ x ] == 0 ) && ( r.exponents[ y ] == 0 ) )
               temp.coefficients[ 0 ] += coefficients[ x ] * r.coefficients[ y ];
            else {
               temp.coefficients[ s ] = coefficients[ x ] * r.coefficients[ y ];
               temp.exponents[ s ] = exponents[ x ] + r.exponents[ y ];
               ++s;
            }

   polynomialCombine( temp );   // combine common terms
   return temp;
}

void Polynomial::polynomialCombine( Polynomial& w )
{
   Polynomial temp = w;
   int exp;

   // zero out elements of w
   for ( int x = 0; x < 100; ++x ) {
      w.coefficients[ x ] = 0;
      w.exponents[ x ] = 0;
   }

   for ( x = 1; x < 100; ++x ) {
      exp = temp.exponents[ x ];

      for ( int y = x + 1; y < 100; ++y )
         if ( exp == temp.exponents[ y ] ) {
            temp.coefficients[ x ] += temp.coefficients[ y ];
            temp.exponents[ y ] = 0;
            temp.coefficients[ y ] = 0;
         }
   }

   w = temp;
}

Polynomial &Polynomial::operator*=( const Polynomial& r )
{
   *this = *this * r;
   return *this;
}

void Polynomial::enterTerms( void )
{
   bool found = false;
   int numberOfTerms, c, e;

   cout << "\nEnter number of polynomial terms: ";
   cin >> numberOfTerms;

   for ( int n = 1; n <= numberOfTerms; ++n ) {
      cout << "\nEnter coefficient: ";
      cin >> c;
      cout << "Enter exponent: ";
      cin >> e;

      if ( c != 0 ) {
         // exponents of zero are forced into first element
         if ( e == 0 ) {
            coefficients[ 0 ] += c;
            continue;
         }

         for ( int term = 1; ( term < 100 ) &&
               ( coefficients[ term ] != 0 ); ++term )
            if ( e == exponents[ term ] ) {
               coefficients[ term ] += c;
               exponents[ term ] = e;
               found = true;  // existing exponent updated
            }

         if ( !found ) {              // add term
            coefficients[ term ] += c;
            exponents[ term ] = e;
         }
      }
   }
}
```

```
// driver for p8_19.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p8_19p.h"

int main()
{
   Polynomial a, b, c, t;

   a.enterTerms();
   b.enterTerms();
   t = a;   //save the value of a
   cout << "First polynomial is:\n";
   a.printPolynomial();
   cout << "Second polynomial is:\n";
   b.printPolynomial();
   cout << "\nAdding the polynomials yields:\n";
   c = a + b;
   c.printPolynomial();
   cout << "\n+= the polynomials yields:\n";
   a += b;
   a.printPolynomial();
   cout << "\nSubtracting the polynomials yields:\n";
   a = t;  // reset a to original value
   c = a - b;
   c.printPolynomial();
   cout << "\n-= the polynomials yields:\n";
   a -= b;
   a.printPolynomial();
   cout << "\nMultiplying the polynomials yields:\n";
   a = t;  // reset a to original value
   c = a * b;
   c.printPolynomial();
   cout << "\n*= the polynomials yields:\n";
   a *= b;
   a.printPolynomial();
   cout << endl;
   return 0;
}
```

```
Enter number of polynomial terms: 2

Enter coefficient: 2
Enter exponent: 2

Enter coefficient: 3
Enter exponent: 3

Enter number of polynomial terms: 3

Enter coefficient: 1
Enter exponent: 1

Enter coefficient: 2
Enter exponent: 2

Enter coefficient: 3
Enter exponent: 3
First polynomial is:
2x^2+3x^3
Second polynomial is:
1x+2x^2+3x^3

Adding the polynomials yields:
1x+4x^2+6x^3

+= the polynomials yields:
1x+4x^2+6x^3

Subtracting the polynomials yields:
-1x

-= the polynomials yields:
-1x

Multiplying the polynomials yields:
2x^3+7x^4+12x^5+9x^6

*= the polynomials yields:
2x^3+7x^4+12x^5+9x^6
```

8.20 The program of Fig. 8.3 contains the comment

```
// Overloaded stream-insertion operator (cannot be
// a member function if we would like to invoke it with
// cout << somePhoneNumber;)
```

Actually, it cannot be a member function of class **`ostream`**, but it can be a member function of class **`PhoneNumber`** if we were willing to invoke it in either of the following ways:

```
somePhoneNumber.operator<<( cout );
```

or

```
somePhoneNumber << cout;
```

Rewrite the program of Fig. 8.3 with the overloaded stream-insertion **`operator<<`** as a member function and try the two preceding statements in the program to prove that they work.

9

Inheritance Solutions

Solutions

9.2 Consider the class **`Bicycle`**. Given your knowledge of some common components of bicycles, show a class hierarchy in which the class **`Bicycle`** inherits from other classes, which, in turn, inherit from yet other classes. Discuss the instantiation of various objects of class **`Bicycle`**. Discuss inheritance from class **`Bicycle`** for other closely related derived classes.

ANS: Possible classes are displayed in bold.

`Bicycle` composed of:
- **`HandleBar`**s
- **`Seat`**
- **`Frame`**
- **`Wheel`**s composed of:
  - **`Tire`**s
  - **`Rim`**s composed of:
    - **`Spoke`**s
- **`Pedal`**s
- **`Chain`** composed of:
  - **`Link`**s
- **`Brake`**s composed of:
  - **`Wire`**s
  - **`BrakePad`**s
  - **`BrakeHandle`**s

Classes that can be derived from **`Bicycle`** are **`Unicycle`**, **`Tricycle`**, **`TandemBicycle`**, etc.

9.3 Briefly define each of the following terms: inheritance, multiple inheritance, base class and derived class.

ANS:

inheritance: The process by which a class incorporates the attributes and behaviors of a previously defined class.

multiple inheritance: The process by which a class incorporates the attributes and behaviors of two or more previously defined classes.

base class: A class from which other classes inherit attributes and behaviors.

derived class: A class that has inherited attributes and behaviors from one or more base classes.

9.4 Discuss why converting a base-class pointer to a derived-class pointer is considered dangerous by the compiler.
ANS: The pointer must "point" to the object of the derived class, before being dereferenced. When the compiler looks at an object through a derived-class pointer, it expects to see the all the pieces of the derived class. However, if the base-class pointer originally pointed to a base-class object, the additional pieces added by the derived class do not exist.

9.5 Distinguish between single inheritance and multiple inheritance.
ANS: Single inheritance inherits from one class only. Multiple inheritance inherits from two or more classes.

9.6 (True/False) A derived class is often called a subclass because it represents a subset of its base class (i.e., a derived classis generally smaller than its base class).
ANS: False. Derived classes are often larger than their base classes, because they need specific features in addition to those inherited from the base class. The term subclass means that the derived class is a more specific version of its base class. For example, a cat is a specific type of animal.

9.7 (True/False) A derived-class object is also an object of that derived class' base class.
ANS: True.

9.8 Some programmers prefer not to use **`protected`** access because it breaks the encapsulation of the base class. Discuss the relative merits of using **`protected`** access vs. insisting on using **`private`** access in base classes.
ANS: Inherited **`private`** data is hidden in the derived class and is accessible only through the **`public`** or **`protected`** member functions of the base class. Using **`protected`** access enables the derived class to manipulate the **`protected`** members without using the base class access functions. If the base class members are **`private`**, the **`public`** or **`protected`** member functions of the base class must be used to acces **`private`** members. This can result in additional function calls—which can decrease performance.

9.9 Many programs written with inheritance could be solved with composition instead, and vice versa. Discuss the relative merits of these approaches in the context of the **`Point`**, **`Circle`**, **`Cylinder`** class hierarchy in this chapter. Rewrite the program of Fig. 9.10 (and the supporting classes) to use composition rather than inheritance. After you do this, reassess the relative meri ts of the two approaches both for the **`Point`**, **`Circle`**, **`Cylinder`** problem and for object-oriented programs in general.

```
// P9_09.H
#ifndef P9_09_H
#define P9_09_H

#include <iostream>
using std::ostream;

class Point {
   friend ostream &operator<<( ostream &, const Point & );
public:
   Point( double a = 0, double b = 0 ) { setPoint( a, b ); }
   void setPoint( double, double );
   void print( void ) const;
   double getX( void ) const { return x; }
   double getY( void ) const { return y; }
private:
   double x, y;
};

#endif
```

```
// P9_09PM.cpp
// Member functions for class Point
#include <iostream>

using std::cout;
using std::ostream;

#include "p9_09.h"

void Point::setPoint( double a, double b )
{
   x = a;
   y = b;
}

ostream &operator<<( ostream &output, const Point &p )
{
   p.print();
   return output;
}

void Point::print( void ) const
   { cout << '[' << getX() << ", " << getY() << ']'; }
```

```
// P9_09C.H
#ifndef P9_09C_H
#define P9_09C_H
#include "p9_09.h"

class Circle {
   friend ostream &operator<<( ostream &, const Circle & );
public:
   Circle( double = 0.0, double = 0.0, double = 0.0 );
   void setRadius( double r ) { radius = r; }
   double getRadius( void ) const { return radius; }
   double area( void ) const;
   void print( void ) const;
private:
   double radius;
   Point pointObject;
};

#endif
```

```
// P9_09CM.cpp
// Member function definitions for class Circle
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include "p9_09c.h"

Circle::Circle( double r, double a, double b ) : pointObject( a, b )
   { setRadius( r ); }

double Circle::area( void ) const
   { return 3.14159 * getRadius() * getRadius(); }

```

```
ostream &operator<<( ostream &output, const Circle &c )
{
   c.print();
   return output;
}

void Circle::print( void ) const
{
   cout << "Center = ";
   pointObject.print();
   cout << "; Radius = " << setiosflags( ios::fixed | ios::showpoint )
        << setprecision( 2 ) << getRadius()
        << resetiosflags( ios::fixed | ios::showpoint );
}
```

```
// P9_09CY.H
#ifndef P9_09CY_H
#define P9_09CY_H
#include "p9_09.h"
#include "p9_09c.h"

class Cylinder {
   friend ostream& operator<<(ostream&, const Cylinder&);
public:
   Cylinder(double = 0.0, double = 0.0, double = 0.0, double = 0.0);
   void setHeight(double h) { height = h; }
   double getHeight(void) const { return height; }
   void print(void) const;
   double area(void) const;
   double volume(void) const;
private:
   double height;
   Circle circleObject;
};

#endif
```

```
// P9_09CYM.cpp
// Member function definitions for class Cylinder.
#include <iostream>

using std::cout;
using std::ostream;

#include "p9_09cy.h"

Cylinder::Cylinder( double h, double r, double x, double y )
   : circleObject( r, x, y ) { height = h; }

double Cylinder::area( void ) const
   { return 2 * circleObject.area() + 2 * 3.14159 *
     circleObject.getRadius() * getHeight(); }

ostream& operator<<( ostream &output, const Cylinder& c )
{
   c.print();
   return output;
}
```

```

double Cylinder::volume( void ) const
   { return circleObject.area() * getHeight(); }

void Cylinder::print( void ) const
{
   circleObject.print();
   cout << "; Height = " << getHeight() << '\n';
}
```

```
// P9_09.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p9_09.h"
#include "p9_09c.h"
#include "p9_09cy.h"

int main()
{
   Point p( 1.1, 8.5 );
   Circle c( 2.0, 6.4, 9.8 );
   Cylinder cyl( 5.7, 2.5, 1.2, 2.3 );

   cout << "Point: " << p << "\nCircle: " << c
        << "\nCylinder: " << cyl << endl;

   return 0;
}
```

```
Point: [1.1, 8.5]
Circle: Center = [6.4, 9.8]; Radius = 2.00
Cylinder: Center = [1.2, 2.3]; Radius = 2.50; Height = 5.7
```

9.10 Rewrite the **Point**, **Circle**, **Cylinder** program of Fig. 9.10 as a **Point**, **Square**, **Cube** program. Do this two ways—once with inheritance and once with composition.

```
// P9_10.H
#ifndef P9_10_H
#define P9_10_H

#include <iostream>
using std::ostream;

class Point {
   friend ostream &operator<<( ostream&, const Point& );
public:
   Point( double = 0, double = 0, double = 0 );
   void setPoint( double, double, double );
   double getX( void ) const { return x; }
   double getY( void ) const { return y; }
   double getZ( void ) const { return z; }
private:
   double x, y, z;
};

#endif
```

```
//P9_10MP.cpp
// member function defintions for class Point
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include "p9_10.h"

Point::Point( double a, double b, double c) { setPoint( a, b, c ); }

void Point::setPoint( double a, double b, double c )
{
   x = a;
   y = b;
   z = c;
}

ostream &operator<<( ostream &output, const Point &p )
{
   output << setiosflags( ios::fixed | ios::showpoint )
          << "The point is: [" << setprecision( 2 ) << p.x
          << ", " << setprecision( 2 ) << p.y << setprecision( 2 )
          << ", " << p.z << "]\n"
          << resetiosflags( ios::fixed | ios::showpoint );

   return output;
}
```

```
//P9_10S.H
#ifndef P9_10S_H
#define P9_10S_H
#include "p9_10.h"

#include <iostream>
using std::ostream;

class Square : public Point {
   friend ostream &operator<<( ostream &, const Square & );
public:
   Square( double = 0, double = 0, double = 0, double = 1.0 );
   void setSide( double s) { side = s > 0 && s <= 20.0 ? s : 1.0; }
   double area( void ) const { return side * side; }
   double getSide( void ) const { return side; }
protected:
   double side;
};

```

```
#endif
```

```
//P9_10MS.cpp
//member functions for class Square
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include "p9_10s.h"

Square::Square( double x, double y, double z, double s ) : Point( x, y, z )
   { setSide( s ); }

ostream &operator<<( ostream &output, const Square &s )
{
   output << setiosflags( ios::fixed | ios::showpoint )
          << "The lower left coordinate of the square is: ["
          << setprecision( 2 ) << s.getX() << ", " << setprecision( 2 )
          << s.getY() << ", " << setprecision( 2 ) << s.getZ() << ']'
          << "\nThe square side is: " << setprecision( 2 ) << s.side
          << "\nThe area of the square is: " << setprecision( 2 )
          << s.area() << '\n'
          << resetiosflags( ios::fixed | ios::showpoint );

   return output;
}
```

```
// P9_10C.H
#ifndef P9_10C_H
#define P9_10C_H
#include "p9_10s.h"

#include <iostream>
using std::ostream;

class Cube : public Square {
   friend ostream &operator<<( ostream&, const Cube& );
public:
   Cube( double = 0, double = 0, double = 0, double = 1.0 );
   double area( void ) const { return 6 * Square::area(); }
   double volume( void ) const { return Square::area() * getSide(); }
};

#endif
```

```
//P9_10MC.cpp
//member function definitions for class Cube
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include "p9_10c.h"

Cube::Cube( double j, double k, double m, double s ) : Square( j, k, m, s ) { }

ostream &operator<<( ostream &output, const Cube &c )
{
   output << setiosflags( ios::fixed | ios::showpoint )
          << "The lower left coordinate of the cube is: ["
          << setprecision( 2 ) << c.getX() << ", " << setprecision( 2 )
          << c.getY() << ", " << setprecision( 2 ) << c.getZ()
          << "]\nThe cube side is: " << setprecision( 2 ) << c.side
          << "\nThe surface area of the cube is: " << setprecision( 2 )
          << c.area() << "\nThe volume of the cube is: "
          << setprecision( 2 ) << c.volume()
          << resetiosflags( ios::showpoint | ios::fixed ) << '\n';

   return output;
}
```

```
// driver for p9_10.cpp
#include <iostream>

using std::cout;

#include "p9_10.h"
#include "p9_10s.h"
#include "p9_10c.h"

int main()
{
   Point p( 7.9, 12.5, 8.8 );
   Square s( 0.0, 0.0, 0.0, 5.0 );
   Cube c( 0.5, 8.3, 12.0, 2.0 );

   cout << p << '\n' << s << '\n' << c << '\n';

   return 0;
}
```

```
The point is: [7.90, 12.50, 8.80]

The lower left coordinate of the square is: [0.00, 0.00, 0.00]
The square side is: 5.00
The area of the square is: 25.00

The lower left coordinate of the cube is: [0.50, 8.30, 12.00]
The cube side is: 2.00
The surface area of the cube is: 24.00
The volume of the cube is: 8.00
```

```cpp
// P9_10B.H
#ifndef P9_10B_H
#define P9_10B_H

#include <iostream>
using std::ostream;

class Point {
   friend ostream &operator<<( ostream&, const Point& );
public:
   Point( double = 0, double = 0, double = 0 );
   void setPoint( double, double, double );
   void print( void ) const;
   double getX( void ) const { return x; }
   double getY( void ) const { return y; }
   double getZ( void ) const { return z; }
private:
   double x, y, z;
};

#endif
```

```cpp
//P9_10BMP.cpp
// member function defintions for class Point
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include "p9_10b.h"

Point::Point( double a, double b, double c ) { setPoint( a, b, c ); }

void Point::setPoint( double a, double b, double c )
{
   x = a;
   y = b;
   z = c;
}

ostream &operator<<( ostream &output, const Point &p )
{
   output << "The point is: ";
   p.print();
   return output;
}

void Point::print( void ) const
{
   cout << setiosflags( ios::fixed | ios::showpoint )
        << '[' << setprecision( 2 ) << getX()
        << ", " << setprecision( 2 ) << getY() << setprecision( 2 )
        << ", " << getZ() << "]\n"
        << resetiosflags( ios::fixed | ios::showpoint );
}
```

```
//P9_10BS.H
#ifndef P9_10BS_H
#define P9_10BS_H
#include "p9_10B.h"

#include <iostream>
using std::ostream;

class Square {
   friend ostream &operator<<( ostream &, const Square & );
public:
   Square( double = 0, double = 0, double = 0, double = 1.0 );
   void setSide( double s ) { side = s > 0 && s <= 20.0 ? s : 1.0; }
   void print( void ) const;
   double getXCoord() const { return pointObject.getX(); }
   double getYCoord() const { return pointObject.getY(); }
   double getZCoord() const { return pointObject.getZ(); }
   double area( void ) const { return side * side; }
   double getSide( void ) const { return side; }
protected:
   double side;
   Point pointObject;
};

#endif
```

```
//P9_10BMS.cpp
//member functions for class Square
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include "p9_10bs.h"

Square::Square( double x, double y, double z, double s )
       : pointObject( x, y, z )
   { setSide( s ); }

ostream &operator<<( ostream &output, const Square &s )
{
   s.print();
   return output;
}

void Square::print( void ) const
{
   cout << setiosflags( ios::fixed | ios::showpoint )
        << "The lower left coordinate of the square is: ";
   pointObject.print();
   cout << "The square side is: " << setprecision( 2 ) << getSide()
        << "\nThe area of the square is: " << setprecision( 2 )
        << area() << '\n';
}
```

```
// P9_10BC.H
#ifndef P9_10BC_H
#define P9_10BC_H
#include "p9_10bs.h"

#include <iostream>
using std::ostream;

class Cube {
   friend ostream &operator<<( ostream&, const Cube& );
public:
   Cube( double = 0, double = 0, double = 0, double = 1.0 );
   void print( void ) const;
   double area( void ) const;
   double volume( void ) const;
private:
   Square squareObject;
};

#endif
```

```
//P9_10BMC.cpp
//member function definitions for class Cube
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include "p9_10bc.h"

Cube::Cube( double j, double k, double m, double s )
     : squareObject( j, k, m, s ) { }

ostream &operator<<( ostream &output, const Cube &c )
{
   c.print();
   return output;
}

void Cube::print( void ) const
{
   cout << setiosflags( ios::fixed | ios::showpoint )
        << "The lower left coordinate of the cube is: [" << setprecision( 2 )
        << squareObject.getXCoord() << ", " << setprecision( 2 )
        << squareObject.getYCoord() << ", " << setprecision( 2 )
        << squareObject.getZCoord() << "]\nThe cube side is: "
        << setprecision( 2 ) << squareObject.getSide()
        << "\nThe surface area of the cube is: " << setprecision( 2 ) << area()
       << "\nThe volume of the cube is: " << setprecision( 2 ) << volume() << '\n';
}

```

```
double Cube::area( void ) const  { return 6 * squareObject.area(); }

double Cube::volume( void ) const
   { return squareObject.area() * squareObject.getSide(); }
```

```
// driver for p9_10b.cpp
#include <iostream>

using std::cout;
using std::endl;

#include "p9_10b.h"
#include "p9_10bs.h"
#include "p9_10bc.h"

int main()
{
   Point p( 7.9, 12.5, 8.8 );
   Square s( 0.0, 0.0, 0.0, 5.0 );
   Cube c( 0.5, 8.3, 12.0, 2.0 );

   cout << p << '\n' << s << '\n' << c << endl;

   return 0;
}
```

```
The point is: [7.90, 12.50, 8.80]

The lower left coordinate of the square is: [0.00, 0.00, 0.00]
The square side is: 5
The area of the square is: 25

The lower left coordinate of the cube is: [0.50, 8.30, 12.00]
The cube side is: 2.00
The surface area of the cube is: 24.00
The volume of the cube is: 8.00
```

9.11 In the chapter, we stated, “When a base-class member is inappropriate for a derived class, that member can be overridden in the derived class with an appropriate implementation.” If this is done, does the derived-class-is-a-base-class-object relationship still hold? Explain your answer.

9.12 Study the inheritance hierarchy of Fig. 9.2. For each class, indicate some common attributes and behaviors consistent with the hierarchy. Add some other classes (e.g., **`UndergraduateStudent`**, **`GraduateStudent`**, **`Freshman`**, **`Sophomore`**, **`Junior`**, **`Senior`**, etc.) to enrich the hierarchy.

9.13 Write an inheritance hierarchy for class **`Quadrilateral`**, **`Trapezoid`**, **`Parallelogram`**, **`Rectangle`** and **`Square`**. Use **`Quadrilateral`** as the base class of the hierarchy. Make the hierarchy as deep (i.e., as many levels) as possible. The **`private`** data of **`Quadrilateral`** should be the (*x*, *y*) coordinate pairs for the four endpoints of the **`Quadrilateral`**. Write a driver program that instantiates and displays objects of each of these classes.

```
// P9_13.H
#ifndef P9_13_H
#define P9_13_H

#include <iostream>
using std::ostream;

class Point {
   friend ostream &operator<<( ostream&, const Point& );
public:
   Point( double = 0, double = 0 );
   void setPoint( double, double );
   void print( void ) const;
   double getX( void ) const { return x; }
   double getY( void ) const { return y; }
private:
   double x, y;
};

#endif
```

```
//P9_13PM.cpp
// member function defintions for class Point
#include <iostream>

using std::cout;
using std::ios;
using std::ostream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include "p9_13.h"

Point::Point( double a, double b ) { setPoint( a, b ); }

void Point::setPoint( double a, double b )
{
   x = a;
   y = b;
}

ostream &operator<<( ostream &output, const Point &p )
{
   output << "The point is: ";
   p.print();
   return output;
}

void Point::print( void ) const
{
   cout << setiosflags( ios::fixed | ios::showpoint )
        << '[' << setprecision( 2 ) << getX()
        << ", " << setprecision( 2 ) << getY() << "]\n"
        << resetiosflags( ios::fixed | ios::showpoint );
}
```

```
// P9_13Q.H
#ifndef P9_13Q_H
#define P9_13Q_H
#include "p9_13.h"

#include <iostream>
using std::ostream;

class Quadrilateral {
   friend ostream &operator<<( ostream&, Quadrilateral& );
public:
   Quadrilateral( double = 0, double = 0, double = 0, double = 0, double = 0,
                  double = 0, double = 0, double = 0 );
   void print( void ) const;
protected:
   Point p1;
   Point p2;
   Point p3;
   Point p4;
};

#endif
```

```
// P9_13QM.cpp
// member functions for class Quadrilateral
#include "p9_13q.h"

#include <iostream>
using std::cout;
using std::ostream;

Quadrilateral::Quadrilateral( double x1, double y1, double x2, double y2,
                              double x3, double y3, double x4, double y4 )
   : p1( x1, y1 ), p2( x2, y2 ), p3( x3, y3 ), p4( x4, y4 )  { }

ostream &operator<<( ostream& output, Quadrilateral& q )
{
   output << "Coordinates of Quadrilateral are:\n";
   q.print();
   output << '\n';
   return output;
}

void Quadrilateral::print( void ) const
{
   cout << '(' << p1.getX()
        << ", " << p1.getY() << ") , (" << p2.getX() << ", " << p2.getY()
        << ") , (" << p3.getX() << ", " << p3.getY() << ") , ("
        << p4.getX() << ", " << p4.getY() << ")\n";
}
```

```
// P9_13T.H
#ifndef P9_13T_H
#define P9_13T_H
#include "p9_13q.h"

#include <iostream>
using std::ostream;

class Trapazoid : public Quadrilateral {
   friend ostream& operator<<( ostream&, Trapazoid& );
public:
   Trapazoid( double = 0, double = 0, double = 0, double = 0, double = 0,
              double = 0, double = 0, double = 0, double = 0 );
   void print( void ) const;
   void setHeight( double h ) { height = h; }
   double getHeight( void ) const { return height; }
private:
   double height;
};

#endif
```

```
// P9_13TM.cpp
// member function definitions for class Trapazoid
#include "p9_13t.h"

#include <iostream>

using std::cout;
using std::ostream;


Trapazoid::Trapazoid( double h, double x1, double y1, double x2, double y2,
                      double x3, double y3, double x4, double y4 )
          : Quadrilateral( x1, y1, x2, y2, x3, y3, x4, y4 )
{ setHeight( h ); }

ostream& operator<<( ostream& out, Trapazoid& t )
{
   out << "The Coordinates of the Trapazoid are:\n";
   t.print();
   return out;
}

void Trapazoid::print( void ) const
{
   Quadrilateral::print();
   cout << "Height is : " << getHeight() << "\n\n";
}
```

```
// P9_13PA_H
#ifndef P9_13PA_H
#define P9_13PA_H
#include "p9_13q.h"

#include <iostream>
using std::ostream;

class Parallelogram : public Quadrilateral {
   friend ostream& operator<<( ostream&, Parallelogram& );
public:
   Parallelogram( double = 0, double = 0, double = 0, double = 0,
                  double = 0, double = 0, double = 0, double = 0 );
   void print( void ) const;
private:
   // no private data members
};

#endif
```

```
// P9_13PAM.cpp
#include "p9_13q.h"
#include "p9_13pa.h"

#include <iostream>
using std::ostream;
```

```

Parallelogram::Parallelogram( double x1, double y1, double x2, double y2,
                              double x3, double y3, double x4, double y4 )
                : Quadrilateral( x1, y1, x2, y2, x3, y3, x4, y4 ) { }

ostream& operator<<( ostream& out, Parallelogram& pa )
{
   out << "The coordinates of the Parallelogram are:\n";
   pa.print();
   return out;
}

void Parallelogram::print( void ) const
   {   Quadrilateral::print();   }
```

```
// P9_13R.H
#ifndef P9_13R_H
#define P9_13R_H
#include "p9_13pa.h"

#include <iostream>
using std::ostream;

class Rectangle : public Parallelogram {
   friend ostream& operator<<( ostream&, Rectangle& );
public:
   Rectangle( double = 0, double = 0, double = 0, double = 0,
              double = 0, double = 0, double = 0, double = 0 );
   void print( void ) const;
private:
   // no private data members
};

#endif
```

```
// P9_13RM.cpp
#include "p9_13r.h"
#include "p9_13pa.h"

#include <iostream>
using std::ostream;

Rectangle::Rectangle( double x1, double y1, double x2, double y2,
                      double x3, double y3, double x4, double y4 )
   : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }

ostream& operator<<( ostream& out, Rectangle& r )
{
   out << "\nThe coordinates of the Rectangle are:\n";
   r.print();
   return out;
}

void Rectangle::print( void ) const
   {   Parallelogram::print(); }
```

```
/ P9_13RH.H
#ifndef P9_13RH_H
```

```
#define P9_13RH_H
#include "p9_13pa.h"

#include <iostream>
using std::ostream;

class Rhombus : public Parallelogram {
   friend ostream& operator<<(ostream&, Rhombus&);
public:
   Rhombus( double = 0, double = 0, double = 0, double = 0,
            double = 0, double = 0, double = 0, double = 0 );
   void print( void ) const { Parallelogram::print(); }
private:
   // no private data members
};

#endif
```

```
//P9_13HM.cpp
#include "p9_13rh.h"
#include "p9_13pa.h"

#include <iostream>
using std::ostream;

Rhombus::Rhombus( double x1, double y1, double x2, double y2,
                  double x3, double y3, double x4, double y4 )
   : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }

ostream& operator<<( ostream& out, Rhombus& r )
{
   out << "\nThe coordinates of the Rhombus are:\n";
   r.print();
   return out;
}
```

```
// P9_13S.H
#ifndef P9_13S_H
#define P9_13S_H
#include "p9_13pa.h"

#include <iostream>
using std::ostream;

class Square : public Parallelogram {
   friend ostream& operator<<( ostream&, Square& );
public:
   Square( double = 0, double = 0, double = 0, double = 0,
          double = 0, double = 0, double = 0, double = 0 );
   void print( void ) const { Parallelogram::print(); }
private:
   // no private data members
};

#endif
```

```
// P9_13SM.cpp
#include "p9_13s.h"
```

```
#include "p9_13pa.h"

#include <iostream>
using std::ostream;

Square::Square( double x1, double y1, double x2, double y2,
                double x3, double y3, double x4, double y4 )
   : Parallelogram( x1, y1, x2, y2, x3, y3, x4, y4 ) { }

ostream& operator<<( ostream& out, Square& s )
{
   out << "\nThe coordinates of the Square are:\n";
   s.print();
   return out;
}
```

```
// P9_13.cpp
#include "p9_13.h"
#include "p9_13q.h"
#include "p9_13t.h"
#include "p9_13pa.h"
#include "p9_13rh.h"
#include "p9_13r.h"
#include "p9_13s.h"

#include <iostream>
using std::cout;
using std::endl;

int main()
{
   // NOTE: All coordinates are assumed to form the proper shapes

   // A quadrilateral is a four-sided polygon
   Quadrilateral q( 1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4 );
   // A trapazoid is a quadrilateral having two and only two parallel sides
   Trapazoid t( 5.0, 0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0 );
   // A parallelogram is a quadrilateral whose opposite sides are parallel
   Parallelogram p( 5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0 );
   // A rhombus is an equilateral parallelogram
   Rhombus rh( 0.0, 0.0, 5.0, 0.0, 8.5, 3.5, 3.5, 3.5 );
   // A rectangle is an equiangular parallelogram
   Rectangle r( 17.0, 14.0, 30.0, 14.0, 30.0, 28.0, 17.0, 28.0 );
   // A square is an equiangular and equilateral parallelogram
   Square s( 4.0, 0.0, 8.0, 0.0, 8.0, 4.0, 4.0, 4.0 );

   cout << q << t << p << rh << r << s << endl;

   return 0;
}
```

```
Coordinates of Quadrilateral are:
(1.1, 1.2) , (6.6, 2.8) , (6.2, 9.9) , (2.2, 7.4)

The Coordinates of the Trapazoid are:
(0, 0) , (10, 0) , (8, 5) , (3.3, 5)
Height is : 5

The coordinates of the Parallelogram are:
(5, 5) , (11, 5) , (12, 20) , (6, 20)

The coordinates of the Rhombus are:
(0, 0) , (5, 0) , (8.5, 3.5) , (3.5, 3.5)

The coordinates of the Rectangle are:
(17, 14) , (30, 14) , (30, 28) , (17, 28)

The coordinates of the Square are:
(4, 0) , (8, 0) , (8, 4) , (4, 4)
```

9.14 Write down all the shapes you can think of—both two-dimensional and three-dimensional—and form those shapes into a shape hierarchy. Your hierarchy should have base class **`Shape`** from which class **`TwoDimensionalShape`** and class **`ThreeDimensionalShape`** are derived. Once you have developed the hierarchy, define each of the classes in the hierarchy. We will use this hierarchy in the exercises of Chapter 10 to process all shapes as objects of base-class **`Shape`**. This is a technique called polymorphism.

ANS: Shape

```
  TwoDimensionalShape
    Quadrilateral
      Parallelogram
        Rectangle
          Square
        Rhombus
    Ellipse
      Circle
    Triangle
      RightTriangle
      EquilateralTriangle
      IsocelesTriangle
    Parabola
    Line
    Hyperbola
  ThreeDimensionalShape
    Ellipsoid
      Sphere
    Prism
    Cylinder
    Cone
    Cube
    Tetrahedron
    Hyperboloid
      OneSheetHyperboloid
      TwoSheetHyperboloid
    Plane
```

10

Virtual Functions and Polymorphism Solutions

Solutions

10.2 What are **`virtual`** functions? Describe a circumstance in which **`virtual`** functions would be appropriate.

ANS: Virtual functions are functions with the same function prototype that are defined throughout a class hierarchy. At least the base class occurrence of the function is preceded by the keyword **`virtual`**. Virtual functions are used to enable generic processing of an entire class hierarchy of objects through a base class pointer. For example, in a shape hierarchy, all shapes can be drawn. If all shapes are derived from a base class **`Shape`** which contains a **`virtual draw`** function, then generic processing of the hierarchy can be performed by calling every shape's **`draw`** generically through a base class **`Shape`** pointer.

10.3 Given that constructors cannot be **`virtual`**, describe a scheme for how you might achieve a similar effect.

ANS: Create a virtual function called **`initialize`** that the constructor invokes.

10.4 How is it that polymorphism enables you to program "in the general" rather than "in the specific." Discuss the key advantages of programming "in the general."

ANS: Polymorphism enables the programmer to concentrate on the processing of common operations that are applied to all data types in the system without going into the individual details of each data type. The general processing capabilities are separated from the internal details of each type.

10.5 Discuss the problems of programming with **`switch`** logic. Explain why polymorphism is an effective alternative to using **`switch`** logic.

ANS: The main problem with programming using the **`switch`** structure is extensibility and maintainability of the program. A program containing many **`switch`** structures is difficult to modify. Many, but not necessarily all, **`switch`** structures will need to add or remove cases for a specified type. Note: switch logic includes if/else structures which are more flexible than the **`switch`** structure.

10.6 Distinguish between static binding and dynamic binding. Explain the use of **`virtual`** functions and the vtable in dynamic binding.

ANS: Static binding is performed at compile-time when a function is called via a specific object or via a pointer to an object. Dynamic binding is performed at run-time when a **`virtual`** function is called via a base class pointer to a derived class object (the object can be of any derived class). The **`virtual`** functions table (vtable) is used at run-time to enable the proper function to be called for the object to which the base class pointer "points". Each class containing **`virtual`** functions has its own vtable that specifies where the **`virtual`** functions for that class are located. Every object of a class with **`virtual`** functions contains a hidden pointer to the class's vtable. When a **`virtual`** function is called via a base class pointer, the hidden pointer is dereferenced to locate the vtable, then the vtable is searched for the proper function call

10.7 Distinguish between inheriting interface and inheriting implementation. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation?

ANS: When a class inherits implementation, it inherits previously defined functionality from another class. When a class inherits interface, it inherits the definition of what the interface to the new class type should be. The implementation is then provided by the programmer defining the new class type. Inheritance hierarchies designed for inheriting implementation are used to reduce the amont of new code that is being written. Such hierarchies are used to facilitate software reusability. Inheritance hierarchies designed for inheriting interface are used to write programs that perform generic processing of many class types. Such hierarchies are commonly used to facilitate software extensibility (i.e., new types can be added to the hierarchy without changing the generic processing capabilitiesof the program.)

10.8 Distinguish between `virtual` functions and pure `virtual` functions.

ANS: A `virtual` function must have a definition in the class in which it is declared. A pure `virtual` function does not provide a definition. Classes derived directly from the abstract class must provide definitions for the inherited pure `virtual` functions in order to avoid becoming an abstract base class.

10.9 (True/False) All `virtual` functions in an abstract base class must be declared as pure `virtual` functions.

ANS: False.

10.10 Suggest one or more levels of abstract base classes for the `Shape` hierarchy discussed in this chapter (the first level is `Shape` and the second level consists of the classes `TwoDimensionalShape` and `ThreeDimensionalShape`).

10.11 How does polymorphism promote extensibility?

ANS: Polymorphism makes programs more extensible by making all function calls generic. When a new class type with the appropriate `virtual` functions is added to the hierarchy, no changes need to be made to the generic function calls.

10.12 You have been asked to develop a flight simulator that will have elaborate graphical outputs. Explain why polymorphic programming would be especially effective for a problem of this nature.

10.13 Develop a basic graphics package. Use the `Shape` class inheritance hierarchy from Chapter 9. Limit yourself to two-dimensional shapes such as squares, rectangles, triangles and circles. Interact with the user. Let the user specify the position, size, shape and fill characters to be used in drawing each shape. The user can specify many items of the same shape. As you create each shape, place a `Shape *` pointer to each new `Shape` object into an array. Each class has its own `draw` member function. Write a polymorphic screen manager that walks through the array (preferably using an iterator) sending `draw` messages to each object in the array to form a screen image. Redraw the screen image each time the user specifies an additional shape.

10.14 Modify the payroll system of Fig. 10.1 to add private data members `birthDate` (a `Date` object) and `departmentCode` (an `int`) to class `Employee`. Assume this payroll is processed once per month. Then, as your program calculates the payroll for each `Employee` (polymorphically), add a $100.00 bonus to the person's payroll amount if this is the month in which the `Employee`'s birthday occurs.

```
// EMPLOY.H
// Abstract base class Employee
#ifndef EMPLOY_H
#define EMPLOY_H

#include "date.h"

class Employee {
public:
   Employee( const char * const, const char * const,
             int, int, int, int);
   ~Employee();
   const char *getFirstName() const;
   const char *getLastName() const;
   Date getBirthDate() const;
   int getDepartmentCode() const;

   // Pure virtual functions make Employee abstract base class.
   virtual double earnings() const = 0; // pure virtual
   virtual void print() const = 0;     // pure virtual
private:
   char *firstName;
   char *lastName;
   Date birthDate;
   int departmentCode;
};

#endif
```

```
// EMPLOY.CPP
// Member function definitions for
// abstract base class Employee.
//
// Note: No definitions given for pure virtual functions.

#include <cstring>

#include <assert.h>
#include "employ.h"

// Constructor dynamically allocates space for the
// first and last name and uses strcpy to copy
// the first and last names into the object.
Employee::Employee( const char * const first, const char * const last,
                    int mn, int dy, int yr, int dept )
   birthDate( mn, dy, yr ), departmentCode( dept )
{
   firstName = new char[ strlen( first ) + 1 ];
   assert( firstName != 0 );    // test that new worked
   strcpy( firstName, first );

   lastName = new char[ strlen( last ) + 1 ];
   assert( lastName != 0 );     // test that new worked
   strcpy( lastName, last );
}

// Destructor deallocates dynamically allocated memory
Employee::~Employee()
{
   delete [] firstName;
   delete [] lastName;
}

// Return a pointer to the first name
const char *Employee::getFirstName() const
{
   // Const prevents caller from modifying private data.
   // Caller should copy returned string before destructor
   // deletes dynamic storage to prevent undefined pointer.

   return firstName;   // caller must delete memory
}

// Return a pointer to the last name
const char *Employee::getLastName() const
{
   // Const prevents caller from modifying private data.
   // Caller should copy returned string before destructor
   // deletes dynamic storage to prevent undefined pointer.

   return lastName;   // caller must delete memory
}

// Return the employee's birth date
```

```
Date Employee::getBirthDate() const { return birthDate; }

// Return the employee's department code
int Employee::getDepartmentCode() const { return departmentCode; }
```

```
// BOSS.H
// Boss class derived from Employee
#ifndef BOSS_H
#define BOSS_H
#include "employ.h"

class Boss : public Employee {
public:
   Boss( const char * const, const char * const, int, int, int,
         double = 0.0, int = 0 );
   void setWeeklySalary( double );
   virtual double earnings() const;
   virtual void print() const;
private:
   double weeklySalary;
};

#endif
```

```
// BOSS.CPP
// Member function definitions for class Boss
#include <iostream>

using std::cout;

#include "boss.h"

// Constructor function for class Boss
Boss::Boss( const char * const first, const char * const last,
            int mn, int dy, int yr, double s, int dept )
   : Employee( first, last, mn, dy, yr, dept )
{ weeklySalary = s > 0 ? s : 0; }

// Set the Boss's salary
void Boss::setWeeklySalary( double s )
   { weeklySalary = s > 0 ? s : 0; }

// Get the Boss's pay
double Boss::earnings() const { return weeklySalary; }

// Print the Boss's name
void Boss::print() const
{
   cout << "\nBoss: " << getFirstName()
        << ' ' << getLastName();
}
```

```
// HOURLY.H
// Definition of class HourlyWorker
#ifndef HOURLY_H
#define HOURLY_H
#include "employ.h"

```

```
class HourlyWorker : public Employee {
public:
   HourlyWorker( const char * const, const char * const, int, int, int,
                 double = 0.0, double = 0.0, int = 0 );
   void setWage( double );
   void setHours( double );
   virtual double earnings() const;
   virtual void print() const;
private:
   double wage;   // wage per hour
   double hours;  // hours worked for week
};

#endif
```

```
// HOURLY.CPP
// Member function definitions for class HourlyWorker
#include <iostream>

using std::cout;

#include "hourly.h"

// Constructor for class HourlyWorker
HourlyWorker::HourlyWorker( const char * const first, const char * const last,
                            int mn, int dy, int yr, double w, double h, int dept )
      : Employee( first, last, mn, dy, yr, dept )
{
   wage = w > 0 ? w : 0;
   hours = h >= 0 && h < 168 ? h : 0;
}

// Set the wage
void HourlyWorker::setWage( double w ) { wage = w > 0 ? w : 0; }

// Set the hours worked
void HourlyWorker::setHours( double h )
   { hours = h >= 0 && h < 168 ? h : 0; }

// Get the HourlyWorker's pay
double HourlyWorker::earnings() const { return wage * hours; }

// Print the HourlyWorker's name
void HourlyWorker::print() const
{
   cout << "\nHourly worker: " << getFirstName()
        << ' ' << getLastName();
}
```

```
// PIECE.H
// PieceWorker class derived from Employee
#ifndef PIECE_H
#define PIECE_H
#include "employ.h"

class PieceWorker : public Employee {
public:
   PieceWorker( const char * const, const char * const, int, int, int,
                double = 0.0, unsigned = 0, int = 0 );
   void setWage( double );
   void setQuantity( unsigned );
   virtual double earnings() const;
   virtual void print() const;
private:
   double wagePerPiece; // wage for each piece output
   unsigned quantity;  // output for week
};

#endif
```

```
// PIECE.CPP
// Member function definitions for class PieceWorker
#include <iostream>

using std::cout;

#include "piece.h"

// Constructor for class PieceWorker
PieceWorker::PieceWorker( const char * const first, const char * const last,
    int mn, int dy, int yr, double w, unsigned q, int dept )
    : Employee( first, last, mn, dy, yr, dept )
{
   wagePerPiece = w > 0 ? w : 0;
   quantity = q > 0 ? q : 0;
}

// Set the wage
void PieceWorker::setWage( double w )
   { wagePerPiece = w > 0 ? w : 0; }

// Set the number of items output
void PieceWorker::setQuantity( unsigned q )
   { quantity = q > 0 ? q : 0; }

// Determine the PieceWorker's earnings
double PieceWorker::earnings() const
   { return quantity * wagePerPiece; }

// Print the PieceWorker's name
void PieceWorker::print() const
{
   cout << "\nPiece worker: " << getFirstName()
        << ' ' << getLastName();
}
```

```
// COMMIS.H
// CommissionWorker class derived from Employee
#ifndef COMMIS_H
#define COMMIS_H
#include "employ.h"

class CommissionWorker : public Employee {
public:
   CommissionWorker(const char * const, const char * const, int, int, int,
                    double = 0.0, double = 0.0, unsigned = 0, int = 0 );
   void setSalary( double );
   void setCommission( double );
   void setQuantity( unsigned );
   virtual double earnings() const;
   virtual void print() const;
private:
   double salary;       // base salary per week
   double commission;   // amount per item sold
   unsigned quantity;  // total items sold for week
};

#endif
```

```
// COMMIS.CPP
// Member function definitions for class CommissionWorker
#include <iostream>

using std::cout;

#include "commis.h"

// Constructor for class CommissionWorker
CommissionWorker::CommissionWorker( const char * const first,
   const char * const last, int mn, int dy, int yr,
   double s, double c, unsigned q, int dept )
   : Employee( first, last, mn, dy, yr, dept )
{
   salary = s > 0 ? s : 0;
   commission = c > 0 ? c : 0;
   quantity = q > 0 ? q : 0;
}

// Set CommissionWorker's weekly base salary
void CommissionWorker::setSalary( double s )
   { salary = s > 0 ? s : 0; }

// Set CommissionWorker's commission
void CommissionWorker::setCommission( double c )
   { commission = c > 0 ? c : 0; }

// Set CommissionWorker's quantity sold
void CommissionWorker::setQuantity( unsigned q )
   { quantity = q > 0 ? q : 0; }

// Determine CommissionWorker's earnings
double CommissionWorker::earnings() const
   { return salary + commission * quantity; }

// Print the CommissionWorker's name
void CommissionWorker::print() const
{
   cout << "\nCommission worker: " << getFirstName()
        << ' ' << getLastName();
}
```

```
// DATE.H
// Declaration of the Date class.
// Member functions defined in DATE1.CPP
#ifndef DATE_H
#define DATE_H

```

```
class Date {
public:
   Date( int = 1, int = 1, int = 1900 );  // default constructor
   int getMonth() const; // return the month
   int getDay() const;   // return the day
   int getYear()const;   // return the year
   void print() const;   // print date in month/day/year format
private:
   int month;  // 1-12
   int day;    // 1-31 based on month
   int year;   // any year

   // utility function to test proper day for month and year
   int checkDay( int ) const;
};

#endif
```

```
// DATE.CPP
// Member function definitions for Date class.
#include <iostream>

using std::cout;

#include "date.h"

// Constructor: Confirm proper value for month;
// call utility function checkDay to confirm proper
// value for day.
Date::Date( int mn, int dy, int yr )
{
   if ( mn > 0 && mn <= 12 )         // validate the month
      month = mn;
   else {
      month = 1;
      cout << "Month " << mn << " invalid. Set to month 1.\n";
   }

   year = yr >= 1900 && yr <= 2100 ? yr : 1990;
   day = checkDay( dy );             // validate the day
}

// Utility function to confirm proper day value
// based on month and year.
int Date::checkDay( int testDay ) const
{
   int daysPerMonth[ 13 ] = { 0, 31, 28, 31, 30, 31, 30,
                              31, 31, 30, 31, 30, 31 };

   if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
      return testDay;

   if ( month == 2 &&     // February: Check for possible leap year
        testDay == 29 &&
        ( year % 400 == 0 || (year % 4 == 0 && year % 100 != 0 ) ) )
      return testDay;

   cout << "Day " << testDay << " invalid. Set to day 1.\n";

   return 1;  // leave object in consistent state if bad value
}
```

```

// Return the month
int Date::getMonth() const { return month; }

// Return the day
int Date::getDay() const { return day; }

// Return the year
int Date::getYear() const { return year; }

// Print Date object in form  month/day/year
void Date::print() const
   { cout << month << '/' << day << '/' << year; }
```

```
// Exercise 10.14 solution
// Driver for Employee hierarchy
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <ctime>
#include <cstdlib>
#include "employ.h"
#include "boss.h"
#include "commis.h"
#include "piece.h"
#include "hourly.h"

int determineMonth();

int main()
{
// set output formatting
   cout << setiosflags( ios::fixed | ios::showpoint )
             << setprecision( 2 );

   Boss b( "John", "Smith", 6, 15, 1944, 800.00, 1 );
   CommissionWorker c( "Sue", "Jones", 9, 8, 1954, 200.0, 3.0, 150, 1 );
   PieceWorker p( "Bob", "Lewis", 3, 2, 1965, 2.5, 200, 1 );
   HourlyWorker h( "Karen", "Price", 12, 29, 1960, 13.75, 40, 1 );

   Employee *ptr[ 4 ] = { &b, &c, &p, &h };

   int month = determineMonth();

   cout << "The month is " << month << "\nThe payroll is:\n";

   for ( int x = 0; x < 4; ++x ) {
   ptr[ x ]->print();
   cout << " of department " << ptr[ x ]->getDepartmentCode()
           << "\n whose birthday is ";
   ptr[ x ]->getBirthDate().print();
   cout << " earned ";

   if ( ptr[ x ]->getBirthDate().getMonth() == month )
       cout << ptr[ x ]->earnings() + 100.0
                        << " HAPPY BIRTHDAY!\n";
   else
      cout << ptr[x]->earnings() << endl;


   return 0;
}

// Determine the current month using standard library functions
// of ctime.
int determineMonth()
{
   time_t currentTime;
   char monthString[ 3 ];

   time( &currentTime );
           strftime( monthString, 3, "%m", localtime( &currentTime ) );
return atoi( monthString );
}
```

```
The month is 5
The payroll is:

Boss: John Smith of department 1
 whose birthday is 6/15/1944 earned 800.00

Commission worker: Sue Jones of department 1
 whose birthday is 9/8/1954 earned 650.00

Piece worker: Bob Lewis of department 1
 whose birthday is 3/2/1965 earned 500.00

Hourly worker: Karen Price of department 1
 whose birthday is 12/29/1960 earned 550.00
```

10.15 In Exercise 9.14, you developed a **Shape** class hierarchy and defined the classes in the hierarchy. Modify the hierarchy so that class **Shape** is an abstract base class containing the interface to the hierarchy. Derive **TwoDimensionalShape** and **ThreeDimensionalShape** from class **Shape**—these classes should also be abstract. Use a **virtual print** function to output the type and dimensions of each class. Also include **virtual area** and **volume** functions so these calculations can be performed for objects of each concrete class in the hierarchy. Write a driver program that tests the **Shape** class hierarchy.

```
// SHAPE.H
// Definition of base-class Shape
#ifndef SHAPE_H
#define SHAPE_H

#include <iostream>
using std::ostream;

class Shape {
   friend ostream & operator<<( ostream &, Shape & );
public:
   Shape( double = 0, double = 0 );
   double getCenterX() const;
   double getCenterY() const;
   virtual void print() const = 0;
protected:
   double xCenter;
   double yCenter;
};

#endif
```

```
// SHAPE.CPP
// Member and friend definitions for Shape
#include "shape.h"

Shape::Shape( double x, double y )
{
   xCenter = x;
   yCenter = y;
}

double Shape::getCenterX() const { return xCenter; }

double Shape::getCenterY() const { return yCenter; }

ostream & operator<<( ostream &out, Shape &s )
{
   s.print();
   return out;
}
```

```
// TWODIM.H
// Defnition of class TwoDimensionalShape
#ifndef TWODIM_H
#define TWODIM_H

#include "shape.h"

class TwoDimensionalShape : public Shape {
public:
   TwoDimensionalShape( double x, double y ) : Shape( x, y ) { }
   virtual double area() const = 0;
};

#endif
```

```
// THREEDIM.H
// Defnition of class ThreeDimensionalShape
#ifndef THREEDIM_H
#define THREEDIM_H

#include "shape.h"

class ThreeDimensionalShape : public Shape {
public:
   ThreeDimensionalShape( double x, double y ) : Shape( x, y ) { }
   virtual double area() const = 0;
   virtual double volume() const = 0;
};

#endif
```

```
// CIRCLE.H
// Definition of class Circle
#ifndef CIRCLE_H
#define CIRCLE_H

#include "twodim.h"

class Circle : public TwoDimensionalShape {
public:
   Circle( double = 0, double = 0, double = 0 );
   double getRadius() const;
   double area() const;
   void print() const;
private:
   double radius;
};

#endif
```

```
// CIRCLE.CPP
// Member function definitions for Circle
#include "circle.h"

#include <iostream>
using std::cout;

Circle::Circle( double r, double x, double y )
   : TwoDimensionalShape( x, y ) { radius = r > 0 ? r : 0; }

double Circle::getRadius() const { return radius; }

double Circle::area() const { return 3.14159 * radius * radius; }

void Circle::print() const
{
   cout << "Circle with radius " << radius << "; center at ("
        << xCenter << ", " << yCenter << ");\narea of " << area() << '\n';
}
```

```
// SQUARE.H
// Definition of class Square
#ifndef SQUARE_H
#define SQUARE_H

#include "twodim.h"

class Square : public TwoDimensionalShape {
public:
   Square( double = 0, double = 0, double = 0 );
   double getSideLength() const;
   double area() const;
   void print() const;
private:
   double sideLength;
};

#endif
```

```
// SQUARE.CPP
// Member function definitions for Square
#include "square.h"

#include <iostream>
using std::cout;

Square::Square( double s, double x, double y )
   : TwoDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }

double Square::getSideLength() const { return sideLength; }

double Square::area() const { return sideLength * sideLength; }

void Square::print() const
{
   cout << "Square with side length " << sideLength << "; center at ("
        << xCenter << ", " << yCenter << ");\narea of " << area() << '\n';
}
```

```
// SPHERE.H
// Definition of class Shere
#ifndef SPHERE_H
#define SPHERE_H

#include "threedim.h"

class Sphere : public ThreeDimensionalShape {
public:
   Sphere( double = 0, double = 0, double = 0 );
   double area() const;
   double volume() const;
   double getRadius() const;
   void print() const;
private:
   double radius;
};

#endif
```

```
// SPHERE.CPP
// Member function definitions for Sphere
#include "sphere.h"

#include <iostream>
using std::cout;

Sphere::Sphere( double r, double x, double y )
   : ThreeDimensionalShape( x, y ) { radius = r > 0 ? r : 0; }

double Sphere::area() const
   { return 4.0 * 3.14159 * radius * radius; }

double Sphere::volume() const
   { return 4.0/3.0 * 3.14159 * radius * radius * radius; }

double Sphere::getRadius() const { return radius; }

void Sphere::print() const
```

```
{
   cout << "Sphere with radius " << radius << "; center at ("
        << xCenter << ", " << yCenter << ");\narea of "
        << area() << "; volume of " << volume() << '\n';
}
```

```
// CUBE.H
// Definition of class Cube
#ifndef CUBE_H
#define CUBE_H

#include "threedim.h"

class Cube : public ThreeDimensionalShape {
public:
   Cube( double = 0, double = 0, double = 0 );
   double area() const;
   double volume() const;
   double getSideLength() const;
   void print() const;
private:
   double sideLength;
};

#endif
```

```
// CUBE.CPP
// Member function definitions for Cube
#include "cube.h"

#include <iostream>
using std::cout;

Cube::Cube( double s, double x, double y )
   : ThreeDimensionalShape( x, y ) { sideLength = s > 0 ? s : 0; }

double Cube::area() const { return 6 * sideLength * sideLength; }

double Cube::volume() const
   { return sideLength * sideLength * sideLength; }

double Cube::getSideLength() const { return sideLength; }

void Cube::print() const
{
   cout << "Cube with side length " << sideLength << "; center at ("
        << xCenter << ", " << yCenter << ");\narea of "
        << area() << "; volume of " << volume() << '\n';
}
```

```
// Exercise 10.15 solution
// Driver to test Shape hierarchy
#include <iostream>

using std::cout;

#include "circle.h"
```

```
#include "square.h"
#include "sphere.h"
#include "cube.h"

int main()
{
   Circle cir( 3.5, 6, 9 );
   Square sqr( 12, 2, 2 );
   Sphere sph( 5, 1.5, 4.5 );
   Cube cub( 2.2 );
   Shape *ptr[ 4 ] = { &cir, &sqr, &sph, &cub };

   for ( int x = 0; x < 4; ++x )
      cout << *( ptr[ x ] ) << '\n';

   return 0;
}
```

```
Circle with radius 3.5; center at (6, 9);
area of 38.4845

Square with side length 12; center at (2, 2);
area of 144

Sphere with radius 5; center at (1.5, 4.5);
area of 314.159; volume of 523.598

Cube with side length 2.2; center at (0, 0);
area of 29.04; volume of 10.648
```

11

C++ Stream Input/Output Solutions

Solutions

11.6 Write a statement for each of the following:

a) Print integer **`40000`** left-justified in a **`15`**-digit field.

ANS: **`cout << setiosflags( ios::left ) << setw( 15 ) << 40000 << '\n';`**

b) Read a string into character array variable **`state`**.

ANS: **`cin >> state;`**

c) Print **`200`** with and without a sign.

ANS:

```
cout << setiosflags( ios::showpos ) << 200 << setw( 4 )  << '\n'
          << resetiosflags( ios::showpos ) << 200 << '\n';
```

d) Print the decimal value **`100`** in hexadecimal form preceded by **`0x`**.

ANS: **`cout << setiosflags( ios::showbase ) << hex << 100 << '\n';`**

e) Read characters into array **`s`** until the character **`'p'`** is encountered up to a limit of 10 characters (including the terminating null character). Extract the delimiter from the input stream and discard it.

ANS: **`cin.getline( s, 10, 'p' );`**

f) Print **`1.234`** in a **`9`**-digit field with preceding zeros.

ANS:

```
cout << setiosflags( ios::fixed | ios::showpoint ) << setw( 9 )
     << setfill( '0' ) << setiosflags( ios::internal ) << 1.234 << '\n';
```

g) Read a string of the form **`"characters"`** from the standard input. Store the string in character array **`s`**. Eliminate the quotation marks from the input stream. Read a maximum of 50 characters (including the terminating null character).

11.7 Write a program to test inputting integer values in decimal, octal and hexadecimal format. Output each integer read by the program in all three formats. Test the program with the following input data: 10, 010, 0x10.

```
// Exercise 11.7 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

```

```
#include <iomanip>

using std::setiosflags;
using std::hex;
using std::oct;
using std::dec;

int main()
{
   int integer;

   cout << "Enter an integer: ";
   cin >> integer;

   cout << setiosflags( ios::showbase ) << "As a decimal number "  << dec
        << integer << "\nAs an octal number " << oct << integer
        << "\nAs a hexadecimal number " << hex << integer << endl;

   cout << "\nEnter an integer in hexadecimal format\n";
   cin >> hex >> integer;

   cout << setiosflags( ios::showbase ) << "As a decimal number "  << dec
        << integer << "\nAs an octal number " << oct << integer
        << "\nAs a hexadecimal number " << hex << integer << endl;

   cout << "\nEnter an integer in octal format\n";
   cin >> oct >> integer;

   cout << setiosflags( ios::showbase ) << "As a decimal number "  << dec
        << integer << "\nAs an octal number " << oct << integer
        << "\nAs a hexadecimal number " << hex << integer << endl;


   return 0;
}
```

```
Enter an integer: 10
As a decimal number 10
As an octal number 012
As a hexadecimal number 0xa

Enter an integer in hexadecimal format
0x10
As a decimal number 16
As an octal number 020
As a hexadecimal number 0x10

Enter an integer in octal format
010
As a decimal number 8
As an octal number 010
As a hexadecimal number 0x8
```

11.8 Write a program that prints pointer values using casts to all the integer data types. Which ones print strange values? Which ones cause errors?

```
// Exercise 11.8 Solution
#include <iostream>

using std::cout;
using std::endl;

int main()
{
   char *string = "test";

   cout << "Value of string is             : " << string << '\n'
        << "Value of static_cast<void *>( string ) is    : "
        << static_cast<void *>( string ) << '\n'

        // The Following generate errors.
        // reinterpret_cast will allow this
        // type of casting. See Chap. 21 for
        // a discussion of reinterpret_cast.

/*      << "Value of static_cast<char>(string) is      : "
        << static_cast<char>( string ) << '\n'
        << "Value of static_cast<int>(string) is       : "
        << static_cast<int>( string ) << '\n'
        << "Value of static_cast<long>(string) is      : "
        << static_cast<long>( string ) << '\n'
        << "Value of static_cast<short>(string) is     : "
        << static_cast<short>( string ) << '\n'
        << "Value of static_cast<unsigned>(string) is  : "
        << static_cast<unsigned>( string )
*/
        << endl;

   return 0;
}
```

```
Value of string is             : test
Value of static_cast<void *>(string) is    : 0046C07C
```

11.9 Write a program to test the results of printing the integer value **12345** and the floating-point value **1.2345** in various-size fields. What happens when the values are printed in fields containing fewer digits than the values?

```
// Exercise 11.9 Solution
#include <iostream>

using std::cout;

#include <iomanip>

using std::setw;

int main()
{
   int x = 12345;
   double y = 1.2345;

   for ( int loop = 0; loop <= 10; ++loop )
```

```
      cout << x << "  printed in a field of size " << loop << " is "
           << setw( loop ) << x << '\n' << y << " printed in a field "
           << "of size " << loop << " is " << setw( loop ) << y << '\n';

   return 0;
}
```

```
12345  printed in a field of size 0 is 12345
1.2345 printed in a field of size 0 is 1.2345
12345  printed in a field of size 1 is 12345
1.2345 printed in a field of size 1 is 1.2345
12345  printed in a field of size 2 is 12345
1.2345 printed in a field of size 2 is 1.2345
12345  printed in a field of size 3 is 12345
1.2345 printed in a field of size 3 is 1.2345
12345  printed in a field of size 4 is 12345
1.2345 printed in a field of size 4 is 1.2345
12345  printed in a field of size 5 is 12345
1.2345 printed in a field of size 5 is 1.2345
12345  printed in a field of size 6 is  12345
1.2345 printed in a field of size 6 is 1.2345
12345  printed in a field of size 7 is   12345
1.2345 printed in a field of size 7 is  1.2345
12345  printed in a field of size 8 is    12345
1.2345 printed in a field of size 8 is   1.2345
12345  printed in a field of size 9 is     12345
1.2345 printed in a field of size 9 is    1.2345
12345  printed in a field of size 10 is      12345
1.2345 printed in a field of size 10 is     1.2345
```

11.10 Write a program that prints the value **100.453627** rounded to the nearest digit, tenth, hundredth, thousandth and ten thousandth.

```
// Exercise 11.10 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

int main()
{
   double x = 100.453627;

   cout << setiosflags( ios::fixed );
   for ( int loop = 0; loop <= 5; ++loop )
       cout << setprecision( loop ) << "Rounded to " << loop
            << " digit(s) is " << x << endl;

   return 0;
}
```

```
Rounded to 0 digit(s) is 100
Rounded to 1 digit(s) is 100.5
Rounded to 2 digit(s) is 100.45
Rounded to 3 digit(s) is 100.454
Rounded to 4 digit(s) is 100.4536
Rounded to 5 digit(s) is 100.45363
```

11.11 Write a program that inputs a string from the keyboard and determines the length of the string. Print the string using twice the length as the field width

```
// Exercise 11.11 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;

#include <cstring>

const int SIZE = 80;

int main()
{
   char string[ SIZE ];
   int stringLength;

   cout << "Enter a string: ";
   cin >> string;

   stringLength = strlen( string );

   cout << "the length of the string is " << strlen( string ) << endl;

   // print string using twice the length as field with
   cout << setw( 2 * stringLength ) << string << endl;

   return 0;
}
```

```
Enter a string: castle
the length of the string is 6
     castle
```

11.12 Write a program that converts integer Fahrenheit temperatures from **0** to **212** degrees to floating-point Celsius temperatures with **3** digits of precision. Use the formula

```
celsius = 5.0 / 9.0 * ( fahrenheit - 32 );
```

to perform the calculation. The output should be printed in two right-justified columns and the Celsius temperatures should be preceded by a sign for both positive and negative values.

```
// Exercise 11.12 Solution
#include <iostream>

using std::cout;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

int main()
{
   double celsius;

   cout << setw( 20 ) << "Fahrenheit " << setw( 20 ) << "Celsius\n"
        << setiosflags( ios::fixed | ios::showpoint );

   for ( int fahrenheit = 0; fahrenheit <= 212; ++fahrenheit ) {
       celsius = 5.0 / 9.0 * ( fahrenheit - 32 );
       cout << setw( 15 ) << resetiosflags( ios::showpos ) << fahrenheit
            << setw( 23 ) << setprecision( 3 ) << setiosflags( ios::showpos )
            << celsius << '\n';
   }

   return 0;
}
```

```
          Fahrenheit             Celsius
              0                 -17.778
              1                 -17.222
              2                 -16.667
              3                 -16.111
              4                 -15.556
...
            206                 +96.667
            207                 +97.222
            208                 +97.778
            209                 +98.333
            210                 +98.889
            211                 +99.444
            212                +100.000
```

11.13 In some programming languages, strings are entered surrounded by either single or double quotation marks. Write a program that reads the three strings **suzy**, **"suzy"** and **'suzy'**. Are the single and double quotes ignored or read as part of the string?

```
// Exercise 11.13 Solution
#include <iostream>

using std::cout;
using std::cin;

const int SIZE = 80;

int main()
```

```
{
   char string[ SIZE ];

   for ( int k = 0; k < 3; ++k ) {
      cout << "Enter a string: ";
      cin >> string;
      cout << "String is " << string << '\n';
   }

   return 0;
}
```

```
Enter a string: "vacuum"
String is "vacuum"
Enter a string: 'grape'
String is 'grape'
Enter a string: water
String is water
```

11.14 In Fig. 8.3, the stream-extraction and -insertion operators were overloaded for input and output of objects of the **PhoneNumber** class. Rewrite the stream-extraction operator to perform the following error checking on input. The **operator>>** function will need to be entirely recoded.

a) Input the entire phone number into an array. Test that the proper number of characters has been entered. There should be a total of 14 characters read for a phone number of the form **(800) 555-1212**. Use the stream member function **clear** to set **ios::failbit** for improper input.

b) The area code and exchange do not begin with **0** or **1**. Test the first digit of the area code and exchange portions of the phone number to be sure that neither begins with **0** or **1**. Use stream member function **clear** to set **ios::failbit** for improper input.

c) The middle digit of an area code used to always be **0** or **1** (although this has changed recently). Test the middle digit for a value of **0** or **1**. Use the stream member function **clear** to set **ios::failbit** for improper input. If none of the above operations results in **ios::failbit** being set for improper input, copy the three parts of the telephone number into the **areaCode**, **exchange** and **line** members of the **PhoneNumber** object. In the main program, if **ios::failbit** has been set on the input, have the program print an error message and end rather than print the phone number.

```
// P11_14.H
#ifndef P11_14_H
#define P11_14_H
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
using std::ios;
using std::ostream;
using std::istream;
using std::cerr;

#include <cstring>
#include <cstdlib>

class PhoneNumber {
   friend ostream& operator<<( ostream&, PhoneNumber& );
   friend istream& operator>>( istream&, PhoneNumber& );
public:
   PhoneNumber();
private:
   char phone[ 15 ];
   char areaCode[ 4 ];
   char exchange[ 4 ];
   char line[ 5 ];
};

#endif
```

```
// P11_14M.cpp
// member function definition definition for p11_14.cpp
#include "p11_14.h"

PhoneNumber::PhoneNumber()
{
   phone[ 0 ] = '\0';
   areaCode[ 0 ] = '\0';
   exchange[ 0 ] = '\0';
   line[ 0 ] = '\0';
}

ostream &operator<<( ostream &output, PhoneNumber &number )
{
   output << "(" << number.areaCode << ") " << number.exchange
          << "-" << number.line << '\n';

   return output;
}

istream &operator>>( istream &input, PhoneNumber &number )
{
   cin.getline( number.phone, 15 );

   if ( strlen( number.phone ) != 14 )
      cin.clear( ios::failbit );

   if ( number.phone[ 1 ] == '0' || number.phone[ 6 ] == '0' ||
        number.phone[ 1 ] == '1' || number.phone[ 6 ] == '1')
      cin.clear( ios::failbit );

   if ( number.phone[ 2 ] != '0' && number.phone[ 2 ] != '1' )
      cin.clear( ios::failbit );

   if ( !cin.fail() ) {
      for ( int loop = 0; loop <= 2; ++loop ) {
         number.areaCode[ loop ] = number.phone[ loop + 1 ];
         number.exchange[ loop ] = number.phone[ loop + 6 ];
      }

      number.areaCode[ loop ] = number.exchange[ loop ] = '\0';

      for ( loop = 0; loop <= 3; ++loop )
         number.line[ loop ] = number.phone[ loop + 10 ];

      number.line[ loop ] = '\0';
   }
   else {
      cerr << "Invalid phone number entered.\n";
      exit( 1 );
   }

   return input;
}
```

```
// driver for p11_14.cpp
#include "p11_14.h"

int main()
{
   PhoneNumber telephone;

   cout << "Enter a phone number in the form (123) 456-7890:\n";
   cin >> telephone;

   cout << "The phone number entered was:  " << telephone << endl;

   cout << "Now enter an invalid phone number:\n";
   cin >> telephone;

   return 0;
}
```

```
Enter a phone number in the form (123) 456-7890:
(800) 987-4567
The phone number entered was:  (800) 987-4567

Now enter an invalid phone number:
(000) 000-0000
Invalid phone number entered.
```

11.15 Write a program that accomplishes each of the following:

a) Create the user-defined class **Point** that contains the private integer data members **xCoordinate** and **yCoordinate** and declares stream-insertion and stream-extraction overloaded operator functions as **friend**s of the class.

b) Define the stream-insertion and stream-extraction operator functions. The stream-extraction operator function should determine if the data entered are valid data, and if not, it should set the **ios::failbit** to indicate improper input. The stream-insertion operator should not be able to display the point after an input error occurred.

c) Write a **main** function that tests input and output of user-defined class **Point** using the overloaded stream-extraction and stream-insertion operators.

```
// P11_15.H
#ifndef P11_15_H
#define P11_15_H
#include <iostream.h>

class Point {
   friend ostream &operator<<( ostream&, Point& );
   friend istream &operator>>( istream&, Point& );
private:
   int xCoordinate;
   int yCoordinate;
};

#endif
```

```
// P11_15M.cpp
// member function definitions for p11_15.cpp
#include "p11_15.h"

ostream& operator<<( ostream& out, Point& p )
{
   if ( !cin.fail() )
```

```
      cout << "(" << p.xCoordinate << ", " << p.yCoordinate << ")" << '\n';
   else
      cout << "\nInvalid data\n";

   return out;
}

istream& operator>>( istream& i, Point& p )
{
   if ( cin.peek() != '(' )
      cin.clear( ios::failbit );
   else
      i.ignore();  // skip (

   cin >> p.xCoordinate;

   if ( cin.peek() != ',' )
      cin.clear( ios::failbit );
   else {
      i.ignore(); // skip ,

      if ( cin.peek() == ' ' )
         i.ignore(); // skip space
      else
         cin.clear( ios::failbit );
   }

   cin >> p.yCoordinate;

   if ( cin.peek() == ')' )
         i.ignore();  // skip )
      else
         cin.clear( ios::failbit );

   return i;
}
```

```
// driver for p11_15.cpp
#include "p11_15.h"

int main()
{
   Point pt;

   cout << "Enter a point in the form (x, y):\n";
   cin >> pt;

   cout << "Point entered was: " << pt << endl;
   return 0;
}
```

```
Enter a point in the form (x, y):
(7, 8)
Point entered was: (7, 8)
```

11.16 Write a program that accomplishes each of the following:

a) Create the user-defined class **`Complex`** that contains the private integer data members **`real`** and **`imaginary`**, and declares stream-insertion and stream-extraction overloaded operator functions as **`friend`**s of the class.

b) Define the stream-insertion and -extraction operator functions. The stream-extraction operator function should determine if the data entered are valid, and if not, it should set **ios::failbit** to indicate improper input. The input should be of the form

3 + 8i

c) The values can be negative or positive, and it is possible that one of the two values is not provided. If a value is not provided, the appropriate data member should be set to 0. The stream-insertion operator should not be able to display the point if an input error occurred. The output format should be identical to the input format shown above. For negative imaginary values, a minus sign should be printed rather than a plus sign.

d) Write a **main** function that tests input and output of user-defined class **Complex** using the overloaded stream-extraction and stream-insertion operators.

```
// P11_16.H
#ifndef P11_16_H
#define P11_16_H
#include <iostream>

using std::ostream;
using std::istream;

class Complex {
   friend ostream &operator<<( ostream&, Complex& );
   friend istream &operator>>( istream&, Complex& );
public:
   Complex( void );    // constructor
private:
   int real;
   int imaginary;
};

#endif
```

```
// P11_16M.cpp
// member function definitions for p11_16.cpp

#include <iostream>

using std::cout;
using std::cin;
using std::ios;
using std::ostream;
using std::istream;

#include <iomanip>

using std::setiosflags;
using std::resetiosflags;

#include "p11_16.h"

Complex::Complex( void )
{
   real = 0;
   imaginary = 0;
}

ostream &operator<<( ostream &output, Complex &c )
{
   if ( !cin.fail() )
      output << c.real
```

```
                 << setiosflags( ios::showpos )
                 << c.imaginary << "i\n"
                 << resetiosflags( ios::showpos );
   else
      output << "Invalid Data Entered" << '\n';

   return output;
}

istream &operator>>( istream &input, Complex &c )
{
   int number, multiplier;
   char temp;

   input >> number;

   if ( cin.peek() == ' ' ) {             // case a + bi
      c.real = number;
      cin >> temp;

      multiplier = ( temp == '+' ) ? 1 : -1;

      if ( cin.peek() != ' ' )
         cin.clear( ios::failbit );       // set bad bit
      else {

         if ( cin.peek() == ' ' ) {
            input >> c.imaginary;
            c.imaginary *= multiplier;

            cin >> temp;
            if ( cin.peek() != '\n' )
               cin.clear( ios::failbit ); // set bad bit
         }
         else
            cin.clear( ios::failbit );    // set bad bit
      }
   }
   else if ( cin.peek() == 'i' ) {         // case bi
         cin >> temp;

         if ( cin.peek() == '\n' ) {
            c.real = 0;
            c.imaginary = number;
         }
         else
            cin.clear( ios::failbit );     // set bad bit
   }

   else if ( cin.peek() == '\n' ) {        // case a
      c.real = number;
      c.imaginary = 0;
   }
   else
      cin.clear( ios::failbit );           // set bad bit

   return input;
}
```

```
// driver for p11_16.cpp
#include "p11_16.h"
```

```

int main()
{
   Complex complex;

   cout << "Input a complex number in the form A + Bi:\n";
   cin >> complex;

   cout << "Complex number entered was:\n" << complex << endl;
   return 0;
}
```

```
Input a complex number in the form A + Bi:
7 - 7777i
Complex number entered was:
7-7777i
```

11.17 Write a program that uses a **for** structure to print a table of ASCII values for the characters in the ASCII character set from **33** to **126**. The program should print the decimal value, octal value, hexadecimal value and character value for each character. Use the stream manipulators **dec**, **oct** and **hex** to print the integer values.

```
// Exercise 11.17 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setiosflags;
using std::dec;
using std::oct;
using std::hex;

int main()
{
   cout << setw( 7 ) << "Decimal" << setw( 9 ) << "Octal " << setw( 15 )
        << "Hexadecimal " << setw( 13 ) << "Character"
        << setiosflags( ios::showbase ) << '\n';

   for ( int loop = 33; loop <= 126; ++loop )
       cout << setw( 7 ) << dec << loop << setw( 9 ) << oct << loop
            << setw( 15 ) << hex << loop << setw(13)
            << static_cast<char>( loop ) << endl;

   return 0;
}
```

```
  Decimal   Octal    Hexadecimal     Character
      33      041           0x21             !
      34      042           0x22             "
      35      043           0x23             #
      36      044           0x24             $
      37      045           0x25             %
      38      046           0x26             &
      39      047           0x27             '
...
     120     0170           0x78             x
     121     0171           0x79             y
     122     0172           0x7a             z
     123     0173           0x7b             {
     124     0174           0x7c             |
     125     0175           0x7d             }
     126     0176           0x7e             ~
```

11.18 Write a program to show that the `getline` and three-argument `get` `istream` member functions each end the input string with a string-terminating null character. Also, show that `get` leaves the delimiter character on the input stream while `getline` extracts the delimiter character and discards it. What happens to the unread characters in the stream?

```cpp
// Exercise 11.18 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <cctype>

const int SIZE = 80;

int main()
{
   char array[ SIZE ], array2[ SIZE ], c;

   cout << "Enter a sentence to test getline() and get():\n";
   cin.getline( array, SIZE, '*' );
   cout << array << '\n';

   cin >> c;  // read next character in input
   cout << "The next character in the input is: " << c << '\n';

   cin.get( array2, SIZE, '*' );
   cout << array2 << '\n';

   cin >> c;  // read next character in input
   cout << "The next character in the input is: " << c << '\n';

   return 0;
}
```

```
Enter a sentence to test getline() and get():
wishing*on*a*star
wishing
The next character in the input is: o
n
The next character in the input is: *
```

11.19 Write a program that creates the user-defined manipulator **`skipwhite`** to skip leading whitespace characters in the input stream. The manipulator should use the**`isspace`** function from the **`<cctype>`** library to test if the character is a whitespace character. Each character should be input using the **`istream`** member function **`get`**. When a nonwhitespace character is encountered, the **`skipwhite`** manipulator finishes its job by placing the character back on the input stream and returning an **`istream`** reference.

Test the manipulator by creating a **`main`** function in which the **`ios::skipws`** flag is unset so that the stream-extraction operator does not automatically skip whitespace. Then test the manipulator on the input stream by entering a character precededby whitespace as input. Print the character that was input to confirm that a whitespace character was not input.

12

Templates Solutions

Solutions

12.3 Write a function template **bubbleSort** based on the sort program of Fig. 5.15. Write a driver program that inputs, sorts and outputs an **int** array and a **float** array.

```
// Exercise 12.3 solution
// This program puts values into an array, sorts the values into
// ascending order, and prints the resulting array.
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

template < class T >
void swap( T * const, T * const);

// Function template for bubbleSort
template < class T >
void bubbleSort( T * const array, int size )
{

   for ( int pass = 1; pass < size; ++pass )
      for ( int j = 0; j < size - pass; ++j )
         if ( array[ j ] > array[ j + 1 ] )
            swap( array + j, array + j + 1  );

}

template < class T >
void swap( T * const element1Ptr, T * const element2Ptr )
{
   T temp = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = temp;
}

int main()
{
   const int arraySize = 10;
   int a[ arraySize ] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 }, i;

   // Process an array of integers
   cout << "Integer data items in original order\n";

   for ( i = 0; i < arraySize; ++i )
      cout << setw( 6 ) << a[ i ];

   bubbleSort( a, arraySize );          // sort the array
   cout << "\nInteger data items in ascending order\n";

   for ( i = 0; i < arraySize; ++i )
      cout << setw( 6 ) << a[ i ];

   cout << "\n\n";

   // Process an array of doubleing point values
   double b[ arraySize ] = { 10.1, 9.9, 8.8, 7.7, 6.6, 5.5,
                             4.4, 3.3, 2.2, 1.1 };

   cout << "doubleing point data items in original order\n";

   for ( i = 0; i < arraySize; ++i )
      cout << setw( 6 ) << b[ i ];

   bubbleSort( b, arraySize );          // sort the array
   cout << "\ndoubleing point data items in ascending order\n";

   for ( i = 0; i < arraySize; ++i )
      cout << setw( 6 ) << b[ i ];

   cout << endl;

   return 0;
}
```

```
Integer data items in original order
    10     9     8     7     6     5     4     3     2     1
Integer data items in ascending order
     1     2     3     4     5     6     7     8     9    10

doubleing point data items in original order
  10.1   9.9   8.8   7.7   6.6   5.5   4.4   3.3   2.2   1.1
doubleing point data items in ascending order
   1.1   2.2   3.3   4.4   5.5   6.6   7.7   8.8   9.9  10.1
```

12.4 Overload function template `printArray` of Fig.12.2 so that it takes two additional integer arguments, namely `int lowSubscript` and `int highSubscript`. A call to this function will print only the designated portion of the array. Validate `lowSubscript` and `highSubscript`; if either is out-of-range or if `highSubscript` is less than or equal to `low-`

Subscript, the overloaded **printArray** function should return 0; otherwise, **printArray** should return the number of elements printed. Then modify **main** to exercise both versions of **printArray** on arrays **a**, **b** and **c**. Be sure to test all capabilities of both versions of **printArray**.

```
// Exercise 12.4 solution
// Using template functions
#include <iostream>

using std::cout;
using std::endl;


template< class T >
int printArray( T const * const array, int size, int lowSubscript,
                int highSubscript )
{
   if ( size < 0 || lowSubscript < 0 || highSubscript >= size )
      return 0;   // negative size or subscript out of range

   for ( int i = lowSubscript, count = 0; i <= highSubscript; ++i ) {
      ++count;
      cout << array[ i ] << ' ';
   }

   cout << '\n';

   return count;  // number or elements output
}

int main()
{
   const int aCount = 5, bCount = 7, cCount = 6;
   int a[ aCount ] = { 1, 2, 3, 4, 5 };
   double b[ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
   char c[ cCount ] = "HELLO";  // 6th position for null
   int elements;

   cout << "\nArray a contains:\n";
   elements = printArray( a, aCount, 0, aCount - 1 );
   cout << elements << " elements were output\n";

   cout << "Array a from 1 to 3 is:\n";
   elements = printArray( a, aCount, 1, 3 );
   cout << elements << " elements were output\n";

   cout << "Array a output with invalid subscripts:\n";
   elements = printArray( a, aCount, -1, 10 );
   cout << elements << " elements were output\n\n";

   cout << "Array b contains:\n";
   elements = printArray( b, bCount, 0, bCount - 1 );
   cout << elements << " elements were output\n";

   cout << "Array b from 1 to 3 is:\n";
   elements = printArray( b, bCount, 1, 3 );
   cout << elements << " elements were output\n";

   cout << "Array b output with invalid subscripts:\n";
   elements = printArray( b, bCount, -1, 10 );
   cout << elements << " elements were output\n\n";

   cout << "Array c contains:\n";
   elements = printArray( c, cCount, 0, cCount - 1 );
   cout << elements << " elements were output\n";

   cout << "Array c from 1 to 3 is:\n";
   elements = printArray( c, cCount, 1, 3 );
   cout << elements << " elements were output\n";

   cout << "Array c output with invalid subscripts:\n";
   elements = printArray( c, cCount, -1, 10 );
   cout << elements << " elements were output" << endl;

   return 0;
}
```

```
Array a contains:
1 2 3 4 5
5 elements were output
Array a from 1 to 3 is:
2 3 4
3 elements were output
Array a output with invalid subscripts:
0 elements were output

Array b contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7
7 elements were output
Array b from 1 to 3 is:
2.2 3.3 4.4
3 elements were output
Array b output with invalid subscripts:
0 elements were output

Array c contains:
H E L L O
6 elements were output
Array c from 1 to 3 is:
E L L
3 elements were output
Array c output with invalid subscripts:
0 elements were output
```

12.5 Overload function template **printArray** of Fig.12.2 with a nontemplate version that specifically prints an array of character strings in neat, tabular, column format.

```
// Exercise 12.5 solution
// Using template functions
#include <iostream>

using std::cout;

#include <iomanip>

using std::setw;

template< class T >
void printArray( T const * const array, int size )
{
   for ( int i = 0; i < size; ++i )
      cout << array[ i ] << ' ';

   cout << '\n';
}

void printArray( char const * const stringArray[], int size )
{
   for ( int i = 0; i < size; ++i ) {
      cout << setw( 10 ) << stringArray[ i ];

      if ( ( i + 1 ) % 4 == 0 )
         cout << '\n';
   }

   cout << '\n';
}

int main()
{
   const int aCount = 5, bCount = 7, cCount = 6, sCount = 8;
   int a[ aCount ] = { 1, 2, 3, 4, 5 };
   double b[ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
   char c[ cCount ] = "HELLO";  // 6th position for null
   const char *strings[ sCount ] = { "one", "two", "three", "four",
                                     "five", "six", "seven", "eight" };

   cout << "Array a contains:\n";
   printArray( a, aCount );  // integer template function

   cout << "\nArray b contains:\n";
   printArray( b, bCount );  // float template function

   cout << "\nArray c contains:\n";
   printArray( c, cCount );  // character template function

   cout << "\nArray strings contains:\n";
   printArray( strings, sCount );  // function specific to string arrays

   return 0;
}
```

```
Array a contains:
1 2 3 4 5

Array b contains:
1.1 2.2 3.3 4.4 5.5 6.6 7.7

Array c contains:
H E L L O

Array strings contains:
       one       two     three      four
      five       six     seven     eight
```

12.6 Write a simple function template for predicate function **isEqualTo** that compares its two arguments with the equality operator (**==**) and returns **true** if they are equal and **false** if they are not equal. Use this function template in a program that calls **isEqualTo** only with a variety of built-in types. Now write a separate version of the program that calls **isEqualTo** with a user-defined class type, but does not overload the equality operator. What happens when you attempt to run this program? Now overload the equality operator (with operator function `operator==`). Now what happens when you attempt to run this program?

```
// Exercise 12.6 solution
// Combined solution to entire problem
#include <iostream>

using std::cout;
using std::cin;
using std::ostream;

template < class T >
bool isEqualTo( const T &arg1, const T &arg2 ) { return arg1 == arg2; }

class SomeClass {
   friend ostream &operator<<(ostream &, SomeClass &);
public:
   SomeClass( int s, double t )
   {
      x = s;
      y = t;
   }

   // Overloaded equality operator. If this is not provided, the
   // program will not compile.
   bool operator==( const SomeClass &right ) const
      { return x == right.x && y == right.y; }
private:
   int x;
   double y;
};

ostream &operator<<( ostream &out, SomeClass &obj )
{
   out << '(' << obj.x << ", " << obj.y << ')';
   return out;
}

int main()
{
   int a, b;

   cout << "Enter two integer values: ";
   cin >> a >> b;
   cout << a << " and " << b << " are "
        << ( isEqualTo( a, b ) ? "equal" : "not equal" ) << '\n';

   char c, d;

   cout << "\nEnter two character values: ";
   cin >> c >> d;
   cout << c << " and " << d << " are "
        << ( isEqualTo( c, d ) ? "equal" : "not equal" ) << '\n';

   double e, f;

   cout << "\nEnter two double values: ";
   cin >> e >> f;

   cout << e << " and " << f << " are "
        << ( isEqualTo( e, f ) ? "equal" : "not equal") << '\n';

   SomeClass g( 1, 1.1 ), h( 1, 1.1 );

   cout << "\nThe class objects " << g << " and " << h << " are "
        << ( isEqualTo( g, h ) ? "equal" : "not equal" ) << '\n';
   return 0;
}
```

```
Enter two integer values: 8 22
8 and 22 are not equal

Enter two character values: Y Y
Y and Y are equal

Enter two double values: 3.3 8.7
3.3 and 8.7 are not equal

The class objects (1, 1.1) and (1, 1.1) are equal
```

12.7 Use a nontype parameter **`numberOfElements`** and a type parameter **`elementType`** to help create a template for the **`Array`** class we developed in Chapter 8, "Operator Overloading." This template will enable **`Array`** objects to be instantiated with a specified number of elements of a specified element type at compile time.

```
#ifndef ARRAY1_H
#define ARRAY1_H

#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <cassert>

template < class elementType, int numberOfElements >
class Array {
public:
   Array();                                 // default constructor
   ~Array();                                // destructor
   int getSize() const;                     // return size
   bool operator==( const Array & ) const;  // compare equal
   bool operator!=( const Array & ) const;  // compare !equal
   elementType &operator[]( int );          // subscript operator
   static int getArrayCount();              // Return count of
                                            // arrays instantiated.
   void inputArray();                       // input the array elements
   void outputArray() const;                // output the array elements
private:
   elementType ptr[ numberOfElements ]; // pointer to first element of array
   int size; // size of the array
   static int arrayCount;  // # of Arrays instantiated
};

// Initialize static data member at file scope
template < class elementType, int numberOfElements >
int Array< elementType, numberOfElements >::arrayCount = 0;   // no objects yet


// Default constructor for class Array
template < class elementType, int numberOfElements >
```

```
Array< elementType, numberOfElements >::Array()
{
   ++arrayCount;                  // count one more object
   size = numberOfElements;

   for ( int i = 0; i < size; ++i )
      ptr[ i ] = 0;                // initialize array
}

// Destructor for class Array
template < class elementType, int numberOfElements >
Array< elementType, numberOfElements >::~Array() { --arrayCount; }

// Get the size of the array
template < class elementType, int numberOfElements >
int Array< elementType, numberOfElements >::getSize() const { return size; }

// Determine if two arrays are equal and
// return true or false.
template < class elementType, int numberOfElements >
bool Array< elementType, numberOfElements >::
            operator==( const Array &right ) const
{
   if ( size != right.size )
      return false;    // arrays of different sizes

   for ( int i = 0; i < size; ++i )
      if ( ptr[ i ] != right.ptr[ i ] )
         return false; // arrays are not equal

   return true;       // arrays are equal
}

// Determine if two arrays are not equal and
// return true or false.
template < class elementType, int numberOfElements >
bool Array< elementType, numberOfElements >::
           operator!=( const Array &right ) const
{
   if ( size != right.size )
      return true;         // arrays of different sizes

   for ( int i = 0; i < size; ++i )
      if ( ptr[ i ] != right.ptr[ i ] )
         return true;      // arrays are not equal

   return false;            // arrays are equal
}

// Overloaded subscript operator
template < class elementType, int numberOfElements >
elementType &Array< elementType, numberOfElements >::
             operator[]( int subscript )
{
   // check for subscript out of range error
   assert( 0 <= subscript && subscript < size );

   return ptr[ subscript ];   // reference return creates lvalue
}

// Return the number of Array objects instantiated
template < class elementType, int numberOfElements >
```

```
int Array< elementType, numberOfElements >::getArrayCount()
   { return arrayCount; }

// Input values for entire array.
template < class elementType, int numberOfElements >
void Array< elementType, numberOfElements >::inputArray()
{
   for ( int i = 0; i < size; ++i )
      cin >> ptr[ i ];
}

// Output the array values
template < class elementType, int numberOfElements >
void Array< elementType, numberOfElements >::outputArray() const
{
   for ( int i = 0; i < size; ++i ) {
      cout << ptr[ i ] << ' ';

      if ( ( i + 1 ) % 10 == 0 )
         cout << '\n';
   }

   if ( i % 10 != 0 )
      cout << '\n';
}

#endif
```

```
// Exercise 12.7 solution
#include <iostream>

using std::cout;

#include "arraytmp.h"

int main()
{
   Array< int, 5 > intArray;

   cout << "Enter " << intArray.getSize() << " integer values:\n";
   intArray.inputArray();

   cout << "\nThe values in intArray are:\n";
   intArray.outputArray();

   Array< float, 5 > floatArray;

   cout << "\nEnter " << floatArray.getSize()
        << " floating point values:\n";
   floatArray.inputArray();

   cout << "\nThe values in the double array are:\n";
   floatArray.outputArray();

   return 0;
}
```

```
Enter 5 integer values:
99 98 97 96 95

The values in intArray are:
99 98 97 96 95

Enter 5 floating point values:
1.12 1.13 1.45 1.22 9.11

The values in the doubleArray are:
1.12 1.13 1.45 1.22 9.11
```

12.8 Write a program with class template **`Array`**. The template can instantiate an **`Array`** of any element type. Override the template with a specific definition for an **`Array`** of **`float`** elements (**`class Array< float >`**). The driver should demonstrate the instantiation of an **`Array`** of **`int`** through the template and should show that an attempt to instantiate an **`Array`** of **`float`** uses the definition provided in **`class Array< float >`**.

12.9 Distinguish between the terms "function template" and "template function."
ANS: A function template is used to instantiate template functions.

12.10 Which is more like a stencil—a class template or a template class? Explain your answer.
ANS: A class template can be viewed as a stencil from which a template class can be created. A template class can be viewed as a stencil from which objects of that class can be created. So, in a way, both can be viewed as stencils.

12.11 What is the relationship between function templates and overloading?
ANS: Function templates create overloaded versions of a function. The main difference is at compile time, where the compiler automatically creates the code for the template functions from the function template rather than the programmer creating the code.

12.12 Why might you choose to use a function template instead of a macro?
ANS: A macro is a text substitution done by the preprocessor. A function template provides real function definitions with all the type checking to ensure proper function calls.

12.13 What performance problem can result from using function templates and class templates?
ANS: There can be a tremendous proliferation of code in the program due to many copies of code generated by the compiler.

12.14 The compiler performs a matching process to determine which template function to call when a function is invoked. Under what circumstances does an attempt to make a match result in a compile error?
ANS: If the compiler cannot match the function call made to a template or if the matching process results in multiple matches at compile time, the compiler generates an error.

12.15 Why is it appropriate to call a class template a parameterized type?
ANS: When creating template classes from a class template, it is necessary to provide a type (or possibly several types) to complete the definition of the new type being declared. For example, when creating an "array of integers" from an **`Array`** class template, the type **`int`** is provided to the class template to complete the definition of an array of integers.

12.16 Explain why you might use the statement

```
Array< Employee > workerList( 100 );
```

in a C++ program.
ANS: When creating template classes from a class template, it is necessary to provide a type (or possibly several types) to complete the definition of the new type being declared. For example, when creating an "array of integers" from an **`Array`** class template, the type **`int`** is provided to the class template to complete the definition of an array of integers.

12.17 Review your answer to Exercise 12.16. Now, why might you use the statement

```
Array< Employee > workerList;
```

in a C++ program?
ANS: Declares an **`Array`** object to store an **`Employee`**. The default constructor is used.

12.18 Explain the use of the following notation in a C++ program:

```
template< class T > Array< T >::Array( int s )
```

ANS: This notation is used to begin the definition of the `Array( int )` constructor for the class template `Array`.

12.19 Why might you typically use a nontype parameter with a class template for a container such as an array or stack?
ANS: To specify at compile time the size of the container class object being declared.

12.20 Describe how to provide a class for a specific type to override the class template for that type.

12.21 Describe the relationship between class templates and inheritance.

12.22 Suppose a class template has the header

```
template< class T1 > class C1
```

Describe the friendship relationships established by placing each of the following friendship declarations inside this class template header. Identifiers beginning with "`f`" are functions, identifiers beginning with "`C`" are classes and identifiers beginning with "`T`" can represent any type (i.e., built-in types or class types).

a) `friend void f1();`
ANS: To specify at compile time the size of the container class object being declared.

b) `friend void f2( C1< T1 > &);`
ANS: Function `f2` for a specific type of `T1` is a `friend` of the template class of type `T1`. For example, if `T1` is of type `int`, the function with the prototype

```
void f2( C1< int > & );
```

is a `friend` of the class `C1< int >`.

c) `friend void C2::f4();`
ANS: Function `f4` of class `C2` is a `friend` of all template classes instantiated from class template `C1`.

d) `friend void C3< T1 >::f5( C1< T1 > & );`
ANS: Function `f5` of class `C3` for a specific type of `T1` is a `friend` of the template class of type `T1`. For example, if `T1` is `int`, the function with the prototype

```
void c3< int >::f5( C1< int > & );
```

e) `friend class C5;`
ANS: Makes every member function of class `C5` a `friend` of all template classes instantiated from the class template `C1`.

f) `friend class C6< T1 >;`
ANS: For a specific type `T1`, makes every member function of `C6< T1 >` a `friend` of class `C1< T1 >`. For example, if `T1` is `int`, every member function of class `C6< int >` is a `friend` of `C1< int >`.

12.23 Suppose class template `Employee` has a `static` data member `count`. Suppose three template classes are instantiated from the class template. How many copies of the `static` data member will exist? How will the use of each be constrained (if at all)?

13

Exception Handling Solutions

Solutions

13.20 List the various exceptional conditions that have occurred in programs throughout this text. List as many additional exceptional conditions as you can. For each of these, describe briefly how a program would typically handle the exception using the e x-ception-handling techniques discussed in this chapter. Some typical exceptions are division by zero, arithmetic overflow, array subscript out of bounds, exhaustion of the free store, etc.

13.21 Under what circumstances would the programmer not provide a parameter name when defining the type of the object that will be caught by a handler?

ANS: If there is no information in the object that is required in the handler, a parameter name is not required in the handler.

13.22 A program contains the statement

```
throw;
```

Where would you normally expect to find such a statement? What if that statement appeared in a different part of the program?

ANS: The statement would be found in an exception handler to rethrow an exception. If any **`throw`** expression occurs outside a **`try`** block, the function **`unexpected`** is called.

13.23 Under what circumstances would you use the following statement?

```
catch(...) { throw; }
```

ANS: The preceding statement is used to **`catch`** any exception and rethrow it for handling by an exception handler in a function within the call stack.

13.24 Compare and contrast exception handling with the various other error-processing schemes discussed in the text.

ANS: Exception handling enables the programmer to build more robust classes with built-in error processing capabilities. Once created, such classes allow clients of classes to concentrate on using the classes rather than defining what should happen if an error occurs while using the class. Exception handling offers the possibility that an error can be processed and that the program can continue execution. Other forms of error checking such as **`assert`** exit the program immediately without any further processing.

13.25 List the advantages of exception handling over conventional means of error processing.

13.26 Provide reasons why exceptions should not be used as an alternate form of program control.

ANS: Exceptions were designed for "exceptional cases." Exceptions do not follow conventional forms of program control. Therefore, using exceptions for anything other than error processing will not be easily recognized by others reading the code. This may make the program more difficult to modify and maintain.

13.27 Describe a technique for handling related exceptions.

ANS: Create a base class for all related exceptions. In the base class, derive all the related exception classes. Once the exception class hierarchy is created, exceptions from the hierarchy can be caught as the base class exception type or as one of the derived class exception types.

13.28 Until this chapter, we have found that dealing with errors detected by constructors is a bit awkward. Exception handling gives us a much better means of dealing with such errors. Consider a constructor for a `String` class. The constructor uses `new` to obtain space from the free store. Suppose `new` fails. Show how you would deal with this without exception handling. Discuss the key issues. Show how you would deal with such memory exhaustion with exception handling. Explain why the exception handling method is superior.

```
// Exercise 13.28 Solution
#include <iostream>

using std::cout;
using std::cerr;

#include <new>
#include <cstdlib>

void message( void );

int main()
{
   long double *d[ 50 ];
   set_new_handler( message );

   for ( int q = 0; q < 50; ++q ){
      d[ q ] = new long double[ 10000000 ];
  cout << "allocated 10000000 long doubles to d[" << q << "]\n";
   }

   cout << "Memory Allocated\n";

   return 0;
}

void message( void )
{
   cerr << "Memory Allocation Failed\n";
   exit( EXIT_FAILURE );
}
```

```
Assertion failed: new_p == 0, file setnewh.cpp, line 52

abnormal program termination
```

13.29 Suppose a program `throw`s an exception and the appropriate exception handler begins executing. Now suppose that the exception handler itself `throw`s the same exception. Does this create an infinite recursion? Write a program to check your observation.

```
// Exercise 13.29 solution
#include <iostream>

using std::cout;

class TestException {
public:
   TestException( char *mPtr ) : message( mPtr ) {}
   void print() const { cout << message << '\n'; }
private:
   char *message;
};

int main()
{
   try {
      throw TestException( "This is a test" );
   }
   catch ( TestException &t ) {
      t.print();
      throw TestException( "This is another test" );
   }

   return 0;
}
```

```
This is a test

abnormal program termination
```

13.30 Use inheritance to create a base exception class and various derived exception classes. Then show that a **catch** handler specifying the base class can **catch** derived-class exceptions.

```
// Exercise 13.30 Solution
#include <iostream>

using std::cout;

#include <cstdlib>
#include <ctime>

class BaseException {
public:
   BaseException( char *mPtr ) : message( mPtr ) {}
   void print() const { cout << message << '\n'; }
private:
   char *message;
};

class DerivedException : public BaseException {
public:
   DerivedException( char *mPtr ) : BaseException( mPtr ) {}
};

class DerivedException2 : public DerivedException {
public:
   DerivedException2( char *mPtr ) : DerivedException( mPtr ) {}
};

int main()
{
   srand( time( 0 ) );

   try {
      throw ( rand() % 2 ? DerivedException( "DerivedException" ) :
                           DerivedException2( "DerivedException2" ) );
   }
   catch ( BaseException &b ) {
      b.print();
   }

   return 0;
}
```

```
DerivedException2
```

13.31 Show a conditional expression that returns either a **double** or an **int**. Provide an **int catch** handler and a **double catch** handler. Show that only the **double catch** handler executes regardless of whether the **int** or the **double** is returned.

```
// Exercise 13.31 Solution
#include <iostream>
using std::cerr;

int main()
{
   try {
      int a = 7;
      double b = 9.9;

      throw a < b ? a : b;
   }
   catch ( int x ) {
      cerr << "The int value " << x << " was thrown\n";
   }
   catch ( double y ) {
      cerr << "The double value " << y << " was thrown\n";
   }

   return 0;
}
```

```
The double value 7 was thrown
```

13.32 Write a program designed to generate and handle a memory exhaustion error. Your program should loop on a request to create dynamic storage through operator **new**.

```
// Exercise 13.32 solution
#include <iostream>

using std::cout;
using std::cerr;

#include <new>
using std::bad_alloc;

#include <cstdlib>

int main()
{
   long double *ptr[ 10 ];

   try {
      for ( int i = 0; i < 10; ++i ) {
         ptr[ i ] = new long double[ 50000000 ];
         cout << "Allocated 50000000 long doubles in ptr[ "
              << i << " ]\n";
      }
   }
   catch ( bad_alloc ex ) {
      cerr << "Memory Allocation Failed.\n";
      exit( EXIT_FAILURE );
   }

   return 0;
}
```

```
Allocated 5000000 long doubles in ptr[ 0 ]
Allocated 5000000 long doubles in ptr[ 1 ]
Allocated 5000000 long doubles in ptr[ 2 ]
Allocated 5000000 long doubles in ptr[ 3 ]
Allocated 5000000 long doubles in ptr[ 4 ]
Allocated 5000000 long doubles in ptr[ 5 ]
Allocated 5000000 long doubles in ptr[ 6 ]
Allocated 5000000 long doubles in ptr[ 7 ]
Allocated 5000000 long doubles in ptr[ 8 ]
Allocated 5000000 long doubles in ptr[ 9 ]
```

13.33 Write a program which shows that all destructors for objects constructed in a block are called before an exception is thrown from that block.

```
// Exercise 13.33 Solution
#include <iostream>

using std::cout;
using std::cerr;


class Object {
public:
   Object( int val ) : value( val )
{ cout << "Object " << value << " constructor\n"; }
   ~Object()
{ cout << "Object " << value << " destructor\n"; }
private:
   int value;
};

class Error {
public:
   Error( char *s ) : string( s ) {}
   void print() const { cerr << '\n' << string << '\n'; }
private:
   char *string;
};

int main()
{
   try {
      Object a( 1 ), b( 2 ), c( 3 );
      cout << '\n';
      throw Error( "This is a test exception" );
   }
   catch ( Error &e ) {
      e.print();
   }

   return 0;
}
```

```
Object 1 constructor
Object 2 constructor
Object 3 constructor

Object 3 destructor
Object 2 destructor
Object 1 destructor

This is a test exception
```

13.34 Write a program which shows that member object destructors are called for only those member objects that were constructed before an exception occurred.

```
// Exercise 13_34 Solution
#include <iostream>

using std::cout;
using std::cerr;

// A sample exception class
class ExceptionClass {
public:
   ExceptionClass() : message( "An exception was thrown" ) {}
   void print() const { cerr << '\n' << message << '\n'; }
private:
   char *message;
};

// A class from which to build member objects
class Member {
public:
   Member( int val ) : value( val )
   {
      cout << "Member object " << value << " constructor called\n";

      // If value is 3, throw an exception for demonstration purposes.
      if ( value == 3 )
         throw ExceptionClass();
   }

   ~Member()
   { cout << "Member object " << value << " destructor called\n"; }
private:
   int value;
};

// A class to encapsulate objects of class Member
class Encapsulate {
public:
   Encapsulate() : m1( 1 ), m2( 2 ), m3( 3 ), m4( 4 ), m5( 5 ) {}
private:
   Member m1, m2, m3, m4, m5;
};

int main()
{
   cout << "Constructing an object of class Encapsulate\n";

   try {
      Encapsulate e;
   }
   catch( ExceptionClass &except ) {
      except.print();
   }

   return 0;
}
```

```
Constructing an object of class Encapsulate
Member object 1 constructor called
Member object 2 constructor called
Member object 3 constructor called
Member object 2 destructor called
Member object 1 destructor called

An exception was thrown
```

13.35 Write a program that demonstrates how any exception is caught with `catch(...)`.

```
// Exercise 13.35 Solution
#include <iostream>

using std::cout;
using std::cerr;


// A sample exception class
class ExceptionClass {
public:
   ExceptionClass() : message( "An exception was thrown" ) {}
   void print() const { cerr << '\n' << message << '\n'; }
private:
   char *message;
};

void generateException();

int main()
{
   try {
      generateException();
   }
   catch( ... ) {
      cerr << "The \"catch all\" exception handler was invoked\n";
   }

   return 0;
}

void generateException()
```

```
{
   throw ExceptionClass();
}
```

```
The "catch all" exception handler was invoked
```

13.36 Write a program which shows that the order of exception handlers is important. The first matching handler is the one that executes. Compile and run your program two different ways to show that two different handlers execute with two different effects.

13.37 Write a program that shows a constructor passing information about constructor failure to an exception handler after a `try` block.

```
// Exercise 13.37 Solution
#include <iostream>

using std::cout;
using std::cerr;

class InvalidIDNumberError {
public:
   InvalidIDNumberError( char *s ) : errorMessage( s ) {}
   void print() const { cerr << errorMessage; }
private:
   char *errorMessage;
};

class TestInvalidIDNumberError {
public:
   TestInvalidIDNumberError( int id ) : idNumber( id )
   {
      cout << "Constructor for object " << idNumber << '\n';

      if ( idNumber < 0 )
         throw InvalidIDNumberError( "ERROR: Negative ID number" );
   }
private:
   int idNumber;
};

int main()
{
   try {
      TestInvalidIDNumberError valid( 10 ), invalid( -1 );
   }
   catch ( InvalidIDNumberError &error ) {
      error.print();
      cerr << '\n';
   }

   return 0;
}
```

```
Constructor for object 10
Constructor for object -1
ERROR: Negative ID number
```

13.38 Write a program that uses a multiple inheritance hierarchy of exception classes to create a situation in which the order of exception handlers matters.

13.39 Using **`setjmp`**/**`longjmp`**, a program can transfer control immediately to an error routine from a deeply nested function invocation. Unfortunately, as the stack is unwound, destructors are not called for the automatic objects that were created during the sequence of nested function calls. Write a program which demonstrates that these destructors are, in fact, not called.

13.40 Write a program that illustrates rethrowing an exception.

```
// Exercise 13.40 Solution
#include <iostream>

using std::cout;
using std::cerr;

class TestException {
public:
   TestException( char *m ) : message( m ) {}
   void print() const { cout << message << '\n'; }
private:
   char *message;
};

void f() { throw TestException( "Test exception thrown" ); }

void g()
{
   try {
      f();
   }
   catch ( ... ) {
      cerr << "Exception caught in function g(). Rethrowing...\n";
      throw;
   }
}

int main()
{
   try {
      g();  // start function call chain
   }
   catch ( ... ) {
      cerr << "Exception caught in function main()\n";
   }

   return 0;
}
```

```
Exception caught in function g(). Rethrowing...
Exception caught in function main()
```

13.41 Write a program that uses **`set_unexpected`** to set a user-defined function for **`unexpected`**, uses **`set_unexpected`** again, and then resets **`unexpected`** back to its previous function. Write a similar program to test **`set_terminate`** and **`terminate`**.

13.42 Write a program which shows that a function with its own **`try`** block does not have to catch every possible error generated within the **`try`**. Some exceptions can slip through to, and be handled in, outer scopes.

```
// Exercise 13.42 Solution
#include <iostream>
```

```

using std::cout;
using std::cerr;

class TestException1 {
public:
   TestException1( char *m ) : message( m ) {}
   void print() const { cerr << message << '\n'; }
private:
   char *message;
};

class TestException2 {
public:
   TestException2( char *m ) : message( m ) {}
   void print() const { cout << message << '\n'; }
private:
   char *message;
};

void f()
{
   throw TestException1( "TestException1" );
}

void g()
{
   try {
      f();
   }
   catch ( TestException2 &t2 ) {
      cerr << "In g: Caught ";
      t2.print();
   }
}

int main()
{
   try {
      g();
   }
   catch ( TestException1 &t1 ) {
      cerr << "In main: Caught ";
      t1.print();
   }

   return 0;
}
```

```
In main: Caught TestException1
```

13.43 Write a program that `throw`s an error from a deeply nested function call and still has the `catch` handler following the `try` block enclosing the call chain catch the exception.

```
// Exercise 13.43 Solution
#include <iostream>

using std::cout;
using std::cerr;

```

```
class TestException {
public:
   TestException( char *m ) : message( m ) {}
   void print() const { cout << message << '\n'; }
private:
   char *message;
};

void f() { throw TestException( "TestException" ); }

void g() { f(); }

void h() { g(); }

int main()
{
   try {
      h();
   }
   catch ( TestException &t ) {
      cerr << "In main: Caught ";
      t.print();
   }

   return 0;
}
```

```
In main: Caught TestException
```

14

File Processing Solutions

Solutions

14.5 Fill in the blanks in each of the following:

a) Computers store large amounts of data on secondary storage devices as ________.
ANS: files.

b) A ________ is composed of several fields.
ANS: record.

c) A field that may contain only digits, letters and blanks is called an ________ field.
ANS: alphanumeric.

d) To facilitate the retrieval of specific records from a file, one field in each record is chosen as a ________.
ANS: key.

e) The vast majority of information stored in computer systems is stored in ________ files.
ANS: sequential.

f) A group of related characters that conveys meaning is called a ________.
ANS: field.

g) The standard stream objects declared by header file `<iostream>` are ________, ________, ________ and ________.
ANS: `cin`, `cout`, `cerr`, `clog`.

h) `ostream` member function ________ outputs a character to the specified stream.
ANS: `put`.

i) `ostream` member function ________ is generally used to write data to a randomly accessed file.
ANS: `write`.

j) `istream` member function ________ repositions the file position pointer in a file.
ANS: `seekg`.

14.6 State which of the following are *true* and which are *false*. If *false*, explain why.

a) The impressive functions performed by computers essentially involve the manipulation of zeros and ones.
ANS: True.

b) People prefer to manipulate bits instead of characters and fields because bits are more compact.
ANS: False. People prefer to manipulate characters and fields because they are less cumbersome and more understandable.

c) People specify programs and data items as characters; computers then manipulate and process these characters as groups of zeros and ones.
ANS: True.

d) A person's 5-digit zip code is an example of a numeric field.

ANS: True.

e) A person's street address is generally considered to be an alphabetic field in computer applications.

ANS: False. A street address is generally considered to be alphanumeric.

f) Data items represented in computers form a data hierarchy in which data items become larger and more complex as we progress from fields to characters to bits, etc.

ANS: False. Data items processed by a computer form a data hierarchy in which data items become larger and more complex as we progress from bits to characters to fields, etc.

g) A record key identifies a record as belonging to a particular field.

ANS: False. A record key identifies a record as belonging to a particular person or entity.

h) Most organizations store all information in a single file to facilitate computer processing.

ANS: False. Most organizations have many files in which they store their information.

i) Each statement that processes a file in a C++ program explicitly refers to that file by name.

ANS: False. A pointer to each file is used to refer to the file.

j) When a program creates a file, the file is automatically retained by the computer for future reference.

ANS: True.

14.7 Exercise 14.3 asked the reader to write a series of single statements. Actually, these statements form the core of an important type of file processing program, namely, a file-matching program. In commercial data processing, it is common to have several files in each application system. In an accounts receivable system, for example, there is generally a master file containing detailed information about each customer such as the customer's name, address, telephone number, outstanding balance, credit limit, discount terms, contract arrangements, and possibly a condensed history of recent purchases and cash payments.

As transactions occur (e.g., sales are made and cash payments arrive), they are entered into a file. At the end of each business period (a month for some companies, a week for others, and a day in some cases) the file of transactions (called `"trans.dat"` in Exercise 14.3) is applied to the master file (called `"oldmast.dat"` in Exercise 14.3), thus updating each account's record of purchases and payments. During an updating run, the master file is rewritten as a new file (`"newmast.dat"`), which is then used at the end of the next business period to begin the updating process again.

File-matching programs must deal with certain problems that do not exist in single-file programs. For example, a match does not always occur. A customer on the master file may not have made any purchases or cash payments in the current business period, and therefore no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments may have just moved to this community, and the company may not have had a chance to create a master record for this customer.

Use the statements from Exercise 14.3 as a basis for writing a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes. Assume that each file is a sequential file with records stored in increasing order by account number.

When a match occurs (i.e., records with the same account number appear on both the master and transaction files), add the dollar amount on the transaction file to the current balance on the master file, and write the `"newmast.dat"` record. (Assume purchases are indicated by positive amounts on the transaction file and payments are indicated by negative amounts.) When there is a master record for a particular account but no corresponding transaction record, merely write the master record to `"newmast.dat"`. When there is a transaction record but no corresponding master record, print the message `"Unmatched transaction record for account number …"` (fill in the account number from the transaction record).

```
// Exercise 14.7 Solution
#include <iostream>

using std::cout;
using std::ios;
using std::cerr;


#include <iomanip>

using std::setprecision;

#include <fstream>

using std::ofstream;
using std::ifstream;

#include <cstdlib>
```

```

void printOutput( ofstream&, int, const char *, const char *, double );

int main()
{
   int masterAccount, transactionAccount;
   double masterBalance, transactionBalance;
   char masterFirstName[ 20 ], masterLastName[ 20 ];

   ifstream inOldMaster( "oldmast.dat" ),
            inTransaction( "trans.dat" );
   ofstream outNewMaster( "newmast.dat" );

   if ( !inOldMaster ) {
      cerr << "Unable to open oldmast.dat\n";
      exit( EXIT_FAILURE );
   }

   if ( !inTransaction ) {
      cerr << "Unable to open trans.dat\n";
      exit( EXIT_FAILURE );
   }

   if ( !outNewMaster ) {
      cerr << "Unable to open newmast.dat\n";
      exit( EXIT_FAILURE );
   }

   cout << "Processing...\n";
   inTransaction >> transactionAccount >> transactionBalance;

   while ( !inTransaction.eof() ) {
      inOldMaster >> masterAccount >> masterFirstName
                  >> masterLastName >> masterBalance;

      while ( masterAccount < transactionAccount && !inOldMaster.eof() ) {
         printOutput( outNewMaster, masterAccount, masterFirstName,
                      masterLastName, masterBalance );
         inOldMaster >> masterAccount >> masterFirstName
                     >> masterLastName >> masterBalance;
      }

      if ( masterAccount > transactionAccount ) {
         cout << "Unmatched transaction record for account "
              << transactionAccount << '\n';

         inTransaction >> transactionAccount >> transactionBalance;
      }

      if ( masterAccount == transactionAccount ) {
         masterBalance += transactionBalance;
         printOutput( outNewMaster, masterAccount, masterFirstName,
                      masterLastName, masterBalance );
      }

      inTransaction >> transactionAccount >> transactionBalance;
   }

   inTransaction.close();
   outNewMaster.close();
   inOldMaster.close();

   return 0;
}

void printOutput( ofstream &oRef, int mAccount, const char *mfName,
                  const char *mlName, double mBalance )
{
      cout.setf( ios::fixed | ios::showpoint );
      oRef.setf( ios::fixed | ios::showpoint );

      oRef << mAccount << ' ' << mfName << ' ' << mlName << ' '
           << setprecision( 2 ) << mBalance << '\n';
      cout << mAccount << ' ' << mfName << ' ' << mlName << ' '
           << setprecision( 2 ) << mBalance << '\n';
}
```

```
Processing...
100 Ajax Guppie 678.06
300 Sue Pirhana 650.07
400 Clint Bass 994.22
Unmatched transaction record for account 445
500 Aias Shark 895.80
700 Tom Mahi-Mahi -23.57
900 Marisal Carp 200.55
```

Contents of “oldmast.dat”

```
100 Ajax Guppie 543.89
300 Sue Pirhana 22.88
400 Clint Bass -6.32
500 Aias Shark 888.71
700 Tom Mahi-Mahi 76.09
900 Marisal Carp 100.55
```

Contents of “trans.dat”

```
100 134.17
300 627.19
400 1000.54
445 55.55
500 7.09
700 -99.66
900 100.00
```

Contents of “newmast.dat”

```
100 Ajax Guppie 678.06
300 Sue Pirhana 650.07
400 Clint Bass 994.22
500 Aias Shark 895.80
700 Tom Mahi-Mahi -23.57
900 Marisal Carp 200.55
```

14.8 After writing the program of Exercise 14.7, write a simple program to create some test data for checking out the program. Use the following sample account data:

Master file Account number	Name	Balance
100	Alan Jones	348.17
300	Mary Smith	27.19
500	Sam Sharp	0.00
700	Suzy Green	-14.22

Transaction file Account number	Transaction amount
100	27.14
300	62.11
400	100.56
900	82.17

```
// Exercise 14.8 Solution
#include <iostream>

using std::cout;
using std::ios;
using std::cerr;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <fstream>

using std::ofstream;
using std::ifstream;

#include <cstdlib>
#include <ctime>

int main()
{
   const char *firstNames[] = { "Walter", "Alice", "Alan", "Mary", "Steve",
                                "Gina", "Tom", "Cindy", "Ilana", "Pam" },
              *lastNames[] = { "Red", "Blue", "Yellow", "Orange", "Purple",
                               "Green", "Violet", "White", "Black", "Brown" };
   ofstream outOldMaster( "oldMast.dat" ),
            outTransaction( "trans.dat" );

   int z;

   srand( time( 0 ) );

   if ( !outOldMaster ) {
      cerr << "Unable to open oldmast.dat\n";
      exit( EXIT_SUCCESS );
   }

   if ( !outTransaction ) {
      cerr << "Unable to open trans.dat\n";
      exit( EXIT_SUCCESS );
   }

   // write data to "oldmast.dat"
   cout << setiosflags( ios::fixed | ios::showpoint )
        << "Contents of \"oldmast.dat\":\n";

   outOldMaster.setf( ios::fixed | ios::showpoint );
   for ( z = 1; z < 11; ++z ) {
      int value = rand() % 10, value2 = rand() % 50;
      outOldMaster << z * 100 << ' ' << firstNames[ z - 1 ] << ' '
                   << lastNames[ value ] << ' ' << setprecision( 2 )
                   << ( value * 100 ) / ( value2 / 3 + 4.32 ) << '\n';
      cout << z * 100 << ' ' << firstNames[ z - 1 ] << ' '
           << lastNames[ value ] << ' ' << setprecision( 2 )
           << ( value * 100 ) / ( value2 / 3 + 4.32 ) << '\n';
   }

   // write data to "trans.dat"
   cout << "\nContents of \"trans.dat\":\n";
   outTransaction.setf( ios::fixed | ios::showpoint );

   for ( z = 1; z < 11; ++z ) {
      int value = 25 - rand() % 50;
      outTransaction << z * 100 << ' ' << setprecision( 2 )
                     << ( value * 100 ) / ( 2.667 * ( 1 + rand() % 10 ) )
                     << '\n';
      cout << z * 100 << ' ' << setprecision( 2 )
           << ( value * 100 ) / ( 2.667 * ( 1 + rand() % 10 ) ) << '\n';
   }

   outTransaction.close();
   outOldMaster.close();

   ifstream inMaster( "oldmast.dat" ), inTrans( "trans.dat" );
   ofstream newMaster( "newMast.dat" );

   if ( !inMaster ) {
      cerr << "Unable to open oldmast.dat\n";
      exit( EXIT_SUCCESS );
   }

   if ( !inTrans ) {
      cerr << "Unable to open trans.dat\n";
      exit( EXIT_SUCCESS );
   }

   if ( !newMaster ) {
      cerr << "Unable to open newmast.dat\n";
      exit( EXIT_SUCCESS );
   }

   int account, mAccount;
   double balance, mBalance;
   char mFirst[ 20 ], mLast[ 20 ];

   cout << "Processing...\n";
   inTrans >> account >> balance;

   while ( !inTrans.eof() ) {
      inMaster >> mAccount >> mFirst >> mLast >> mBalance;

      while ( mAccount < account && !inMaster.eof() )
         inMaster >> mAccount >> mFirst >> mLast >> mBalance;

      if ( mAccount > account ) {
         cout << "Unmatched transaction record: account " << account << '\n';

         inTrans >> account >> balance;
      }

      if ( mAccount == account ) {
         mBalance += balance;
         newMaster << mAccount << ' ' << mFirst << ' ' << mLast
                   << ' ' << mBalance << '\n';
      }

      inTrans >> account >> balance;
   }

   newMaster.close();
   inTrans.close();
   inMaster.close();

   return 0;
}
```

```
Contents of "oldmast.dat":
100 Walter Brown 46.58
200 Alice Orange 47.47
300 Alan Green 28.87
400 Mary Black 49.02
500 Steve White 67.83
600 Gina Orange 56.39
700 Tom White 38.21
800 Cindy Red 0.00
900 Ilana Brown 87.21
1000 Pam Red 0.00

Contents of "trans.dat":
100 -149.98
200 262.47
300 -41.24
400 -412.45
500 23.43
600 318.71
700 -224.97
800 -33.33
900 -249.97
1000 -71.24
```

Contents of “oldmast.dat”

```
100 Walter Brown 46.58
200 Alice Orange 47.47
300 Alan Green 28.87
400 Mary Black 49.02
500 Steve White 67.83
600 Gina Orange 56.39
700 Tom White 38.21
800 Cindy Red 0.00
900 Ilana Brown 87.21
1000 Pam Red 0.00
```

Contents of “trans.dat”

```
100 -149.98
200 262.47
300 -41.24
400 -412.45
500 23.43
600 318.71
700 -224.97
800 -33.33
900 -249.97
1000 -71.24
```

14.9 Run the program of Exercise 14.7 using the files of test data created in Exercise 14.8. Print the new master file. Check that the accounts have been updated correctly.

ANS: Contents of newmast.dat

```
100 Walter Brown -103.40
200 Alice Orange 309.94
300 Alan Green -12.37
400 Mary Black -363.43
500 Steve White 91.26
600 Gina Orange 375.10
700 Tom White -186.76
800 Cindy Red -33.33
900 Ilana Brown -162.76
1000 Pam Red -71.24
```

14.10 It is possible (actually common) to have several transaction records with the same record key. This occurs because a particular customer might make several purchases and cash payments during a business period. Rewrite your accounts receivable file-matching program of Exercise 14.7 to provide for the possibility of handling several transaction records with the same record key. Modify the test data of Exercise 14.8 to include the following additional transaction records:

Account number	Dollar amount
300	83.89
700	80.78
700	1.53

```cpp
// Exercise 14.10 Solution
#include <iostream>
```

```

using std::cout;
using std::ios;
using std::cerr;

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include <cstdlib>

void printOutput( ofstream &, int, const char *, const char *, double );

int main()
{
   int masterAccount, transactionAccount;
   double masterBalance, transactionBalance;
   char masterFirstName[ 20 ], masterLastName[ 20 ];

   ifstream inOldmaster( "oldmast.dat" ),
            inTransaction( "trans.dat" );
   ofstream outNewmaster( "newmast.dat" );

   if ( !inOldmaster ) {
      cerr << "Unable to open oldmast.dat\n";
      exit( EXIT_FAILURE );
   }

   if ( !inTransaction ) {
      cerr << "Unable to open trans.dat\n";
      exit( EXIT_FAILURE );
   }

   if ( !outNewmaster ) {
      cerr << "Unable to open newmast.dat\n";
      exit( EXIT_FAILURE );
   }

   cout << "Processing....\n";
   inTransaction >> transactionAccount >> transactionBalance;

   while ( !inTransaction.eof() ) {
      inOldmaster >> masterAccount >> masterFirstName >> masterLastName
                  >> masterBalance;

      while ( masterAccount < transactionAccount && !inOldmaster.eof() ) {
         printOutput( outNewmaster, masterAccount, masterFirstName,
                      masterLastName, masterBalance );
         inOldmaster >> masterAccount >> masterFirstName >> masterLastName
                     >> masterBalance;
      }

      if ( masterAccount > transactionAccount ) {
         cout << "Unmatched transaction record for account "
              << transactionAccount << '\n';
         inTransaction >> transactionAccount >> transactionBalance;
      }
      else if ( masterAccount < transactionAccount ) {
         cout << "Unmatched transaction record for account "
              << transactionAccount << '\n';
         inTransaction >> transactionAccount >> transactionBalance;
      }

      while ( masterAccount == transactionAccount &&
              !inTransaction.eof() ) {
         masterBalance += transactionBalance;
         inTransaction >> transactionAccount >> transactionBalance;
      }

      printOutput( outNewmaster, masterAccount, masterFirstName,
                   masterLastName, masterBalance );
   }

   inTransaction.close();
   outNewmaster.close();
   inOldmaster.close();

   return 0;
}

void printOutput( ofstream &oRef, int mAccount, const char *mfName,
                  const char *mlName, double mBalance )
{
      cout.setf( ios::showpoint | ios::fixed );
      oRef.setf( ios::showpoint | ios::fixed );
      oRef << mAccount << ' ' << mfName << ' ' << mlName << ' '
           << setprecision( 2 ) << mBalance << '\n';
      cout << mAccount << ' ' << mfName << ' ' << mlName << ' '
           << setprecision( 2 ) << mBalance << '\n';
      cout.unsetf( ios::showpoint | ios::fixed );
}
```

```
Processing....
100 Walter Brown -103.40
200 Alice Orange 309.94
300 Alan Green 71.52
400 Mary Black -363.43
500 Steve White 91.26
600 Gina Orange 375.10
700 Tom White -104.45
800 Cindy Red -33.33
900 Ilana Brown -162.76
1000 Pam Red -71.24
```

Contents of “oldmast.dat”

```
100 Walter Brown 46.58
200 Alice Orange 47.47
300 Alan Green 28.87
400 Mary Black 49.02
500 Steve White 67.83
600 Gina Orange 56.39
700 Tom White 38.21
800 Cindy Red 0.00
900 Ilana Brown 87.21
1000 Pam Red 0.00
```

Contents of “trans.dat”

```
100 -149.98
200 262.47
300 -41.24
300 83.89
400 -412.45
500 23.43
600 318.71
700 -224.97
700 80.78
700 1.53
800 -33.33
900 -249.97
1000 -71.24
```

Contents of “newmast.dat”

```
100 Walter Brown -103.40
200 Alice Orange 309.94
300 Alan Green 71.52
400 Mary Black -363.43
500 Steve White 91.26
600 Gina Orange 375.10
700 Tom White -104.45
800 Cindy Red -33.33
900 Ilana Brown -162.76
1000 Pam Red -71.24
```

14.11 Write a series of statements that accomplish each of the following. Assume the structure

```
struct Person {
   char lastName[ 15 ];
   char firstName[ 15 ];
   char age[ 4 ];
};
```

has been defined, and that the random-access file has been opened properly.

a) Initialize the file `"nameage.dat"` with 100 records containing `lastName = "unassigned"`, `firstName = ""` and `age = "0"`.

ANS:

```
// fstream object "fileObject" corresponds to file "nameage.dat"
Person personInfo = { "unassigned", "", 0 };
for ( int r = 0; r < 100; ++r )
   fileObject.write( reinterpret_cast< char * >( &personInfo,
                     sizeof( personInfo ) );
```

b) Input 10 last names, first names and ages, and write them to the file.

ANS:

```
fileObject.seekp( 0 );
for ( int x = 1; x <= 10; ++x ) {
    cout << "Enter last name, first name, and age: ";
    cin >> personInfo.lastName >> personInfo.firstName >> personInfo.age;
    fileObject.write( reinterpret_cast< char * > &personInfo,
                            sizeof( personInfo ) );
}
```

c) Update a record that has information in it, and if there is none, tell the user "No info."
d) Delete a record that has information by reinitializing that particular record.

14.12 You are the owner of a hardware store and need to keep an inventory that can tell you what different tools you have, how many of each you have on hand and the cost of each one. Write a program that initializes the random-access file `"hardware.dat"` to one hundred empty records, lets you input the data concerning each tool, enables you to list all your tools, lets you delete a record for a tool that you no longer have and lets you update *any* information in the file. The tool identification number should be the record number. Use the following information to start your file:

Record #	Tool name	Quantity	Cost
3	Electric sander	7	57.98
17	Hammer	76	11.99
24	Jig saw	21	11.00
39	Lawn mower	3	79.50
56	Power saw	18	99.99
68	Screwdriver	106	6.99
77	Sledge hammer	11	21.50
83	Wrench	34	7.50

```cpp
// Exercise 14.12 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;
using std::cerr;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <cstring>
#include <cctype>
#include <cstdlib>

void initializeFile( fstream & );
void inputData( fstream & );
void listTools( fstream & );
void updateRecord( fstream & );
void insertRecord( fstream & );
void deleteRecord( fstream & );
int instructions( void );

const int LENGTH = 30;

struct Data {
   int partNumber;
   char toolName[ LENGTH ];
   int inStock;
   double unitPrice;
};
```

```

int main()
{
   int choice;
   char response;
   fstream file( "hardware.dat", ios::in | ios::out );
   void ( *f[] )( fstream & ) = { listTools, updateRecord, insertRecord,
                                   deleteRecord };

   if ( !file ) {
      cerr << "File could not be opened.\n";
      exit( EXIT_FAILURE );
   }

   cout << "Should the file be initialized (Y or N): ";
   cin >> response;

   while ( toupper( response ) != 'Y' && toupper( response ) != 'N' ) {
      cout << "Invalid response. Enter Y or N: ";
      cin >> response;
   }

   if ( toupper( response ) == 'Y' ) {
      initializeFile( file );
      inputData( file );
   }

   while ( ( choice = instructions() ) != 5 ) {
      ( *f[ choice - 1 ] )( file );
      file.clear();    // reset eof indicator
   }

   file.close();

   return 0;
}

void initializeFile( fstream &fRef )
{
   Data blankItem = { -1, "", 0, 0.0 };

   // See Chapter 21 for a discussion of reinterpret_cast
   for ( int i = 0; i < 100; ++i )
      fRef.write( reinterpret_cast< char * >( &blankItem ), sizeof( Data ) );
}

void inputData( fstream &fRef )
{
   Data temp;

   cout << "Enter the partnumber (0 - 99, -1 to end input): ";
   cin >> temp.partNumber;

   while ( temp.partNumber != -1 ) {
      cout << "Enter the tool name: ";
      cin.ignore();  // ignore the newline on the input stream
      cin.get( temp.toolName, LENGTH );
      cout << "Enter quantity and price: ";
      cin >> temp.inStock >> temp.unitPrice;
      fRef.seekp( ( temp.partNumber ) * sizeof( Data ) );
      fRef.write( reinterpret_cast< char * >( &temp ), sizeof( Data ) );

      cout << "Enter the partnumber (0 - 99, -1 to end input): ";
      cin >> temp.partNumber;
   }
}

int instructions( void )
{
   int choice;

   cout << "\nEnter a choice:\n1  List all tools."
        << "\n2  Update record.\n3  Insert record."
        << "\n4  Delete record.\n5  End program.\n";

   do {
      cout << "? ";
      cin >> choice;
   } while ( choice < 1 || choice > 5 );

   return choice;
}

void listTools( fstream &fRef )
{
   Data temp;

   cout << setw( 7 ) << "Record#" << "    " << setiosflags( ios::left )
        << setw( 30 ) << "Tool name" << resetiosflags( ios::left )
        << setw( 13 ) << "Quantity" << setw( 10 ) << "Cost\n";

   for ( int count = 0; count < 100 && !fRef.eof(); ++count ) {
      fRef.seekg( count * sizeof( Data ) );
      fRef.read( reinterpret_cast< char * >( &temp ), sizeof( Data ) );

      if ( temp.partNumber >= 0 && temp.partNumber < 100 ) {
         cout.setf( ios::fixed | ios::showpoint );
         cout << setw( 7 ) << temp.partNumber << "    "
              << setiosflags( ios::left ) << setw( 30 ) << temp.toolName
              << resetiosflags( ios::left ) << setw( 13 ) << temp.inStock
              << setprecision( 2 ) << setw( 10 ) << temp.unitPrice << '\n';
      }
   }
}

void updateRecord( fstream &fRef )
{
   Data temp;
   int part;

   cout << "Enter the part number for update: ";
   cin >> part;
   fRef.seekg( part * sizeof( Data ) );
   fRef.read( reinterpret_cast< char * >( &temp ), sizeof( Data ) );

   if ( temp.partNumber != -1 ) {
      cout << setw( 7 ) << "Record#" << "    " << setiosflags( ios::left )
           << setw( 30 ) << "Tool name" << resetiosflags( ios::left )
           << setw( 13 ) << "Quantity" << setw( 10 ) << "Cost\n";

      cout.setf( ios::fixed | ios::showpoint );
      cout << setw( 7 ) << temp.partNumber << "    "
           << setiosflags( ios::left ) << setw( 30 ) << temp.toolName
           << resetiosflags( ios::left ) << setw( 13 ) << temp.inStock
```

```
            << setprecision( 2 ) << setw( 10 ) << temp.unitPrice << '\n'
            << "Enter the tool name: ";

      cin.ignore();  // ignore the newline on the input stream
      cin.get( temp.toolName, LENGTH );
      cout << "Enter quantity and price: ";
      cin >> temp.inStock >> temp.unitPrice;

      fRef.seekp( ( temp.partNumber ) * sizeof( Data ) );
      fRef.write( reinterpret_cast< char * > ( &temp ), sizeof( Data ) );
   }
   else
      cerr << "Cannot update. The record is empty.\n";
}

void insertRecord( fstream &fRef )
{
   Data temp;
   int part;

   cout << "Enter the partnumber for insertion: ";
   cin >> part;
   fRef.seekg( ( part ) * sizeof( Data ) );
   fRef.read( reinterpret_cast< char * > ( &temp ), sizeof( Data ) );

   if ( temp.partNumber == -1 ) {
      temp.partNumber = part;
      cout << "Enter the tool name: ";
      cin.ignore();  // ignore the newline on the input stream
      cin.get( temp.toolName, LENGTH );
      cout << "Enter quantity and price: ";
      cin >> temp.inStock >> temp.unitPrice;

      fRef.seekp( ( temp.partNumber ) * sizeof( Data ) );
      fRef.write( reinterpret_cast< char * >( &temp ), sizeof( Data ) );
   }
   else
      cerr << "Cannot insert. The record contains information.\n";
}

void deleteRecord( fstream &fRef )
{
   Data blankItem = { -1, "", 0, 0.0 }, temp;
   int part;

   cout << "Enter the partnumber for deletion: ";
   cin >> part;

   fRef.seekg( part * sizeof( Data ) );
   fRef.read( reinterpret_cast< char * >( &temp ), sizeof( Data ) );

   if ( temp.partNumber != -1 ) {
      fRef.seekp( part * sizeof( Data ) );
      fRef.write( reinterpret_cast< char * >( &blankItem ), sizeof( Data ) );
      cout << "Record deleted.\n";
   }
   else
      cerr << "Cannot delete. The record is empty.\n";
}
```

```
Should the file be initialized (Y or N): Y
Enter the partnumber (0 - 99, -1 to end input): 77
Enter the tool name: Sledge hammer

Enter quantity and price: 11 21.50
Enter the partnumber (0 - 99, -1 to end input): -1

Enter a choice:
1  List all tools.
2  Update record.
3  Insert record.
4  Delete record.
5  End program.
? 1
Record#    Tool name                          Quantity     Cost
     77    Sledge hammer                            11     21.50

Enter a choice:
1  List all tools.
2  Update record.
3  Insert record.
4  Delete record.
5  End program.
? 3

Enter the partnumber for insertion: 44
Enter the tool name: Pipe wrench
Enter quantity and price: 1 19.99

Enter a choice:
1  List all tools.
2  Update record.
3  Insert record.
4  Delete record.
5  End program.
? 2
Enter the part number for update: 44
Record#    Tool name                          Quantity     Cost
     44    Pipe wrench                               1     19.99

Enter the tool name: Pipe wrench
Enter quantity and price: 1 14.99

Enter a choice:
1  List all tools.
2  Update record.
3  Insert record.
4  Delete record.
5  End program.
? 1

Record#    Tool name                          Quantity     Cost
     44    Pipe wrench                               1     14.99
     77    Sledge hammer                            11     21.50

Enter a choice:
1  List all tools.
2  Update record.
3  Insert record.
4  Delete record.
5  End program.
? 4
Enter the partnumber for deletion: 44
Record deleted.
...
```

14.13 Modify the telephone number word-generating program you wrote in Chapter 4 so that it writes its output to a file. This allows you to read the file at your convenience. If you have a computerized dictionary available, modify your program to look up the thousands of seven-letter words in the dictionary. Some of the interesting seven-letter combinations created by this programay consist of two or more words. For example, the phone number 8432677 produces "THEBOSS." Modify your program to use the computerized dictionary to check each possible seven-letter word to see if it is a valid one-letter word followed by a valid sixletter word, a valid two-letter word followed by a valid five-letter word, etc.

```
// Exercise 14.13 Solution
#include <iostream>

using std::cout;
using std::cin;
using std::ios;
using std::cerr;

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <cstdlib>
#include <cctype>

void wordGenerator( const int * const );

int main()
{
   int phoneNumber[ 7 ] = { 0 };

   cout << "Enter a phone number (digits 2 through 9) "
        << "in the form: xxx-xxxx\n";

   // loop 8 times: 7 digits plus hyphen
   // hyphen is is not placed in phoneNumber
   for ( int u = 0, v = 0; u < 8; ++u ) {
      int i = cin.get();

      // See chapter 16 for a discussion of isdigit
      if ( isdigit( i ) )  // ctype library
         phoneNumber[ v++ ] = i - '0';
   }

   wordGenerator( phoneNumber );

   return 0;
}

void wordGenerator( const int * const n )
{
   ofstream outFile( "phone.dat" );
   const char *phoneLetters[ 10 ] = { "", "", "ABC", "DEF", "GHI", "JKL",
                                       "MNO", "PRS", "TUV", "WXY" };

   if ( !outFile ) {
      cerr << "\"phone.dat\" could not be opened.\n";
      exit( EXIT_FAILURE );
   }

   int count = 0;

   // output all possible combinations
   for ( int i1 = 0; i1 <= 2; ++i1 )
```

```
      for ( int i2 = 0; i2 <= 2; ++i2 )
         for ( int i3 = 0; i3 <= 2; ++i3 )
            for ( int i4 = 0; i4 <= 2; ++i4 )
               for ( int i5 = 0; i5 <= 2; ++i5 )
                  for ( int i6 = 0; i6 <= 2; ++i6 )
                     for ( int i7 = 0; i7 <= 2; ++i7 ) {
                        outFile << phoneLetters[ n[ 0 ] ][ i1 ]
                                << phoneLetters[ n[ 1 ] ][ i2 ]
                                << phoneLetters[ n[ 2 ] ][ i3 ]
                                << phoneLetters[ n[ 3 ] ][ i4 ]
                                << phoneLetters[ n[ 4 ] ][ i5 ]
                                << phoneLetters[ n[ 5 ] ][ i6 ]
                                << phoneLetters[ n[ 6 ] ][ i7 ] << ' ';

                        if ( ++count % 9 == 0 )
                           outFile << '\n';
                     }

   outFile << "\nPhone number is ";

   for ( int i = 0; i < 7; ++i ) {
      if ( i == 3 )
         outFile << '-';

      outFile << n[ i ];
   }

   outFile.close();
}
```

Contents of phone.dat

```
TMPJDPW TMPJDPX TMPJDPY TMPJDRW TMPJDRX TMPJDRY TMPJDSW TMPJDSX TMPJDSY
TMPJEPW TMPJEPX TMPJEPY TMPJERW TMPJERX TMPJERY TMPJESW TMPJESX TMPJESY
TMPJFPW TMPJFPX TMPJFPY TMPJFRW TMPJFRX TMPJFRY TMPJFSW TMPJFSX TMPJFSY
TMPKDPW TMPKDPX TMPKDPY TMPKDRW TMPKDRX TMPKDRY TMPKDSW TMPKDSX TMPKDSY
TMPKEPW TMPKEPX TMPKEPY TMPKERW TMPKERX TMPKERY TMPKESW TMPKESX TMPKESY
...
VOSLDPW VOSLDPX VOSLDPY VOSLDRW VOSLDRX VOSLDRY VOSLDSW VOSLDSX VOSLDSY
VOSLEPW VOSLEPX VOSLEPY VOSLERW VOSLERX VOSLERY VOSLESW VOSLESX VOSLESY
VOSLFPW VOSLFPX VOSLFPY VOSLFRW VOSLFRX VOSLFRY VOSLFSW VOSLFSX VOSLFSY

Phone number is 867-5379
```

14.14 Write a program that uses the **`sizeof`** operator to determine the sizes in bytes of the various data types on your computer system. Write the results to the file **`"datasize.dat"`** so you may print the results later. The format for the results in the file should be

Data type	Size
`char`	`1`
`unsigned char`	`1`
`short int`	`2`
`unsigned short int`	`2`
`int`	`4`
`unsigned int`	`4`

Data type	Size
long int	4
unsigned long int	4
float	4
double	8
long double	16

Note: The sizes of the built-in data types on your computer may differ from those listed above.

```
// Exercise 14.14 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cerr;

#include <iomanip>

using std::setw;

#include <fstream>

using std::ofstream;
using std::ifstream;

#include <cstdlib>

int main()
{
   ofstream outFile( "datasize.dat" );

   if ( !outFile ) {
      cerr << "Unable to open \"datasize.dat\".\n";
      exit( EXIT_FAILURE );
   }

   outFile << "Data type" << setw( 16 ) << "Size\nchar"
           << setw( 21 ) << sizeof( char )
           << "\nunsigned char" << setw( 12 ) << sizeof( unsigned char )
           << "\nshort int" << setw(16) << sizeof( short int )
           << "\nunsigned short int" << setw( 7 ) << sizeof( unsigned short )
           << "\nint" << setw( 22 ) << sizeof( int ) << '\n';

   outFile << "unsigned int" << setw( 13 ) << sizeof( unsigned )
           << "\nlong int" << setw( 17 ) << sizeof( long )
           << "\nunsigned long int" << setw( 8 ) << sizeof( unsigned long )
           << "\ndouble" << setw( 20 ) << sizeof( double )
           << "\ndouble" << setw( 19 ) << sizeof( double )
           << "\nlong double" << setw( 14 ) << sizeof( long double ) << endl;

   outFile.close();

   return 0;
}
```

Contents of datasize.dat

```
Data type                 Size
char                        1
unsigned char               1
short int                   2
unsigned short int          2
int                         4
unsigned int                4
long int                    4
unsigned long int           4
float                       4
double                      8
long double                 8
```

15

Data Structures Solutions

Solutions

15.6 Write a program that concatenates two linked list objects of characters. The program should include function **concatenate**, which takes references to both list objects as arguments and concatenates the second list to the first list.

```
// LIST.H
// Template List class definition
// Added copy constructor to member functions (not included in chapter).
#ifndef LIST_H
#define LIST_H

#include <iostream>

using std::cout;

#include <cassert>
#include "listnd.h"

template< class NODETYPE >
class List {
public:
   List();                            // default constructor
   List( const List< NODETYPE > & );  // copy constructor
   ~List();                           // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
```

```
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
}

// Insert a node at the back of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )             // List is empty
      return false;             // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}

// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;               // data
   ListNode *nextPtr;           // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// Exercise 15.6 solution
#include <iostream>

using std::cout;

#include "list.h"

template< class T >
void concatenate( List< T > &first, List< T > &second )
{
   List< T > temp( second ); // create a copy of second
   T value;              // variable to store removed item from temp

   while ( !temp.isEmpty() ) {
      temp.removeFromFront( value );  // remove value from temp list
      first.insertAtBack( value );    // insert at end of first list
   }
}

int main()
{
   List< char > list1, list2;
   char c;

   for ( c = 'a'; c <= 'e'; ++c )
      list1.insertAtBack( c );

   list1.print();

   for ( c = 'f'; c <= 'j'; ++c )
      list2.insertAtBack( c );

   list2.print();

   concatenate( list1, list2 );
   cout << "The new list1 after concatenation is:\n";
   list1.print();

   return 0;
}
```

```
The list is: a b c d e

The list is: f g h i j


All nodes destroyed

The new list1 after concatenation is:
The list is: a b c d e f g h i j

Destroying nodes ...
f g h i j
All nodes destroyed

Destroying nodes ...
a b c d e f g h i j
All nodes destroyed
```

15.7 Write a program that merges two ordered list objects of integers into a single ordered list object of integers. Function `merge` should receive references to each of the list objects to be merged, and should return an object containing the merged list.

```
// LIST.H
// Template List class definition
#ifndef LIST_H
#define LIST_H

#include <iostream>

using std::cout;

#include <cassert>
#include "listnd.h"

template< class NODETYPE >
class List {
public:
   List();                              // default constructor
   List( const List< NODETYPE > & );  // copy constructor
   ~List();                             // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {           // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
}

// Insert a node at the back of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {           // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )             // List is empty
      return false;             // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
```

```
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}

// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;               // data
   ListNode *nextPtr;           // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// Exercise 15.7 solution
#include <iostream>

using std::cout;

#include "list.h"

template< class T >
List< T > &merge( List< T > &first, List< T > &second )
{
   // If both lists are empty, return an empty result
   if ( first.isEmpty() && second.isEmpty() ) {
      List< T > *ptr = new List< T >;         // dynamically allocated
      return *ptr;
   }

   // If first list is empty, return result containing second list
   if ( first.isEmpty() ) {
      List< T > *ptr = new List< T >( second ); // dynamically allocated
      return *ptr;
   }

   // If second list is empty, return result containing first list
   if ( second.isEmpty() ) {
      List< T > *ptr = new List< T >( first ); // dynamically allocated
      return *ptr;
   }

   List< T > tempFirst( first ),   // create a copy of first
             tempSecond( second ); // create a copy of second
   List< T > *ptr = new List< T >; // dynamically allocated result object
   T value1, value2;

   tempFirst.removeFromFront( value1 );
   tempSecond.removeFromFront( value2 );

   while ( !tempFirst.isEmpty() && !tempSecond.isEmpty() ) {
      if ( value1 <= value2 ) {
         ptr -> insertAtBack( value1 );
         tempFirst.removeFromFront( value1 );
      }
      else {
         ptr -> insertAtBack( value2 );
         tempSecond.removeFromFront( value2 );
      }
   }

   // Insert the values currently in value1 and value2
   if ( value1 < value2 ) {
      ptr -> insertAtBack( value1 );
      ptr -> insertAtBack( value2 );
   }
   else {
      ptr -> insertAtBack( value2 );
      ptr -> insertAtBack( value1 );
   }

   // Complete the insertion of the list that is not empty.
   // NOTE: Only one of the following 2 while structures will execute
   // because one of the lists must be empty to exit the preceding while.
   if ( !tempFirst.isEmpty() )  // Items left in tempFirst? Insert in result.
      do {
         tempFirst.removeFromFront( value1 );
         ptr -> insertAtBack( value2 );
      } while ( !tempFirst.isEmpty() );
   else                         // Items left in tempSecond? Insert in result.
      do {
         tempSecond.removeFromFront( value2 );
         ptr -> insertAtBack( value2 );
      } while ( !tempSecond.isEmpty() );

   return *ptr;
}

int main()
{
   List< int > list1, list2;
   int i;

   for ( i = 1; i <= 9; i += 2 )
      list1.insertAtBack( i );

   list1.print();

   for ( i = 2; i <= 10; i += 2 )
      list2.insertAtBack( i );

   list2.print();

   List< int > &listRef = merge( list1, list2 );

   cout <<  "The merged list is:\n";
   listRef.print();

   delete &listRef;   // delete the dynamically allocated list

   return 0;
}
```

```
The list is: 1 3 5 7 9

The list is: 2 4 6 8 10


All nodes destroyed


All nodes destroyed

The merged list is:
The list is: 1 2 3 4 5 6 7 8 9 10

Destroying nodes ...
1 2 3 4 5 6 7 8 9 10
All nodes destroyed

Destroying nodes ...
2 4 6 8 10
All nodes destroyed

Destroying nodes ...
1 3 5 7 9
All nodes destroyed
```

15.8 Write a program that inserts 25 random integers from 0 to 100 in order in a linked list object. The program should calculate the sum of the elements and the floating-point average of the elements.

```
// LIST
// Template List class definition
// Added copy constructor to member functions (not included in chapter).
#ifndef LIST_H
#define LIST_H

#include <iostream>

using std::cout;

#include <cassert>
#include "Listnd.h"

template< class NODETYPE >
class List {
public:
   List();                            // default constructor
   List( const List< NODETYPE > & );  // copy constructor
   ~List();                           // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
```

```
}

// Insert a node at the back of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {           // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )             // List is empty
      return false;             // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}
```

```

// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

#endif
```

```
// LIST2
// Template List2 class definition
// Enhances List by adding insertInOrder
#ifndef LIST2_H
#define LIST2_H

#include <cassert>
#include "Listnd.h"
#include "List.h"

template< class NODETYPE >
class List2 : public List< NODETYPE > {
public:
   void insertInOrder( const NODETYPE & );
};

// Insert a node in order
template< class NODETYPE >
void List2< NODETYPE >::insertInOrder( const NODETYPE &value )
{
   if ( isEmpty() ) { // List is empty
      ListNode< NODETYPE > *newPtr = getNewNode( value );
      firstPtr = lastPtr = newPtr;
   }
   else {            // List is not empty
      if ( firstPtr -> getData() > value )
         insertAtFront( value );
      else if ( lastPtr -> getData() < value )
         insertAtBack( value );
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr -> getNextPtr(),
                              *previousPtr = firstPtr,
                              *newPtr = getNewNode( value );

         while ( currentPtr != lastPtr && currentPtr -> getData() < value ) {
            previousPtr = currentPtr;
            currentPtr = currentPtr -> getNextPtr();
         }

         previousPtr -> setNextPtr( newPtr );
         newPtr -> setNextPtr( currentPtr );
      }
   }
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;               // data
   ListNode *nextPtr;           // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// Exercise 15.8 solution
#include <iostream>
```

```

using std::cout;

#include <cstdlib>
#include <ctime>
#include "List2.h"

// Integer specific list sum
int sumList( List2< int > &listRef )
{
   List2< int > temp( listRef );
   int sum = 0, value;

   while ( !temp.isEmpty() ) {
      temp.removeFromFront( value );
      sum += value;
   }

   return sum;
}

// Integer specific list average
double aveList( List2< int > &listRef )
{
   List2< int > temp( listRef );
   int sum = 0, value, count = 0;

   while ( !temp.isEmpty() ) {
      temp.removeFromFront( value );
      ++count;
      sum += value;
   }

   return static_cast< double >( sum ) / count;
}

int main()
{
   srand( time( 0 ) );  // randomize the random number generator

   List2< int > intList;

   for ( int i = 1; i <= 25; ++i )
      intList.insertInOrder( rand() % 101 );

   intList.print();

   cout << "The sum of the elements is: " << sumList( intList ) << '\n';
   cout << "The average of the elements is: " << aveList( intList ) << '\n';

   return 0;
}
```

```
The list is: 6 11 14 22 23 25 31 34 41 42 43 43 46 47 48 51 54 61 66 72 72 82 89
 94 95


All nodes destroyed

The sum of the elements is: 1212

All nodes destroyed

The average of the elements is: 48.48
Destroying nodes ...
6 11 14 22 23 25 31 34 41 42 43 43 46 47 48 51 54 61 66 72 72 82 89 94 95
All nodes destroyed
```

15.9 Write a program that creates a linked list object of 10 characters and then creates a second list object containing a copy of the first list, but in reverse order.

```
// LIST.H
// Template List class definition
// Added copy constructor to member functions (not included in chapter).
#ifndef LIST_H
#define LIST_H

#include <iostream>

using std::cout;

#include <cassert>
#include "listnd.h"

template< class NODETYPE >
class List {
public:
   List();                              // default constructor
   List( const List< NODETYPE > & );  // copy constructor
   ~List();                             // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
}

// Insert a node at the back of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )             // List is empty
      return false;             // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}

// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;              // data
   ListNode *nextPtr;          // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// Exercise 15.9 solution
#include <iostream>

using std::cout;

#include "list.h"

// Function template that takes two List objects as arguments
// and makes a copy of the second argument reversed in the first argument.
template< class T >
void reverseList( List< T > &first, List< T > &second )
```

```
{
   List< T > temp( second ); // create a copy of second
   T value;                  // variable to store removed item from temp

   while ( !temp.isEmpty() ) {
      temp.removeFromFront( value );  // remove value from temp list
      first.insertAtFront( value );   // insert at beginning of first list
   }
}

int main()
{
   List< char > list1, list2;

   for ( char c = 'a'; c <= 'g'; ++c )
      list1.insertAtBack( c );

   list1.print();

   reverseList( list2, list1 );
   cout << "After reversing:\n";
   list2.print();

   return 0;
}
```

```
The list is: a b c d e f g


All nodes destroyed

After reversing:
The list is: g f e d c b a

Destroying nodes ...
g f e d c b a
All nodes destroyed

Destroying nodes ...
a b c d e f g
All nodes destroyed
```

15.10 Write a program that inputs a line of text and uses a stack object to print the line reversed.

```
// STACK.H
// Definition of class Stack
// NOTE: This Stack class is a standalone Stack class template.
#ifndef STACK_H
#define STACK_H

#include <iostream>

using std::cout;

#include <cassert>
#include "stacknd.h"

template < class T >
class Stack {
```

```
public:
   Stack();              // default constructor
   ~Stack();             // destructor
   void push( T & );     // insert item in stack
   T pop();              // remove item from stack
   bool isEmpty() const; // is the stack empty?
   void print() const;   // output the stack
   StackNode< T > *getTopPtr() const { return topPtr; }
private:
   StackNode< T > *topPtr;    // pointer to fist StackNode
};

// Member function definitions for class Stack
template < class T >
Stack< T >::Stack() { topPtr = 0; }

template < class T >
Stack< T >::~Stack()
{
   StackNode< T > *tempPtr, *currentPtr = topPtr;

   while ( currentPtr != 0 ) {
      tempPtr = currentPtr;
      currentPtr = currentPtr -> getNextPtr();
      delete tempPtr;
   }
}

template < class T >
void Stack< T >::push( T &d )
{
   StackNode< T > *newPtr = new StackNode< T >( d, topPtr );

   assert( newPtr != 0 );  // was memory allocated?
   topPtr = newPtr;
}

template < class T >
T Stack< T >::pop()
{
   assert( !isEmpty() );

   StackNode< T > *tempPtr = topPtr;

   topPtr = topPtr -> nextPtr;
   T poppedValue = tempPtr -> data;
   delete tempPtr;
   return poppedValue;
}

template < class T >
bool Stack< T >::isEmpty() const { return topPtr == 0; }

template < class T >
void Stack< T >::print() const
{
   StackNode< T > *currentPtr = topPtr;

   if ( isEmpty() )          // Stack is empty
      cout << "Stack is empty\n";
   else {                   // Stack is not empty
      cout << "The stack is:\n";

      while ( currentPtr != 0 ) {
         cout << currentPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
      }

      cout << '\n';
   }
}

#endif
```

```
// STACKND.H
// Definition of template class StackNode
#ifndef STACKND_H
#define STACKND_H

template< class T > class Stack;  // forward declaration

template < class T >
class StackNode {
   friend class Stack< T >;
public:
   StackNode( const T & = 0, StackNode * = 0 );
   T getData() const;
   void setNextPtr( StackNode *nPtr ) { nextPtr = nPtr; }
   StackNode *getNextPtr() const { return nextPtr; }
private:
   T data;
   StackNode *nextPtr;
};

// Member function definitions for class StackNode
template < class T >
StackNode< T >::StackNode( const T &d, StackNode< T > *ptr )
{
   data = d;
   nextPtr = ptr;
}

template < class T >
T StackNode< T >::getData() const { return data; }

#endif
```

```
// Exercise 15.10 solution
#include <iostream>

using std::cout;
using std::cin;

#include "stack.h"

int main()
{
   Stack< char > charStack;
   char c;

   cout << "Enter a sentence:\n";
   while ( ( c = static_cast< char >( cin.get() ) ) != '\n' )
      charStack.push( c );

   cout << "\nThe sentence in reverse is:\n";

   while ( !charStack.isEmpty() )
      cout << charStack.pop();

   cout << '\n';

   return 0;
}
```

```
Enter a sentence:
Hello World!

The sentence in reverse is:
!dlroW olleH
```

15.11 Write a program that uses a stack object to determine if a string is a palindrome (i.e., the string is spelled identically backwards and forwards). The program should ignore spaces and punctuation.

```
// STACK.H
// Definition of class Stack
// NOTE: This Stack class is a standalone Stack class template.
#ifndef STACK_H
#define STACK_H

#include <iostream>

using std::cout;

#include <cassert>
#include "stacknd.h"

template < class T >
class Stack {
public:
   Stack();               // default constructor
   ~Stack();              // destructor
   void push( T & );      // insert item in stack
   T pop();               // remove item from stack
   bool isEmpty() const;  // is the stack empty?
   void print() const;    // output the stack
   StackNode< T > *getTopPtr() const { return topPtr; }
private:
   StackNode< T > *topPtr;    // pointer to fist StackNode
};

// Member function definitions for class Stack
template < class T >
Stack< T >::Stack() { topPtr = 0; }

template < class T >
Stack< T >::~Stack()
{
   StackNode< T > *tempPtr, *currentPtr = topPtr;

   while ( currentPtr != 0 ) {
      tempPtr = currentPtr;
      currentPtr = currentPtr -> getNextPtr();
      delete tempPtr;
   }
}

template < class T >
void Stack< T >::push( T &d )
{
   StackNode< T > *newPtr = new StackNode< T >( d, topPtr );

   assert( newPtr != 0 );  // was memory allocated?
   topPtr = newPtr;
}

template < class T >
T Stack< T >::pop()
{
   assert( !isEmpty() );

   StackNode< T > *tempPtr = topPtr;

   topPtr = topPtr -> nextPtr;
   T poppedValue = tempPtr -> data;
   delete tempPtr;
   return poppedValue;
}

template < class T >
bool Stack< T >::isEmpty() const { return topPtr == 0; }

template < class T >
void Stack< T >::print() const
{
   StackNode< T > *currentPtr = topPtr;

   if ( isEmpty() )          // Stack is empty
      cout << "Stack is empty\n";
   else {                    // Stack is not empty
      cout << "The stack is:\n";

      while ( currentPtr != 0 ) {
         cout << currentPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
      }

      cout << '\n';
   }
}

#endif
```

```
// STACKND.H
// Definition of template class StackNode
#ifndef STACKND_H
#define STACKND_H

template< class T > class Stack;  // forward declaration

template < class T >
```

```
class StackNode {
   friend class Stack< T >;
public:
   StackNode( const T & = 0, StackNode * = 0 );
   T getData() const;
   void setNextPtr( StackNode *nPtr ) { nextPtr = nPtr; }
   StackNode *getNextPtr() const { return nextPtr; }
private:
   T data;
   StackNode *nextPtr;
};

// Member function definitions for class StackNode
template < class T >
StackNode< T >::StackNode( const T &d, StackNode< T > *ptr )
{
   data = d;
   nextPtr = ptr;
}

template < class T >
T StackNode< T >::getData() const { return data; }

#endif
```

```
// Exercise 15.11 solution
#include <iostream>

using std::cout;
using std::cin;

#include <cctype>

#include <cstring>
#include "stack.h"

int main()
{
   Stack< char > charStack;
   char c, string1[ 80 ], string2[ 80 ];
   int i = 0;

   cout << "Enter a sentence:\n";

   while ( ( c = static_cast< char >( cin.get() ) ) != '\n' )
      if ( isalpha( c ) ) {
         string1[ i++ ] = c;
         charStack.push( c );
      }

   string1[ i ] = '\0';

   i = 0;

   while ( !charStack.isEmpty() )
      string2[ i++ ] = charStack.pop();

   string2[ i ] = '\0';

   if ( strcmp( string1, string2 ) == 0 )
      cout << "\nThe sentence is a palindrome\n";
   else
      cout << "\nThe sentence is not a palindrome\n";

   return 0;
}
```

```
Enter a sentence:
oat y tao

The sentence is a palindrome
```

15.12 Stacks are used by compilers to help in the process of evaluating expressions and generating machine language code. In this and the next exercise, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like **3 + 4** and **7 / 9** in which the operator (**+** or **/** here) is written between its operands—this is called *infix notation*. Computers "prefer" *postfix notation* in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as **3 4 +** and **7 9 /**, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation and then evaluate the postfix version of the expression. Each of these algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack object in support of its operation, and in each algorithm the stack is used for a different purpose.

In this exercise, you will write a C++ version of the infix-to-postfix conversion algorithm. In the next exercise, you will write a C++ version of the postfix expression evaluation algorithm. Later in the chapter, you will discover that code you write in this exercise can help you implement a complete working compiler.

Write a program that converts an ordinary infix arithmetic expression (assume a valid expression is entered) with single-digit integers such as

```
(6 + 2) * 5 - 8 / 4
```

to a postfix expression. The postfix version of the preceding infix expression is

```
6 2 + 5 * 8 4 / -
```

The program should read the expression into character array **infix**, and use modified versions of the stack functions implemented in this chapter to help create the postfix expression in character array **postfix**. The algorithm for creating a postfix expression is as follows:

```
1) Push a left parenthesis '(' onto the stack.
2) Append a right parenthesis ')' to the end of infix.
3) While the stack is not empty, read infix from left to right and do the following:
      If the current character in infix is a digit, copy it to the next element of postfix.
      If the current character in infix is a left parenthesis, push it onto the stack.
      If the current character in infix is an operator,
         Pop operators (if there are any) at the top of the stack while they have equal or
            higher precedence than the current operator, and insert the popped
            operators in postfix.
         Push the current character in infix onto the stack.
      If the current character in infix is a right parenthesis
         Pop operators from the top of the stack and insert them in postfix until a left
            parenthesis is at the top of the stack.
         Pop (and discard) the left parenthesis from the stack.
```

The following arithmetic operations are allowed in an expression:

+	addition
-	subtraction
*****	multiplication
/	division
^	exponentiation

`%` modulus

The stack should be maintained with stack nodes that each contain a data member and a pointer to the next stack node.

Some of the functional capabilities you may want to provide are:

a) Function **`convertToPostfix`** that converts the infix expression to postfix notation.
b) Function **`isOperator`** that determines if **`c`** is an operator.
c) Function **`precedence`** that determines if the precedence of **`operator1`** is less than, equal to or greater than the precedence of **`operator2`**. The function returns -1, 0 and 1, respectively.
d) Function **`push`** that pushes a value onto the stack.
e) Function **`pop`** that pops a value off the stack.
f) Function **`stackTop`** that returns the top value of the stack without popping the stack.
g) Function **`isEmpty`** that determines if the stack is empty.
h) Function **`printStack`** that prints the stack.

```
// STACK.H
// Definition of class Stack
// NOTE: This Stack class is a standalone Stack class template.
#ifndef STACK_H
#define STACK_H

#include <iostream>

using std::cout;

#include <cassert>
#include "stacknd.h"

template < class T >
class Stack {
public:
   Stack();              // default constructor
   ~Stack();             // destructor
   void push( T & );     // insert item in stack
   T pop();              // remove item from stack
   bool isEmpty() const; // is the stack empty?
   void print() const;   // output the stack
   StackNode< T > *getTopPtr() const { return topPtr; }
private:
   StackNode< T > *topPtr;    // pointer to fist StackNode
};

// Member function definitions for class Stack
template < class T >
Stack< T >::Stack() { topPtr = 0; }

template < class T >
Stack< T >::~Stack()
{
   StackNode< T > *tempPtr, *currentPtr = topPtr;

   while ( currentPtr != 0 ) {
      tempPtr = currentPtr;
      currentPtr = currentPtr -> getNextPtr();
      delete tempPtr;
   }
}

template < class T >
void Stack< T >::push( T &d )
{
   StackNode< T > *newPtr = new StackNode< T >( d, topPtr );
   assert( newPtr != 0 );  // was memory allocated?
   topPtr = newPtr;
}

template < class T >
T Stack< T >::pop()
{
   assert( !isEmpty() );

   StackNode< T > *tempPtr = topPtr;

   topPtr = topPtr -> nextPtr;
   T poppedValue = tempPtr -> data;
   delete tempPtr;
   return poppedValue;
}

template < class T >
bool Stack< T >::isEmpty() const { return topPtr == 0; }

template < class T >
void Stack< T >::print() const
{
   StackNode< T > *currentPtr = topPtr;

   if ( isEmpty() )          // Stack is empty
      cout << "Stack is empty\n";
   else {                   // Stack is not empty
      cout << "The stack is:\n";

      while ( currentPtr != 0 ) {
         cout << currentPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
      }

      cout << '\n';
   }
}

#endif
```

```
// STACKND.H
// Definition of template class StackNode
#ifndef STACKND_H
#define STACKND_H

template< class T > class Stack;  // forward declaration

template < class T >
class StackNode {
   friend class Stack< T >;
public:
   StackNode( const T & = 0, StackNode * = 0 );
   T getData() const;
   void setNextPtr( StackNode *nPtr ) { nextPtr = nPtr; }
   StackNode *getNextPtr() const { return nextPtr; }
private:
   T data;
   StackNode *nextPtr;
};
```

```

// Member function definitions for class StackNode
template < class T >
StackNode< T >::StackNode( const T &d, StackNode< T > *ptr )
{
   data = d;
   nextPtr = ptr;
}

template < class T >
T StackNode< T >::getData() const { return data; }

#endif
```

```
// STACK2
// Definition of class Stack2
#ifndef STACK2_H
#define STACK2_H

#include <iostream>

using std::cout;

#include <cassert>
#include "stacknd.h"
#include "stack.h"

template < class T >
class Stack2 : public Stack< T > {
public:
   T stackTop() const;   // check the top value
};

template < class T >
T Stack2< T >::stackTop() const
{ return !isEmpty() ? getTopPtr() -> getData() : static_cast< T >( 0 ); }

#endif
```

```
// Exercise 15.12 Solution
// Infix to postfix conversion
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cctype>
#include <cstring>
#include "stack2.h"

void convertToPostfix( char * const, char * const );
bool isOperator( char );
bool precedence( char, char );

int main()
{
   const int MAXSIZE = 100;
   char c, inFix[ MAXSIZE ], postFix[ MAXSIZE ];
   int pos = 0;

   cout << "Enter the infix expression.\n";

   while ( ( c = static_cast< char >( cin.get() ) ) != '\n' )
      if ( c != ' ' )
         inFix[ pos++ ] = c;

   inFix[ pos ] = '\0';

   cout << "The original infix expression is:\n" << inFix << '\n';
   convertToPostfix( inFix, postFix );
   cout << "The expression in postfix notation is:\n" << postFix << endl;

   return 0;
}

void convertToPostfix( char * const infix, char * const postfix )
{
   Stack2< char > charStack;
   int infixCount, postfixCount;
   bool higher;
   char popValue, leftParen = '(';

   // push a left paren onto the stack and add a right paren to infix
   charStack.push( leftParen );
   charStack.print();
   strcat( infix, ")" );

   // convert the infix expression to postfix
   for ( infixCount = 0, postfixCount = 0; charStack.stackTop();
         ++infixCount ) {

      if ( isdigit( infix[ infixCount ] ) )
         postfix[ postfixCount++ ] = infix[ infixCount ];
      else if ( infix[ infixCount ] == '(' ) {
         charStack.push( leftParen );
         charStack.print();
      }
      else if ( isOperator( infix[ infixCount ] ) ) {
         higher = true;   // used to store value of precedence test

         while ( higher ) {
            if ( isOperator( charStack.stackTop() ) )
               if ( precedence( charStack.stackTop(), infix[ infixCount ] ) ) {
                  postfix[ postfixCount++ ] = charStack.pop();
                  charStack.print();
               }
               else
                  higher = false;
            else
               higher = false;
         }

         charStack.push( infix[ infixCount ] );
         charStack.print();
      }
      else if ( infix[ infixCount ] == ')' ) {
         while ( ( popValue = charStack.pop() ) != '(' ) {
            charStack.print();
            postfix[ postfixCount++ ] = popValue;
         }

         charStack.print();
      }
   }

   postfix[ postfixCount ] = '\0';
}

// check if c is an operator
bool isOperator( char c )
{
   if ( c == '+' || c == '-' || c == '*' || c == '/' || c == '^' )
      return true;
   else
      return false;
}

bool precedence( char operator1, char operator2 )
{
   if ( operator1 == '^' )
      return true;
   else if ( operator2 == '^' )
      return false;
   else if ( operator1 == '*' || operator1 == '/' )
      return true;
   else if ( operator1 == '+' || operator1 == '-' )
      if ( operator2 == '*' || operator2 == '/' )
         return false;
      else
         return true;

   return false;
}
```

```
Enter the infix expression.
(6 + 2) * 5 - 8 / 4
The original infix expression is:
(6+2)*5-8/4
The stack is:
(
The stack is:
( (
The stack is:
+ ( (
The stack is:
( (
The stack is:
(
The stack is:
* (
The stack is:
(
The stack is:
- (
The stack is:
/ - (
The stack is:
- (
The stack is:
(
Stack is empty
The expression in postfix notation is:
62+5*84/-
```

15.13 Write a program that evaluates a postfix expression (assume it is valid) such as

6 2 + 5 * 8 4 / -

The program should read a postfix expression consisting of digits and operators into a character array. Using modified versions of the stack functions implemented earlier in this chapter, the program should scan the expression and evaluate it. The algorithm is as follows:

```
1) Append the null character ('\0') to the end of the postfix expression. When the null character is encountered, no further processing is necessary.
2) While '\0' has not been encountered, read the expression from left to right.
      If the current character is a digit,
            Push its integer value onto the stack (the integer value of a digit character is its
                  value in the computer's character set minus the value of '0' in the
                  computer's character set).
      Otherwise, if the current character is an operator,
            Pop the two top elements of the stack into variables x and y.
            Calculate y operator x.
            Push the result of the calculation onto the stack.
3) When the null character is encountered in the expression, pop the top value of the stack. This is the result of the postfix expression.
```

Note: In step 2) above, if the operator is **'/'**, the top of the stack is **2** and the next element in the stack is **8**, then pop **2** into **x**, pop **8** into **y**, evaluate **8 / 2** and push the result, **4**, back onto the stack. This note also applies to operator **'-'**. The arithmetic operations allowed in an expression are:

+	addition
-	subtraction
*****	multiplication
/	division
^	exponentiation

`%` modulus

The stack should be maintained with stack nodes that contain an **`int`** data member and a pointer to the next stack node. You may want to provide the following functional capabilities:

a) Function **`evaluatePostfixExpression`** that evaluates the postfix expression.
b) Function **`calculate`** that evaluates the expression **`op1 operator op2`**
c) Function **`push`** that pushes a value onto the stack.
d) Function **`pop`** that pops a value off the stack.
e) Function **`isEmpty`** that determines if the stack is empty.
f) Function **`printStack`** that prints the stack.

```
// STACK.H
// Definition of class Stack
// NOTE: This Stack class is a standalone Stack class template.
#ifndef STACK_H
#define STACK_H

#include <iostream>

using std::cout;

#include <cassert>
#include "stacknd.h"

template < class T >
class Stack {
public:
   Stack();              // default constructor
   ~Stack();             // destructor
   void push( T & );     // insert item in stack
   T pop();              // remove item from stack
   bool isEmpty() const; // is the stack empty?
   void print() const;   // output the stack
   StackNode< T > *getTopPtr() const { return topPtr; }
private:
   StackNode< T > *topPtr;    // pointer to fist StackNode
};

// Member function definitions for class Stack
template < class T >
Stack< T >::Stack() { topPtr = 0; }

template < class T >
Stack< T >::~Stack()
{
   StackNode< T > *tempPtr, *currentPtr = topPtr;

   while ( currentPtr != 0 ) {
      tempPtr = currentPtr;
      currentPtr = currentPtr -> getNextPtr();
      delete tempPtr;
   }
}

template < class T >
void Stack< T >::push( T &d )
{
   StackNode< T > *newPtr = new StackNode< T >( d, topPtr );

   assert( newPtr != 0 );  // was memory allocated?
   topPtr = newPtr;
}
```

```

template < class T >
T Stack< T >::pop()
{
   assert( !isEmpty() );

   StackNode< T > *tempPtr = topPtr;

   topPtr = topPtr -> nextPtr;
   T poppedValue = tempPtr -> data;
   delete tempPtr;
   return poppedValue;
}

template < class T >
bool Stack< T >::isEmpty() const { return topPtr == 0; }

template < class T >
void Stack< T >::print() const
{
   StackNode< T > *currentPtr = topPtr;

   if ( isEmpty() )          // Stack is empty
      cout << "Stack is empty\n";
   else {                    // Stack is not empty
      cout << "The stack is:\n";

      while ( currentPtr != 0 ) {
         cout << currentPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
      }

      cout << '\n';
   }
}

#endif
```

```
// STACKND.H
// Definition of template class StackNode
#ifndef STACKND_H
#define STACKND_H

template< class T > class Stack;  // forward declaration

template < class T >
class StackNode {
   friend class Stack< T >;
public:
   StackNode( const T & = 0, StackNode * = 0 );
   T getData() const;
   void setNextPtr( StackNode *nPtr ) { nextPtr = nPtr; }
   StackNode *getNextPtr() const { return nextPtr; }
private:
   T data;
   StackNode *nextPtr;
};

// Member function definitions for class StackNode
template < class T >
StackNode< T >::StackNode( const T &d, StackNode< T > *ptr )
```

```
{
   data = d;
   nextPtr = ptr;
}

template < class T >
T StackNode< T >::getData() const { return data; }

#endif
```

```
// Exercise 15.13 Solution
// Using a stack to evaluate an expression in postfix notation
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>
using std::string;

#include <cctype>
#include <cmath>
#include "stack.h"

int evaluatePostfixExpression( char * const );
int calculate( int, int, char );

int main()
{
   char expression[ 100 ], c;
   int answer, i = 0;

   cout << "Enter a postfix expression:\n";

   while ( ( c = static_cast< char >( cin.get() ) ) != '\n')
      if ( c != ' ' )
         expression[ i++ ] = c;

   expression[ i ] = '\0';

   answer = evaluatePostfixExpression( expression );
   cout << "The value of the expression is: " << answer << endl;

   return 0;
}

int evaluatePostfixExpression( char * const expr )
{
   int i, popVal1, popVal2, pushVal;
   Stack< int > intStack;
   char c;

   strcat( expr, ")" );

   for ( i = 0; ( c = expr[ i ] ) != ')'; ++i )
      if ( isdigit( expr[ i ] ) ) {
         pushVal = c - '0';
         intStack.push( pushVal );
         intStack.print();
      }
      else {
         popVal2 = intStack.pop();
         intStack.print();
         popVal1 = intStack.pop();
         intStack.print();
         pushVal = calculate( popVal1, popVal2, expr[ i ] );
         intStack.push( pushVal );
         intStack.print();
      }

   return intStack.pop();
}

int calculate( int op1, int op2, char oper )
{
   switch( oper ) {
      case '+':
         return op1 + op2;
      case '-':
         return op1 - op2;
      case '*':
         return op1 * op2;
      case '/':
         return op1 / op2;
      case '^':   // exponentiation
         return static_cast< int >( pow( op1, op2 ) );
   }

   return 0;
}
```

```
 Enter a postfix expression:
6 2 + 5 * 8 4 / -
The stack is:
6
The stack is:
2 6
The stack is:
6
Stack is empty
The stack is:
8
The stack is:
5 8
The stack is:
8
Stack is empty
The stack is:
40
The stack is:
8 40
The stack is:
4 8 40
The stack is:
8 40
The stack is:
40
The stack is:
2 40
The stack is:
40
Stack is empty
The stack is:
38
The value of the expression is: 38
```

15.14 Modify the postfix evaluator program of Exercise 15.13 so that it can process integer operands larger than 9.

15.15 *(Supermarket simulation)* Write a program that simulates a checkout line at a supermarket. The line is a queue object. Customers (i.e., customer objects) arrive in random integer intervals of 1 to 4 minutes. Also, each customer is served in random integer intervals of 1 to 4 minutes. Obviously, the rates need to be balanced. If the average arrival rate is larger than the average service rate, the queue will grow infinitely. Even with "balanced" rates, randomness can still cause long lines. Run the supermarket simulation for a 12-hour day (720 minutes) using the following algorithm:

```
1) Choose a random integer between 1 and 4 to determine the minute at which the first customer arrives.
2) At the first customer's arrival time:
   Determine customer's service time (random integer from 1 to 4);
   Begin servicing the customer;
   Schedule arrival time of next customer (random integer 1 to 4 added to the current time).
3) For each minute of the day:
       If the next customer arrives,
           Say so,
           Enqueue the customer;
           Schedule the arrival time of the next customer;
       If service was completed for the last customer;
           Say so
           Dequeue next customer to be serviced
           Determine customer's service completion time
               (random integer from 1 to 4 added to the current time).
```

Now run your simulation for 720 minutes and answer each of the following:

a) What is the maximum number of customers in the queue at any time?

b) What is the longest wait any one customer experiences?
c) What happens if the arrival interval is changed from 1-to-4 minutes to 1-to-3 minutes?

15.16 Modify the program of Fig. 15.16 to allow the binary tree object to contain duplicates.

```
// TREE
// Definition of template class Tree
#ifndef TREE_H
#define TREE_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"

template< class NODETYPE >
class Tree {
public:
   Tree();
   void insertNode( const NODETYPE & );
   void preOrderTraversal() const;
   void inOrderTraversal() const;
   void postOrderTraversal() const;
protected:
   TreeNode<NODETYPE> *rootPtr;

   // utility functions
   void insertNodeHelper( TreeNode< NODETYPE > **, const NODETYPE & );
   void preOrderHelper( TreeNode< NODETYPE > * ) const;
   void inOrderHelper( TreeNode< NODETYPE > * ) const;
   void postOrderHelper( TreeNode< NODETYPE > * ) const;
};

template< class NODETYPE >
Tree< NODETYPE >::Tree() { rootPtr = 0; }

template< class NODETYPE >
void Tree< NODETYPE >::insertNode( const NODETYPE &value )
   { insertNodeHelper( &rootPtr, value ); }

// This function receives a pointer to a pointer so the
// pointer can be modified.
// NOTE: THIS FUNCTION WAS MODIFIED TO ALLOW DUPLICATES.
template< class NODETYPE >
void Tree< NODETYPE >::insertNodeHelper( TreeNode< NODETYPE > **ptr,
                                         const NODETYPE &value )
{
   if ( *ptr == 0 ) {                     // tree is empty
      *ptr = new TreeNode< NODETYPE >( value );
      assert( *ptr != 0 );
   }
   else                                // tree is not empty
      if ( value <= ( *ptr ) -> data )
         insertNodeHelper( &( ( *ptr ) -> leftPtr ), value );
      else
         insertNodeHelper( &( ( *ptr ) -> rightPtr ), value );
}

template< class NODETYPE >
void Tree< NODETYPE >::preOrderTraversal() const { preOrderHelper( rootPtr ); }
```

```

template< class NODETYPE >
void Tree< NODETYPE >::preOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      cout << ptr -> data << ' ';
      preOrderHelper( ptr -> leftPtr );
      preOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::inOrderTraversal() const { inOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::inOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      inOrderHelper( ptr -> leftPtr );
      cout << ptr -> data << ' ';
      inOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderTraversal() const { postOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::postOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      postOrderHelper( ptr -> leftPtr );
      postOrderHelper( ptr -> rightPtr );
      cout << ptr -> data << ' ';
   }
}

#endif
```

```
// TREENODE.H
// Definition of class TreeNode
#ifndef TREENODE_H
#define TREENODE_H

template< class T > class Tree;    // forward declaration

template< class NODETYPE >
class TreeNode {
   friend class Tree< NODETYPE >;
public:
   TreeNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;      // return data
   TreeNode *getLeftPtr() const { return leftPtr; }
   TreeNode *getRightPtr() const { return rightPtr; }
   void setLeftPtr( TreeNode *ptr ) { leftPtr = ptr; }
   void setRightPtr( TreeNode *ptr ) { rightPtr = ptr; }
private:
   TreeNode *leftPtr;   // pointer to left subtree
   NODETYPE data;
   TreeNode *rightPtr;  // pointer to right subtree
};

// Constructor
template< class NODETYPE >
TreeNode< NODETYPE >::TreeNode( const NODETYPE &d )
{
   data = d;
   leftPtr = rightPtr = 0;
}

//Return a copy of the data value
template< class NODETYPE >
NODETYPE TreeNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// Exercise 15.16 solution
// Driver to test class Tree
#include <iostream>

using std::cout;
using std::cin;
using std::ios;

#include <iomanip>

using std::setprecision;
using std::setiosflags;

#include "tree.h"

int main()
{
   Tree< int > intTree;
   int intVal, i;

   cout << "Enter 10 integer values:\n";
   for ( i = 0; i < 10; ++i ) {
      cin >> intVal;
      intTree.insertNode( intVal );
   }

   cout << "\nPreorder traversal\n";
   intTree.preOrderTraversal();

   cout << "\nInorder traversal\n";
   intTree.inOrderTraversal();

   cout << "\nPostorder traversal\n";
   intTree.postOrderTraversal();

   Tree< double > doubleTree;
   double doubleVal;

   cout << "\n\n\nEnter 10 double values:\n"
        << setiosflags( ios::fixed | ios::showpoint )
        << setprecision( 1 );
   for ( i = 0; i < 10; ++i ) {
      cin >> doubleVal;
      doubleTree.insertNode( doubleVal );
   }

   cout << "\nPreorder traversal\n";
   doubleTree.preOrderTraversal();

   cout << "\nInorder traversal\n";
   doubleTree.inOrderTraversal();

   cout << "\nPostorder traversal\n";
   doubleTree.postOrderTraversal();

   return 0;
}
```

```
Enter 10 integer values:
22 8 88 22 73 88 83 68 99 92

Preorder traversal
22 8 22 88 73 68 88 83 99 92
Inorder traversal
8 22 22 68 73 83 88 88 92 99
Postorder traversal
22 8 68 83 88 73 92 99 88 22


Enter 10 double values:
9.9 2.2 8.8 2.2 0.0 6.0 3.4 9.8 5.2 6.3

Preorder traversal
9.9 2.2 2.2 0.0 8.8 6.0 3.4 5.2 6.3 9.8
Inorder traversal
0.0 2.2 2.2 3.4 5.2 6.0 6.3 8.8 9.8 9.9
Postorder traversal
0.0 2.2 5.2 3.4 6.3 6.0 9.8 8.8 2.2 9.9
```

15.17 Write a program based on Fig. 15.16 that inputs a line of text, tokenizes the sentence into separate words (you may want to use the **strtok** library function), inserts the words in a binary search tree and prints the inorder, preorder and postorder traversals of the tree. Use an OOP approach.

```
// TREE
// Definition of template class Tree
#ifndef TREE_H
#define TREE_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"

template< class NODETYPE >
class Tree {
public:
   Tree();
   void insertNode( const NODETYPE & );
   void preOrderTraversal() const;
   void inOrderTraversal() const;
   void postOrderTraversal() const;
protected:
   TreeNode<NODETYPE> *rootPtr;

   // utility functions
   void insertNodeHelper( TreeNode< NODETYPE > **, const NODETYPE & );
   void preOrderHelper( TreeNode< NODETYPE > * ) const;
   void inOrderHelper( TreeNode< NODETYPE > * ) const;
   void postOrderHelper( TreeNode< NODETYPE > * ) const;
};

template< class NODETYPE >
Tree< NODETYPE >::Tree() { rootPtr = 0; }

template< class NODETYPE >
void Tree< NODETYPE >::insertNode( const NODETYPE &value )
   { insertNodeHelper( &rootPtr, value ); }

// This function receives a pointer to a pointer so the
// pointer can be modified.
// NOTE: THIS FUNCTION WAS MODIFIED TO ALLOW DUPLICATES.
template< class NODETYPE >
void Tree< NODETYPE >::insertNodeHelper( TreeNode< NODETYPE > **ptr,
                                          const NODETYPE &value )
{
   if ( *ptr == 0 ) {                   // tree is empty
      *ptr = new TreeNode< NODETYPE >( value );
      assert( *ptr != 0 );
   }
   else                              // tree is not empty
      if ( value <= ( *ptr ) -> data )
         insertNodeHelper( &( ( *ptr ) -> leftPtr ), value );
      else
         insertNodeHelper( &( ( *ptr ) -> rightPtr ), value );
}

template< class NODETYPE >
void Tree< NODETYPE >::preOrderTraversal() const { preOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::preOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      cout << ptr -> data << ' ';
      preOrderHelper( ptr -> leftPtr );
      preOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::inOrderTraversal() const { inOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::inOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      inOrderHelper( ptr -> leftPtr );
      cout << ptr -> data << ' ';
      inOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
```

```
void Tree< NODETYPE >::postOrderTraversal() const { postOrderHelper( rootPtr );}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      postOrderHelper( ptr -> leftPtr );
      postOrderHelper( ptr -> rightPtr );
      cout << ptr -> data << ' ';
   }
}

#endif
```

```
// TREENODE.H
// Definition of class TreeNode
#ifndef TREENODE_H
#define TREENODE_H

template< class T > class Tree;    // forward declaration

template< class NODETYPE >
class TreeNode {
   friend class Tree< NODETYPE >;
public:
   TreeNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;      // return data
   TreeNode *getLeftPtr() const { return leftPtr; }
   TreeNode *getRightPtr() const { return rightPtr; }
   void setLeftPtr( TreeNode *ptr ) { leftPtr = ptr; }
   void setRightPtr( TreeNode *ptr ) { rightPtr = ptr; }
private:
   TreeNode *leftPtr;   // pointer to left subtree
   NODETYPE data;
   TreeNode *rightPtr;  // pointer to right subtree
};

// Constructor
template< class NODETYPE >
TreeNode< NODETYPE >::TreeNode( const NODETYPE &d )
{
   data = d;
   leftPtr = rightPtr = 0;
}

//Return a copy of the data value
template< class NODETYPE >
NODETYPE TreeNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// STRING2.H
// Definition of a String class
#ifndef STRING1_H
#define STRING1_H

#include <iostream>

using std::ostream;
using std::istream;

class String {
   friend ostream &operator<<( ostream &, const String & );
   friend istream &operator>>( istream &, String & );
public:
   String( const char * = "" ); // conversion constructor
   String( const String & );    // copy constructor
   ~String();                   // destructor
   const String &operator=( const String & );  // assignment
   String &operator+=( const String & );     // concatenation
   bool operator!() const;                   // is String empty?
   bool operator==( const String & ) const;  // test s1 == s2
   bool operator!=( const String & ) const;  // test s1 != s2
   bool operator<( const String & )  const;  // test s1 < s2
   bool operator>( const String & )  const;  // test s1 > s2
   bool operator>=( const String & ) const;  // test s1 >= s2
   bool operator<=( const String & ) const;  // test s1 <= s2
   char &operator[]( int );        // return char reference
   String &operator()( int, int ); // return a substring
   int getLength() const;        // return string length
private:
   char *sPtr;                   // pointer to start of string
   int length;                   // string length
};

#endif
```

```
// STRING2.CPP
// Member function definitions for class String.
// NOTE: The printing capabilities have been removed
// from the constructor and destructor functions.
#include <iostream>

using std::cout;
using std::ostream;
using std::istream;

#include <iomanip>

using std::setw;

#include <cstring>
using std::string;

#include <cassert>
#include "string2.h"

// Conversion constructor: Convert char * to String
String::String( const char *ptr )
{
   length = strlen( ptr );           // compute length
   sPtr = new char[ length + 1 ];     // allocate storage
   assert( sPtr != 0 );              // terminate if memory not allocated
   strcpy( sPtr, ptr );                // copy literal to object
}

// Copy constructor
String::String( const String &copy )
{
   length = copy.length;        // copy length
   sPtr = new char[ length + 1 ]; // allocate storage
   assert( sPtr != 0 );           // ensure memory allocated
   strcpy( sPtr, copy.sPtr );     // copy string
}

// Destructor
String::~String()
{
   delete [] sPtr;              // reclaim string
}

// Overloaded = operator; avoids self assignment
const String &String::operator=( const String &right )
{
   if ( &right != this ) {           // avoid self assignment
      delete [] sPtr;                // prevents memory leak
      length = right.length;         // new String length
      sPtr = new char[ length + 1 ]; // allocate memory
      assert( sPtr != 0 );           // ensure memory allocated
      strcpy( sPtr, right.sPtr );    // copy string
   }
   else
      cout << "Attempted assignment of a String to itself\n";

   return *this;   // enables concatenated assignments
}

// Concatenate right operand to this object and
// store in this object.
String &String::operator+=( const String &right )
{
   char *tempPtr = sPtr;          // hold to be able to delete
   length += right.length;        // new String length
   sPtr = new char[ length + 1 ]; // create space
   assert( sPtr != 0 );           // terminate if memory not allocated
   strcpy( sPtr, tempPtr );       // left part of new String
   strcat( sPtr, right.sPtr );    // right part of new String
   delete [] tempPtr;             // reclaim old space
   return *this;                  // enables concatenated calls
}

// Is this String empty?
bool String::operator!() const { return length == 0; }

// Is this String equal to right String?
bool String::operator==( const String &right ) const
   { return strcmp( sPtr, right.sPtr ) == 0; }

// Is this String not equal to right String?
bool String::operator!=( const String &right ) const
   { return strcmp( sPtr, right.sPtr ) != 0; }

// Is this String less than right String?
bool String::operator<( const String &right ) const
   { return strcmp( sPtr, right.sPtr ) < 0; }

// Is this String greater than right String?
bool String::operator>( const String &right ) const
   { return strcmp( sPtr, right.sPtr ) > 0; }

// Is this String greater than or equal to right String?
bool String::operator>=( const String &right ) const
```

```
   { return strcmp( sPtr, right.sPtr ) >= 0; }

// Is this String less than or equal to right String?
bool String::operator<=( const String &right ) const
   { return strcmp( sPtr, right.sPtr ) <= 0; }

// Return a reference to a character in a String.
char &String::operator[]( int subscript )
{
   // First test for subscript out of range
   assert( subscript >= 0 && subscript < length );

   return sPtr[ subscript ];  // creates lvalue
}

// Return a substring beginning at index and
// of length subLength as a reference to a String object.
String &String::operator()( int index, int subLength )
{
   // ensure index is in range and substring length >= 0
   assert( index >= 0 && index < length && subLength >= 0 );

   String *subPtr = new String;   // empty String
   assert( subPtr != 0 );     // ensure new String allocated

   // determine length of substring
   if ( ( subLength == 0 ) || ( index + subLength > length ) )
      subPtr -> length = length - index + 1;
   else
      subPtr -> length = subLength + 1;

   // allocate memory for substring
   delete subPtr -> sPtr;       // delete character from object
   subPtr -> sPtr = new char[ subPtr -> length ];
   assert( subPtr -> sPtr != 0 ); // ensure space allocated

   // copy substring into new String
   strncpy( subPtr -> sPtr, &sPtr[ index ], subPtr -> length );
   subPtr -> sPtr[ subPtr -> length ] = '\0'; // terminate new String

   return *subPtr;           // return new String
}

// Return string length
int String::getLength() const { return length; }

// Overloaded output operator
ostream &operator<<( ostream &output, const String &s )
{
   output << s.sPtr;
   return output;   // enables concatenation
}

// Overloaded input operator
istream &operator>>( istream &input, String &s )
{
   char temp[ 100 ];   // buffer to store input

   input >> setw( 100 ) >> temp;
   s = temp;        // use String class assignment operator
   return input;    // enables concatenation
}
```

```
// Exercise 15.17 solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstring>
using std::string;

#include "tree.h"
#include "string2.h"

int main()
{
   Tree< String > stringTree;
   char sentence[ 80 ], *tokenPtr;

   cout << "Enter a sentence:\n";
   cin.getline( sentence, 80 );

   tokenPtr = strtok( sentence, " " );

   while ( tokenPtr != 0 ) {
      String *newString = new String( tokenPtr );
      stringTree.insertNode( *newString );
      tokenPtr = strtok( 0, " " );
   }

   cout << "\nPreorder traversal\n";
   stringTree.preOrderTraversal();

   cout << "\nInorder traversal\n";
   stringTree.inOrderTraversal();

   cout << "\nPostorder traversal\n";
   stringTree.postOrderTraversal();

   cout << endl;

   return 0;
}
```

```
Enter a sentence:
ANSI/ISO C++ How to Program

Preorder traversal
ANSI/ISO C++ How to Program
Inorder traversal
ANSI/ISO C++ How Program to
Postorder traversal
Program to How C++ ANSI/ISO
```

15.18 In this chapter, we saw that duplicate elimination is straightforward when creating a binary search tree. Describe how you would perform duplicate elimination using only a single-subscripted array. Compare the performance of array-based duplicate elimination with the performance of binary-search-tree-based duplicate elimination.

15.19 Write a function `depth` that receives a binary tree and determines how many levels it has.

```
// TREE
// Definition of template class Tree
#ifndef TREE_H
#define TREE_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"

template< class NODETYPE >
class Tree {
public:
   Tree();
   void insertNode( const NODETYPE & );
   void preOrderTraversal() const;
   void inOrderTraversal() const;
   void postOrderTraversal() const;
protected:
   TreeNode<NODETYPE> *rootPtr;

   // utility functions
   void insertNodeHelper( TreeNode< NODETYPE > **, const NODETYPE & );
   void preOrderHelper( TreeNode< NODETYPE > * ) const;
   void inOrderHelper( TreeNode< NODETYPE > * ) const;
   void postOrderHelper( TreeNode< NODETYPE > * ) const;
};

template< class NODETYPE >
Tree< NODETYPE >::Tree() { rootPtr = 0; }

template< class NODETYPE >
void Tree< NODETYPE >::insertNode( const NODETYPE &value )
   { insertNodeHelper( &rootPtr, value ); }

// This function receives a pointer to a pointer so the
// pointer can be modified.
// NOTE: THIS FUNCTION WAS MODIFIED TO ALLOW DUPLICATES.
template< class NODETYPE >
void Tree< NODETYPE >::insertNodeHelper( TreeNode< NODETYPE > **ptr,
                                         const NODETYPE &value )
{
   if ( *ptr == 0 ) {                   // tree is empty
      *ptr = new TreeNode< NODETYPE >( value );
      assert( *ptr != 0 );
   }
   else                              // tree is not empty
      if ( value <= ( *ptr ) -> data )
         insertNodeHelper( &( ( *ptr ) -> leftPtr ), value );
      else
         insertNodeHelper( &( ( *ptr ) -> rightPtr ), value );
}

template< class NODETYPE >
void Tree< NODETYPE >::preOrderTraversal() const { preOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::preOrderHelper( TreeNode< NODETYPE > *ptr ) const
```

```
{
   if ( ptr != 0 ) {
      cout << ptr -> data << ' ';
      preOrderHelper( ptr -> leftPtr );
      preOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::inOrderTraversal() const { inOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::inOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      inOrderHelper( ptr -> leftPtr );
      cout << ptr -> data << ' ';
      inOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderTraversal() const { postOrderHelper( rootPtr );
}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      postOrderHelper( ptr -> leftPtr );
      postOrderHelper( ptr -> rightPtr );
      cout << ptr -> data << ' ';
   }
}

#endif
```

```
// TREENODE.H
// Definition of class TreeNode
#ifndef TREENODE_H
#define TREENODE_H

template< class T > class Tree;    // forward declaration

template< class NODETYPE >
class TreeNode {
   friend class Tree< NODETYPE >;
public:
   TreeNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;      // return data
   TreeNode *getLeftPtr() const { return leftPtr; }
   TreeNode *getRightPtr() const { return rightPtr; }
   void setLeftPtr( TreeNode *ptr ) { leftPtr = ptr; }
   void setRightPtr( TreeNode *ptr ) { rightPtr = ptr; }
private:
   TreeNode *leftPtr;   // pointer to left subtree
   NODETYPE data;
   TreeNode *rightPtr;  // pointer to right subtree
};

// Constructor
```

```
template< class NODETYPE >
TreeNode< NODETYPE >::TreeNode( const NODETYPE &d )
{
   data = d;
   leftPtr = rightPtr = 0;
}

//Return a copy of the data value
template< class NODETYPE >
NODETYPE TreeNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// TREE2.H
// Definition of template class Tree
// Modified to include getDepth and determineDepth member functions.
#ifndef TREE2_H
#define TREE2_H

#include <cassert>
#include "treenode.h"
#include "tree.h"

template< class NODETYPE >
class Tree2 : public Tree< NODETYPE > {
public:
   int getDepth() const;
private:
   void determineDepth( TreeNode< NODETYPE > *, int *, int * ) const;
};

template< class NODETYPE >
int Tree2< NODETYPE >::getDepth() const
{
   int totalDepth = 0, currentDepth = 0;

   determineDepth( rootPtr, &totalDepth, &currentDepth );

   return totalDepth;
}

template< class NODETYPE >
void Tree2< NODETYPE >::determineDepth( TreeNode< NODETYPE > *ptr,
                                        int *totPtr, int *currPtr ) const
{
   if ( ptr != 0 ) {
      ++( *currPtr );

      if ( *currPtr > *totPtr )
         *totPtr = *currPtr;

      determineDepth( ptr -> getLeftPtr(), totPtr, currPtr );
      determineDepth( ptr -> getRightPtr(), totPtr, currPtr );
      --( *currPtr );
   }
}

#endif
```

```
// Exercise 15.19 solution
#include <iostream>

using std::cout;
using std::cin;

#include "tree2.h"

int main()
{
   Tree2< int > intTree;
   int intVal;

   cout << "Enter 10 integer values:\n";

   for ( int i = 0; i < 10; ++i ) {
      cin >> intVal;
      intTree.insertNode( intVal );
   }

   cout << "\nPreorder traversal\n";
   intTree.preOrderTraversal();

   cout << "\nInorder traversal\n";
   intTree.inOrderTraversal();

   cout << "\nPostorder traversal\n";
   intTree.postOrderTraversal();

   cout << "\n\nThere are " << intTree.getDepth()
        << " levels in this binary tree\n";

   return 0;
}
```

```
Enter 10 integer values:
1 2 3 88 4 6 0 22 21 10

Preorder traversal
1 0 2 3 88 4 6 22 21 10
Inorder traversal
0 1 2 3 4 6 10 21 22 88
Postorder traversal
0 10 21 22 6 4 88 3 2 1

There are 9 levels in this binary tree
```

15.20 (*Recursively print a list backwards*) Write a member function `printListBackwards` that recursively outputs the items in a linked list object in reverse order. Write a test program that creates a sorted list of integers and prints the list in reverse order.

```
// LIST
// Template List class definition
// Added copy constructor to member functions (not included in chapter).
#ifndef LIST_H
#define LIST_H

#include <iostream>

```

```
using std::cout;

#include <cassert>
#include "Listnd.h"

template< class NODETYPE >
class List {
public:
   List();                              // default constructor
   List( const List< NODETYPE > & );  // copy constructor
   ~List();                             // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}
```

```

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
}

// Insert a node at the back of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )             // List is empty
      return false;             // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}

// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;               // data
   ListNode *nextPtr;           // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// LIST2.H
// Template List class definition
#ifndef LIST2_H
#define LIST2_H

#include <iostream>

using std::cout;

#include <cassert>
#include "listnd.h"
#include "list.h"

template< class NODETYPE >
class List2 : public List< NODETYPE > {
public:
   void recursivePrintReverse() const;
private:
   void recursivePrintReverseHelper( ListNode< NODETYPE > * ) const;
};

// Print a List backwards recursively.
template< class NODETYPE >
void List2< NODETYPE >::recursivePrintReverse() const
{
   cout << "The list printed recursively backwards is:\n";
   recursivePrintReverseHelper( firstPtr );
   cout << '\n';
}

// Helper for printing a list backwards recursively.
template< class NODETYPE >
void List2< NODETYPE >::recursivePrintReverseHelper(
                          ListNode< NODETYPE > *currentPtr ) const
{
   if ( currentPtr == 0 )
      return;

   recursivePrintReverseHelper( currentPtr -> getNextPtr() );
   cout << currentPtr -> getData() << ' ';
}

#endif
```

```
// Exercise 15.20 solution

#include "list2.h"

int main()
{
   List2< int > intList;

   for ( int i = 1; i <= 10; ++i )
      intList.insertAtBack( i );

   intList.print();
   intList.recursivePrintReverse();

   return 0;
}
```

```
The list is: 1 2 3 4 5 6 7 8 9 10

The list printed recursively backwards is:
10 9 8 7 6 5 4 3 2 1
Destroying nodes ...
1 2 3 4 5 6 7 8 9 10
All nodes destroyed
```

15.21 (*Recursively search a list*) Write a member function **searchList** that recursively searches a linked list object for a specified value. The function should return a pointer to the value if it is found; otherwise, null should be returned. Use your function in a test program that creates a list of integers. The program should prompt the user for a value to locate in the list.

```
// LIST
// Template List class definition
// Added copy constructor to member functions (not included in chapter).
#ifndef LIST_H
#define LIST_H

#include <iostream>

using std::cout;

#include <cassert>
#include "Listnd.h"

template< class NODETYPE >
class List {
public:
   List();                             // default constructor
   List( const List< NODETYPE > & );   // copy constructor
   ~List();                            // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
}

```

```
// Insert a node at the back of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {            // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )              // List is empty
      return false;              // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}

```

```
// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;               // data
   ListNode *nextPtr;           // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// LIST2.H
// Template List class definition
#ifndef LIST2_H
#define LIST2_H

#include <iostream>

using std::cout;

#include <cassert>
#include "listnd.h"
#include "list.h"

template< class NODETYPE >
class List2 : public List< NODETYPE > {
public:
   void recursivePrintReverse() const;
   NODETYPE *recursiveSearch( NODETYPE & ) const;
private:
   // Utility functions
   void recursivePrintReverseHelper( ListNode< NODETYPE > * ) const;
   NODETYPE *recursiveSearchHelper( ListNode< NODETYPE > *, NODETYPE & ) const;
};

// Print a List backwards recursively.
template< class NODETYPE >
void List2< NODETYPE >::recursivePrintReverse() const
{
   cout << "The list printed recursively backwards is:\n";
   recursivePrintReverseHelper( firstPtr );
   cout << '\n';
}

// Helper for printing a list backwards recursively.
template< class NODETYPE >
void List2< NODETYPE >::recursivePrintReverseHelper(
                          ListNode< NODETYPE > *currentPtr ) const
{
   if ( currentPtr == 0 )
      return;

   recursivePrintReverseHelper( currentPtr -> nextPtr );
   cout << currentPtr -> data << ' ';
}

// Search a List recursively.
template< class NODETYPE >
NODETYPE *List2< NODETYPE >::recursiveSearch( NODETYPE &val ) const
   { return recursiveSearchHelper( firstPtr, val ); }

// Helper for searching a list recursively.
template< class NODETYPE >
```

```
NODETYPE *List2< NODETYPE >::recursiveSearchHelper(
                    ListNode< NODETYPE > *currentPtr, NODETYPE &value ) const
{
   if ( currentPtr == 0 )
      return 0;

   if ( currentPtr -> getData() == value )
      return currentPtr -> getAddress();

   return recursiveSearchHelper( currentPtr -> getNextPtr(), value );
}

#endif
```

```
// Exercise 15.21 solution
#include <iostream>

using std::cout;
using std::cin;

#include "list2.h"

int main()
{
   List2< int > intList;

   for ( int i = 2; i <= 20; i += 2 )
      intList.insertAtBack( i );

   intList.print();

   int value, *ptr;

   cout << "Enter a value to search for: ";
   cin >> value;
   ptr = intList.recursiveSearch( value );

   if ( ptr != 0 )
      cout << *ptr << " was found\n";
   else
      cout << "Element not found\n";

   return 0;
}
```

```
The list is: 2 4 6 8 10 12 14 16 18 20

Enter a value to search for: 14
14 was found
Destroying nodes ...
2 4 6 8 10 12 14 16 18 20
All nodes destroyed
```

15.22 (*Binary tree delete*) In this exercise, we discuss deleting items from binary search trees. The deletion algorithm is not as straightforward as the insertion algorithm. There are three cases that are encountered when deleting an item—the item is contained in a leaf node (i.e., it has no children), the item is contained in a node that has one child or the item is contained in a node that has two children.

If the item to be deleted is contained in a leaf node, the node is deleted and the pointer in the parent node is set to null.

If the item to be deleted is contained in a node with one child, the pointer in the parent node is set to point to the child nod e and the node containing the data item is deleted. This causes the child node to take the place of the deleted node in the tree.

The last case is the most difficult. When a node with two children is deleted, another node in the tree must take its place. However, the pointer in the parent node cannot be assigned to point to one of the children of the node to be deleted. In most cases, the resulting binary search tree would not adhere to the following characteristic of binary search trees (with no duplicate values): *The values in any left subtree are less than the value in the parent node, and the values in any right subtree are greater than the value in the parent node*.

Which node is used as a *replacement node* to maintain this characteristic? Either the node containing the largest value in the tree less than the value in the node being deleted, or the node containing the smallest value in the tree greater than the value in the node being deleted. Let us consider the node with the smaller value. In a binary search tree, the largest value less than a parent's value is located in the left subtree of the parent node and is guaranteed to be contained in the rightmost node of the subtree. This node is located by walking down the left subtree to the right until the pointer to the right child of the current node is null. We are now pointing to the replacement node, which is either a leaf node or a node with one child to its left. If the replacement node is a leaf node, the steps to perform the deletion are as follows:

1) Store the pointer to the node to be deleted in a temporary pointer variable (this pointer is used to delete the dynamically allocated memory)
2) Set the pointer in the parent of the node being deleted to point to the replacement node
3) Set the pointer in the parent of the replacement node to null
4) Set the pointer to the right subtree in the replacement node to point to the right subtree of the node to be deleted
5) Delete the node to which the temporary pointer variable points.

The deletion steps for a replacement node with a left child are similar to those for a replacement node with no children, but the algorithm also must move the child in to the replacement node's position in the tree. If the replacement node is a node with a left child, the steps to perform the deletion are as follows:

1) Store the pointer to the node to be deleted in a temporary pointer variable
2) Set the pointer in the parent of the node being deleted to point to the replacement node
3) Set the pointer in the parent of the replacement node to point to the left child of the replacement node
4) Set the pointer to the right subtree in the replacement node to point to the right subtree of the node to be deleted
5) Delete the node to which the temporary pointer variable points.

Write member function **`deleteNode`**, which takes as its arguments a pointer to the root node of the tree object and the value to be deleted. The function should locate in the tree the node containing the value to be deleted and use the algorithms discussed here to delete the node. If the value is not found in the tree, the function should print a message that indicates whether or not the value is deleted. Modify the program of Fig. 15.16 to use this function. After deleting an item, call the **`inOrder`**, **`preOrder`** and **`postOrder`** traversal functions to confirm that the delete operation was performed correctly.

15.23 (*Binary tree search*) Write member function **`binaryTreeSearch`**, which attempts to locate a specified value in a binary search tree object. The function should take as arguments a pointer to the root node of the binary tree and a search key to be located. If the node containing the search key is found, the function should return a pointer to that node; otherwise, the function should return a null pointer.

```
// TREE
// Definition of template class Tree
#ifndef TREE_H
#define TREE_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"

template< class NODETYPE >
class Tree {
public:
   Tree();
   void insertNode( const NODETYPE & );
   void preOrderTraversal() const;
   void inOrderTraversal() const;
   void postOrderTraversal() const;
protected:
   TreeNode<NODETYPE> *rootPtr;

   // utility functions
   void insertNodeHelper( TreeNode< NODETYPE > **, const NODETYPE & );
   void preOrderHelper( TreeNode< NODETYPE > * ) const;
   void inOrderHelper( TreeNode< NODETYPE > * ) const;
   void postOrderHelper( TreeNode< NODETYPE > * ) const;
};

template< class NODETYPE >
Tree< NODETYPE >::Tree() { rootPtr = 0; }

template< class NODETYPE >
void Tree< NODETYPE >::insertNode( const NODETYPE &value )
   { insertNodeHelper( &rootPtr, value ); }

// This function receives a pointer to a pointer so the
// pointer can be modified.
// NOTE: THIS FUNCTION WAS MODIFIED TO ALLOW DUPLICATES.
template< class NODETYPE >
void Tree< NODETYPE >::insertNodeHelper( TreeNode< NODETYPE > **ptr,
                                         const NODETYPE &value )
{
   if ( *ptr == 0 ) {                    // tree is empty
      *ptr = new TreeNode< NODETYPE >( value );
      assert( *ptr != 0 );
   }
   else                               // tree is not empty
      if ( value <= ( *ptr ) -> data )
         insertNodeHelper( &( ( *ptr ) -> leftPtr ), value );
      else
         insertNodeHelper( &( ( *ptr ) -> rightPtr ), value );
}

template< class NODETYPE >
void Tree< NODETYPE >::preOrderTraversal() const { preOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::preOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      cout << ptr -> data << ' ';
      preOrderHelper( ptr -> leftPtr );
      preOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::inOrderTraversal() const { inOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::inOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      inOrderHelper( ptr -> leftPtr );
      cout << ptr -> data << ' ';
      inOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
```

```
void Tree< NODETYPE >::postOrderTraversal() const { postOrderHelper( rootPtr );}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      postOrderHelper( ptr -> leftPtr );
      postOrderHelper( ptr -> rightPtr );
      cout << ptr -> data << ' ';
   }
}

#endif
```

```
// TREENODE.H
// Definition of class TreeNode
#ifndef TREENODE_H
#define TREENODE_H

template< class T > class Tree;    // forward declaration

template< class NODETYPE >
class TreeNode {
   friend class Tree< NODETYPE >;
public:
   TreeNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;      // return data
   TreeNode *getLeftPtr() const { return leftPtr; }
   TreeNode *getRightPtr() const { return rightPtr; }
   void setLeftPtr( TreeNode *ptr ) { leftPtr = ptr; }
   void setRightPtr( TreeNode *ptr ) { rightPtr = ptr; }
private:
   TreeNode *leftPtr;   // pointer to left subtree
   NODETYPE data;
   TreeNode *rightPtr;  // pointer to right subtree
};

// Constructor
template< class NODETYPE >
TreeNode< NODETYPE >::TreeNode( const NODETYPE &d )
{
   data = d;
   leftPtr = rightPtr = 0;
}

//Return a copy of the data value
template< class NODETYPE >
NODETYPE TreeNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// TREE2.H
// Definition of template class Tree
#ifndef TREE2_H
#define TREE2_H

#include <iostream>

using std::cout;
```

```

#include <cassert>
#include "treenode.h"
#include "tree.h"

template< class NODETYPE >
class Tree2 : public Tree< NODETYPE > {
public:
   TreeNode< NODETYPE > *binarySearch( int ) const;
private:
   TreeNode< NODETYPE > *binarySearchHelper( TreeNode< NODETYPE > *,
                                             int ) const;
};

template< class NODETYPE >
TreeNode< NODETYPE > *Tree2< NODETYPE >::binarySearch( int val ) const
   { return binarySearchHelper( rootPtr, val ); }

template< class NODETYPE >
TreeNode< NODETYPE > *Tree2< NODETYPE >::binarySearchHelper(
                           TreeNode< NODETYPE > *ptr, int value ) const
{
   if ( ptr == 0 )
      return 0;

   cout << "Comparing " << value << " to " << ptr -> getData();

   if ( value == ptr -> getData() ) {    // match
      cout << "; search complete\n";
      return ptr;
   }
   else if ( value < ptr -> getData() ) { // search val less than current data
      cout << "; smaller, walk left\n";
      return binarySearchHelper( ptr -> getLeftPtr(), value );
   }
   else {                          // search val greater than current data
      cout << "; larger, walk right\n";
      return binarySearchHelper( ptr -> getRightPtr(), value );
   }
}

#endif
```

```
// Exercise 15.23 solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>
#include <ctime>
#include "tree2.h"

int main()
{
   srand( time( 0 ) );  // randomize the random number generator

   Tree2< int > intTree;
   int intVal;

   cout << "The values being placed in the tree are:\n";

   for ( int i = 1; i <= 15; ++i ) {
      intVal = rand() % 100;
      cout << intVal << ' ';
      intTree.insertNode( intVal );
   }

   cout << "\n\nEnter a value to search for: ";
   cin >> intVal;

   TreeNode< int > *ptr = intTree.binarySearch( intVal );

   if ( ptr != 0 )
      cout << ptr -> getData() << " was found\n";
   else
      cout << "Element was not found\n";

   cout << endl;

   return 0;
}
```

```
The values being placed in the tree are:
96 96 58 31 44 0 12 61 99 32 29 92 79 55 1

Enter a value to search for: 1
Comparing 1 to 96; smaller, walk left
Comparing 1 to 96; smaller, walk left
Comparing 1 to 58; smaller, walk left
Comparing 1 to 31; smaller, walk left
Comparing 1 to 0; larger, walk right
Comparing 1 to 12; smaller, walk left
Comparing 1 to 1; search complete
1 was found
```

15.24 (*Level-order binary tree traversal*) The program of Fig. 15.16 illustrated three recursive methods of traversing a binary tree—inorder, preorder and postorder traversals. This exercise presents the *level-order traversal* of a binary tree in which the node values are printed level by level, starting at the root node level. The nodes on each level are printed from left to right. The level-order traversal is not a recursive algorithm. It uses a queue object to control the output of the nodes. The algorithm is as follows:

```
1) Insert the root node in the queue
2) While there are nodes left in the queue,
      Get the next node in the queue
      Print the node's value
      If the pointer to the left child of the node is not null
         Insert the left child node in the queue
      If the pointer to the right child of the node is not null
         Insert the right child node in the queue.
```

Write member function **levelOrder** to perform a level-order traversal of a binary tree object. Modify the program of Fig 15.16 to use this function. (Note: You will also need to modify and incorporate the queue-processing functions of Fig. 15.12 in this program.)

```
// TREE
// Definition of template class Tree
#ifndef TREE_H
#define TREE_H

```

```
#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"

template< class NODETYPE >
class Tree {
public:
   Tree();
   void insertNode( const NODETYPE & );
   void preOrderTraversal() const;
   void inOrderTraversal() const;
   void postOrderTraversal() const;
protected:
   TreeNode<NODETYPE> *rootPtr;

   // utility functions
   void insertNodeHelper( TreeNode< NODETYPE > **, const NODETYPE & );
   void preOrderHelper( TreeNode< NODETYPE > * ) const;
   void inOrderHelper( TreeNode< NODETYPE > * ) const;
   void postOrderHelper( TreeNode< NODETYPE > * ) const;
};

template< class NODETYPE >
Tree< NODETYPE >::Tree() { rootPtr = 0; }

template< class NODETYPE >
void Tree< NODETYPE >::insertNode( const NODETYPE &value )
   { insertNodeHelper( &rootPtr, value ); }

// This function receives a pointer to a pointer so the
// pointer can be modified.
// NOTE: THIS FUNCTION WAS MODIFIED TO ALLOW DUPLICATES.
template< class NODETYPE >
void Tree< NODETYPE >::insertNodeHelper( TreeNode< NODETYPE > **ptr,
                                         const NODETYPE &value )
{
   if ( *ptr == 0 ) {                    // tree is empty
      *ptr = new TreeNode< NODETYPE >( value );
      assert( *ptr != 0 );
   }
   else                               // tree is not empty
      if ( value <= ( *ptr ) -> data )
         insertNodeHelper( &( ( *ptr ) -> leftPtr ), value );
      else
         insertNodeHelper( &( ( *ptr ) -> rightPtr ), value );
}

template< class NODETYPE >
void Tree< NODETYPE >::preOrderTraversal() const { preOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::preOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      cout << ptr -> data << ' ';
      preOrderHelper( ptr -> leftPtr );
      preOrderHelper( ptr -> rightPtr );
   }
}
```

```cpp

template< class NODETYPE >
void Tree< NODETYPE >::inOrderTraversal() const { inOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::inOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      inOrderHelper( ptr -> leftPtr );
      cout << ptr -> data << ' ';
      inOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderTraversal() const { postOrderHelper( rootPtr );}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      postOrderHelper( ptr -> leftPtr );
      postOrderHelper( ptr -> rightPtr );
      cout << ptr -> data << ' ';
   }
}

#endif
```

```cpp
// TREENODE.H
// Definition of class TreeNode
#ifndef TREENODE_H
#define TREENODE_H

template< class T > class Tree;    // forward declaration

template< class NODETYPE >
class TreeNode {
   friend class Tree< NODETYPE >;
public:
   TreeNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;      // return data
   TreeNode *getLeftPtr() const { return leftPtr; }
   TreeNode *getRightPtr() const { return rightPtr; }
   void setLeftPtr( TreeNode *ptr ) { leftPtr = ptr; }
   void setRightPtr( TreeNode *ptr ) { rightPtr = ptr; }
private:
   TreeNode *leftPtr;   // pointer to left subtree
   NODETYPE data;
   TreeNode *rightPtr;  // pointer to right subtree
};

// Constructor
template< class NODETYPE >
TreeNode< NODETYPE >::TreeNode( const NODETYPE &d )
{
   data = d;
   leftPtr = rightPtr = 0;
}

//Return a copy of the data value
```

```
template< class NODETYPE >
NODETYPE TreeNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// TREE2.H
// Definition of template class Tree
#ifndef TREE2_H
#define TREE2_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"
#include "queue.h"
#include "tree.h"

template< class NODETYPE >
class Tree2 : public Tree< NODETYPE > {
public:
   void levelOrderTraversal();
};

template< class NODETYPE >
void Tree2< NODETYPE >::levelOrderTraversal()
{
   Queue< TreeNode< NODETYPE > * > queue;
   TreeNode< NODETYPE > *nodePtr;

   if ( rootPtr != 0 )
      queue.enqueue( rootPtr );

   while ( !queue.isEmpty() ) {
      nodePtr = queue.dequeue();
      cout << nodePtr -> getData() << ' ';

      if ( nodePtr -> getLeftPtr() != 0 )
         queue.enqueue( nodePtr -> getLeftPtr() );

      if ( nodePtr -> getRightPtr() != 0 )
         queue.enqueue( nodePtr -> getRightPtr() );
   }
}

#endif
```

```
// QUEUE.H
// Definition of class Queue
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>

using std::cout;

#include <new>
#include <cstdlib>
#include "queuend.h"
```

```

template < class T >
class Queue {
public:
   Queue();               // default constructor
   ~Queue();              // destructor
   void enqueue( T );     // insert item in queue
   T dequeue();           // remove item from queue
   bool isEmpty() const; // is the queue empty?
   void print() const;    // output the queue
private:
   QueueNode< T > *headPtr;  // pointer to first QueueNode
   QueueNode< T > *tailPtr;  // pointer to last QueueNode
};

// Member function definitions for class Queue
template < class T >
Queue< T >::Queue() { headPtr = tailPtr = 0; }

template < class T >
Queue< T >::~Queue()
{
   QueueNode< T > *tempPtr, *currentPtr = headPtr;

   while ( currentPtr != 0 ) {
      tempPtr = currentPtr;
      currentPtr = currentPtr -> nextPtr;
      delete tempPtr;
   }
}

template < class T >
void Queue< T >::enqueue( T d )
{
   QueueNode< T > *newPtr = new QueueNode< T >( d );
   assert( newPtr != 0 );

   if ( isEmpty() )
      headPtr = tailPtr = newPtr;
   else {
      tailPtr -> nextPtr = newPtr;
      tailPtr = newPtr;
   }
}

template < class T >
T Queue< T >::dequeue()
{
   assert( !isEmpty() );

   QueueNode< T > *tempPtr = headPtr;

   headPtr = headPtr -> nextPtr;
   T value = tempPtr -> data;
   delete tempPtr;

   if ( headPtr == 0 )
      tailPtr = 0;

   return value;
}

```

```
template < class T >
bool Queue< T >::isEmpty() const { return headPtr == 0; }

template < class T >
void Queue< T >::print() const
{
   QueueNode< T > *currentPtr = headPtr;

   if ( isEmpty() )           // Queue is empty
      cout << "Queue is empty\n";
   else {                    // Queue is not empty
      cout << "The queue is:\n";

      while ( currentPtr != 0 ) {
         cout << currentPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
      }

      cout << endl;
   }
}

#endif
```

```
// QUEUEND.H
// Definition of template class QueueNode
#ifndef QUEUEND_H
#define QUEUEND_H

template< class T > class Queue;  // forward declaration

template < class T >
class QueueNode {
   friend class Queue< T >;
public:
   QueueNode( const T & = 0 );
   T getData() const;
private:
   T data;
   QueueNode *nextPtr;
};

// Member function definitions for class QueueNode
template < class T >
QueueNode< T >::QueueNode( const T &d )
{
   data = d;
   nextPtr = 0;
}

template < class T >
T QueueNode< T >::getData() const { return data; }

#endif
```

```
// Exercise 15.24 solution
#include <iostream>
using std::cout;
using std::endl;
```

```

#include <cstdlib>
#include <ctime>
#include "tree2.h"

int main()
{
   srand( time( 0 ) );  // randomize the random number generator

   Tree2< int > intTree;
   int intVal;

   cout << "The values being placed in the tree are:\n";

   for ( int i = 1; i <= 15; ++i ) {
      intVal = rand() % 100;
      cout << intVal << ' ';
      intTree.insertNode( intVal );
   }

   cout << "\n\nThe level order traversal is:\n";
   intTree.levelOrderTraversal();

   cout << endl;

   return 0;
}
```

```
The values being placed in the tree are:
13 21 85 48 83 42 93 38 29 11 21 4 6 69 81

The level order traversal is:
13 11 21 4 21 85 6 48 93 42 83 38 69 29 81
```

15.25 (*Printing trees*) Write a recursive member function **`outputTree`** to display a binary tree object on the screen. The function should output the tree row by row, with the top of the tree at the left of the screen and the bottom of the tree toward theright of the screen. Each row is output vertically. For example, the binary tree illustrated in Fig. 15.19 is output as follows:

```
                  99
             97
                  92
        83
                  72
             71
                  69
49
                  44
             40
                  32
        28
                  19
             18
                  11
```

Note that the rightmost leaf node appears at the top of the output in the rightmost column and the root node appears at the left of the output. Each column of output starts five spaces to the right of the previous column. Function **`outputTree`** should receive an argument **`totalSpaces`** representing the number of spaces preceding the value to be output (this variable should start at zero so the root node is output at the left of the screen). The function uses a modified inorder traversal to output the tree—it starts at t he right-most node in the tree and works back to the left. The algorithm is as follows:

```
While the pointer to the current node is not null
    Recursively call outputTree with the right subtree of the current node and
        totalSpaces + 5
    Use a for structure to count from 1 to totalSpaces and output spaces
    Output the value in the current node
    Set the pointer to the current node to point to the left subtree of the current node
    Increment totalSpaces by 5.
```

```cpp
// TREE
// Definition of template class Tree
#ifndef TREE_H
#define TREE_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"

template< class NODETYPE >
class Tree {
public:
   Tree();
   void insertNode( const NODETYPE & );
   void preOrderTraversal() const;
   void inOrderTraversal() const;
   void postOrderTraversal() const;
protected:
   TreeNode<NODETYPE> *rootPtr;

   // utility functions
   void insertNodeHelper( TreeNode< NODETYPE > **, const NODETYPE & );
   void preOrderHelper( TreeNode< NODETYPE > * ) const;
   void inOrderHelper( TreeNode< NODETYPE > * ) const;
   void postOrderHelper( TreeNode< NODETYPE > * ) const;
};

template< class NODETYPE >
Tree< NODETYPE >::Tree() { rootPtr = 0; }

template< class NODETYPE >
void Tree< NODETYPE >::insertNode( const NODETYPE &value )
   { insertNodeHelper( &rootPtr, value ); }

// This function receives a pointer to a pointer so the
// pointer can be modified.
// NOTE: THIS FUNCTION WAS MODIFIED TO ALLOW DUPLICATES.
template< class NODETYPE >
void Tree< NODETYPE >::insertNodeHelper( TreeNode< NODETYPE > **ptr,
                                         const NODETYPE &value )
{
   if ( *ptr == 0 ) {                    // tree is empty
      *ptr = new TreeNode< NODETYPE >( value );
      assert( *ptr != 0 );
   }
   else                               // tree is not empty
      if ( value <= ( *ptr ) -> data )
         insertNodeHelper( &( ( *ptr ) -> leftPtr ), value );
      else
         insertNodeHelper( &( ( *ptr ) -> rightPtr ), value );
}

template< class NODETYPE >
void Tree< NODETYPE >::preOrderTraversal() const { preOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::preOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      cout << ptr -> data << ' ';
      preOrderHelper( ptr -> leftPtr );
      preOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::inOrderTraversal() const { inOrderHelper( rootPtr ); }

template< class NODETYPE >
void Tree< NODETYPE >::inOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      inOrderHelper( ptr -> leftPtr );
      cout << ptr -> data << ' ';
      inOrderHelper( ptr -> rightPtr );
   }
}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderTraversal() const { postOrderHelper( rootPtr );}

template< class NODETYPE >
void Tree< NODETYPE >::postOrderHelper( TreeNode< NODETYPE > *ptr ) const
{
   if ( ptr != 0 ) {
      postOrderHelper( ptr -> leftPtr );
      postOrderHelper( ptr -> rightPtr );
      cout << ptr -> data << ' ';
   }
}

#endif
```

```
// TREENODE.H
// Definition of class TreeNode
#ifndef TREENODE_H
#define TREENODE_H

template< class T > class Tree;    // forward declaration

template< class NODETYPE >
class TreeNode {
   friend class Tree< NODETYPE >;
public:
   TreeNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;      // return data
   TreeNode *getLeftPtr() const { return leftPtr; }
   TreeNode *getRightPtr() const { return rightPtr; }
   void setLeftPtr( TreeNode *ptr ) { leftPtr = ptr; }
   void setRightPtr( TreeNode *ptr ) { rightPtr = ptr; }
private:
   TreeNode *leftPtr;   // pointer to left subtree
   NODETYPE data;
   TreeNode *rightPtr;  // pointer to right subtree
};

// Constructor
template< class NODETYPE >
TreeNode< NODETYPE >::TreeNode( const NODETYPE &d )
{
   data = d;
   leftPtr = rightPtr = 0;
}

//Return a copy of the data value
template< class NODETYPE >
NODETYPE TreeNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// TREE2
// Definition of template class Tree
#ifndef TREE2_H
#define TREE2_H

#include <iostream>

using std::cout;

#include <cassert>
#include "treenode.h"
#include "tree.h"

template< class NODETYPE >
class Tree2 : public Tree< NODETYPE > {
public:
   void outputTree() const;
private:
   void outputTreeHelper( TreeNode< NODETYPE > *, int ) const;
};

template< class NODETYPE >
void Tree2< NODETYPE >::outputTree() const { outputTreeHelper( rootPtr, 0 ); }

template< class NODETYPE >
void Tree2< NODETYPE >::outputTreeHelper( TreeNode< NODETYPE > *ptr,
                                          int totalSpaces ) const
{
   if ( ptr != 0 ) {
      outputTreeHelper( ptr -> getRightPtr(), totalSpaces + 5 );

      for ( int i = 1; i <= totalSpaces; ++i )
         cout << ' ';

      cout << ptr -> getData() << '\n';
      outputTreeHelper( ptr -> getLeftPtr(), totalSpaces + 5 );
   }
}

#endif
```

```
// Exercise 15.25 solution
#include <iostream>

using std::cout;

#include <cstdlib>
#include <ctime>
#include "tree2.h"

int main()
{
   srand( time( 0 ) );  // randomize the random number generator

   Tree2< int > intTree;
   int intVal;

   cout << "The values being placed in the tree are:\n";

   for ( int i = 1; i <= 15; ++i ) {
      intVal = rand() % 100;
      cout << intVal << ' ';
      intTree.insertNode( intVal );
   }

   cout << "\n\nThe tree is:\n";
   intTree.outputTree();

   return 0;
}
```

```
The values being placed in the tree are:
12 14 56 18 12 97 12 82 96 27 99 90 22 87 15

The tree is:
                    99
               97
                         96
                              90
                                   87
                    82
          56
                    27
                         22
               18
                    15
     14
12
     12
          12
```

SPECIAL SECTION—BUILDING YOUR OWN COMPILER

In Exercises 5.18 and 5.19, we introduced Simpletron Machine Language (SML) and you implemented a Simpletron computer simulator to execute programs written in SML. In this section, we build a compiler that converts programs written in a high-leve l programming language to SML. This section “ties” together the entire programming process. You will write programs in this new high-level language, compile these programs on the compiler you build and run the programs on the simulator you built in Exercis 7.19. You should make every effort to implement your compiler in an object-oriented manner.

15.26 (*The Simple Language*) Before we begin building the compiler, we discuss a simple, yet powerful, high-level language similar to early versions of the popular language BASIC. We call the language *Simple*. Every Simple *statement* consists of a *line number* and a Simple *instruction*. Line numbers must appear in ascending order. Each instruction begins with one of the following Simple *commands*: **rem**, **input**, **let**, **print**, **goto**, **if**/**goto** and **end** (see Fig. 15.20). All commands except **end** can be used repeatedly. Simple evaluates only integer expressions using the **+**, **-**, ***** and **/** operators. These operators have the same precedence as in C. Parentheses can be used to change the order of evaluation of an expression.

Our Simple compiler recognizes only lowercase letters. All characters in a Simple file should be lowercase (uppercase letters result in a syntax error unless they appear in a **rem** statement, in which case they are ignored). A *variable name* is a single letter. Simple does not allow descriptive variable names, so variables should be explained in remarks to indicate their use in a program . Simple uses only integer variables. Simple does not have variable declarations—merely mentioning a variable name in a program causes the variable to be declared and initialized to zero automatically. The syntax of Simple does not allow string manipulatio n (reading a string, writing a string, comparing strings, etc.). If a string is encountered in a Simple program (after a command o ther than **rem**), the compiler generates a syntax error. The first version of our compiler will assume that Simple programs are entered correctly. Exercise 15.29 asks the student to modify the compiler to perform syntax error checking.

Command	Example statement	Description
`rem`	`50 rem this is a remark`	Text following **rem** is for documentation purposes and is ignored by the compiler.
`input`	`30 input x`	Display a question mark to prompt the user to enter an integer. Read that integer from the keyboard and store the integer in **x**.
`let`	`80 let u = 4 * (j - 56)`	Assign **u** the value of **4 * (j - 56)**. Note that an arbitrarily complex expression can appear to the right of the equals sign.
`print`	`10 print w`	Display the value of **w**.
`goto`	`70 goto 45`	Transfer program control to line **45**.
`if/goto`	`35 if i == z goto 80`	Compare **i** and **z** for equality and transfer control to line **80** if the condition is true; otherwise, continue execution with the next statement.
`end`	`99 end`	Terminate program execution.

Fig. 15.20 Simple commands.

Simple uses the conditional **if**/**goto** statement and the unconditional **goto** statement to alter the flow of control during program execution. If the condition in the **if**/**goto** statement is true, control is transferred to a specific line of the program. The following relational and equality operators are valid in an **if**/**goto** statement: **<**, **>**, **<=**, **>=**, **==** and **!=**. The precedence of these operators is the same as in C++.

Let us now consider several programs that demonstrate Simple's features. The first program (Fig. 15.21) reads two integers from the keyboard, stores the values in variables **a** and **b** and computes and prints their sum (stored in variable **c**).

The program of Fig. 15.22 determines and prints the larger of two integers. The integers are input from the keyboard and stored in **s** and **t**. The **if/goto** statement tests the condition **s >= t**. If the condition is true, control is transferred to line **90** and **s** is output; otherwise, **t** is output and control is transferred to the **end** statement in line **99** where the program terminates.

Simple does not provide a repetition structure (such as C++'s **for**, **while** or **do**/**while**). However, Simple can simulate each of C++'s repetition structures using the **if**/**goto** and **goto** statements. Figure 15.23 uses a sentinel-controlled loop to calculate the squares of several integers. Each integer is input from the keyboard and stored in variable **j**. If the value entered is the sentinel
-9999, control is transferred to line **99**, where the program terminates. Otherwise, **k** is assigned the square of **j**, **k** is output to the screen and control is passed to line **20**, where the next integer is input.

```
10 rem   determine and print the sum of two integers
15 rem
```

Fig. 15.21 Simple program that determines the sum of two integers (part 1 of 2).

```
20 rem   input the two integers
30 input a
40 input b
45 rem
50 rem   add integers and store result in c
60 let c = a + b
65 rem
70 rem   print the result
80 print c
90 rem   terminate program execution
99 end
```

Fig. 15.21 Simple program that determines the sum of two integers (part 2 of 2).

```
10 rem   determine the larger of two integers
20 input s
30 input t
32 rem
35 rem   test if s >= t
40 if s >= t goto 90
45 rem
50 rem   t is greater than s, so print t
60 print t
70 goto 99
75 rem
80 rem   s is greater than or equal to t, so print s
90 print s
99 end
```

Fig. 15.22 Simple program that finds the larger of two integers.

```
10 rem   calculate the squares of several integers
20 input j
23 rem
25 rem   test for sentinel value
30 if j == -9999 goto 99
33 rem
35 rem   calculate square of j and assign result to k
40 let k = j * j
50 print k
53 rem
55 rem   loop to get next j
60 goto 20
99 end
```

Fig. 15.23 Calculate the squares of several integers.

Using the sample programs of Fig. 15.21, Fig. 15.22 and Fig. 15.23 as your guide, write a Simple program to accomplish each of the following:

a) Input three integers, determine their average and print the result.
b) Use a sentinel-controlled loop to input 10 integers and compute and print their sum.
c) Use a counter-controlled loop to input 7 integers, some positive and some negative, and compute and print their average.
d) Input a series of integers and determine and print the largest. The first integer input indicates how many numbers should be processed.
e) Input 10 integers and print the smallest.
f) Calculate and print the sum of the even integers from 2 to 30.
g) Calculate and print the product of the odd integers from 1 to 9.

15.27 (*Building A Compiler; Prerequisite: Complete Exercises 5.18, 5.19, 15.12, 15.13 and 15.26*) Now that the Simple language has been presented (Exercise 15.26), we discuss how to build a Simple compiler. First, we consider the process by which a Simple program is converted to SML and executed by the Simpletron simulator (see Fig. 15.24). A file containing a Simple program is read by the compiler and converted to SML code. The SML code is output to a file on disk, in which SML instructions appear one per line. The SML file is then loaded into the Simpletron simulator, and the results are sent to a file on disk and to the screen. Note that the Simpletron program developed in Exercise 5.19 took its input from the keyboard. It must be modified to read from a file so it can run the programs produced by our compiler.

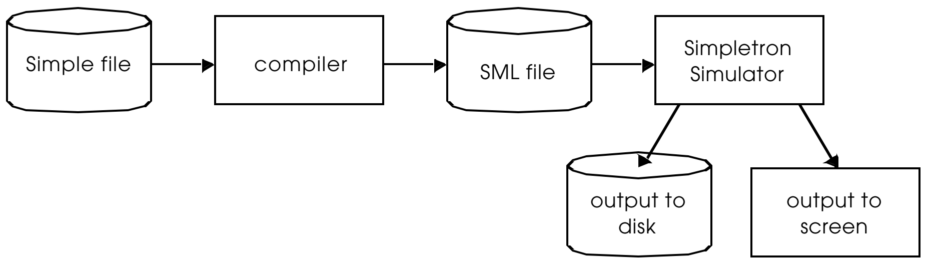

Fig. 15.24 Writing, compiling and executing a Simple language program.

The Simple compiler performs two *passes* of the Simple program to convert it to SML. The first pass constructs a *symbol table* (object) in which every *line number* (object), *variable name* (object) and *constant* (object) of the Simple program is stored with its type and corresponding location in the final SML code (the symbol table is discussed in detail below). The first pass also produces the corresponding SML instruction object(s) for each of the Simple statements (object, etc.). As we will see, if the Simple program contains statements that transfer control to a line later in the program, the first pass results in an SML program containing some "unfinished" instructions. The second pass of the compiler locates and completes the unfinished instructions, and outputs the SML program to a file.

First Pass

The compiler begins by reading one statement of the Simple program into memory. The line must be separated into its individual *tokens* (i.e., "pieces" of a statement) for processing and compilation (standard library function **`strtok`** can be used to facilitate this task). Recall that every statement begins with a line number followed by a command. As the compiler breaks a statement into tokens, if the token is a line number, a variable or a constant, it is placed in the symbol table. A line number is placed in the symbol table only if it is the first token in a statement. The **`symbolTable`** object is an array of **`tableEntry`** objects representing each symbol in the program. There is no restriction on the number of symbols that can appear in the program. Therefore, the **`symbolTable`** for a particular program could be large. Make the **`symbolTable`** a 100-element array for now. You can increase or decrease its size once the program is working.

Each **`tableEntry`** object contains three members. Member **`symbol`** is an integer containing the ASCII representation of a variable (remember that variable names are single characters), a line number or a constant. Member **`type`** is one of the following characters indicating the symbol's type: **`'C'`** for constant, **`'L'`** for line number and **`'V'`** for variable. Member **`location`** contains the Simpletron memory location (**`00`** to **`99`**) to which the symbol refers. Simpletron memory is an array of 100 integers in which SML instructions and data are stored. For a line number, the location is the element in the Simpletron memory array at which the SML instructions for the Simple statement begin. For a variable or constant, the location is the element in the Simpletron memory array in which the variable or constant is stored. Variables and constants are allocated from the end of Simpletron's memory backwards. The first variable or constant is stored in location at **`99`**, the next in location at **`98`**, etc.

The symbol table plays an integral part in converting Simple programs to SML. We learned in Chapter 5 that an SML instruction is a four-digit integer composed of two parts—the *operation code* and the *operand*. The operation code is determined by commands in Simple. For example, the simple command **`input`** corresponds to SML operation code **`10`** (read), and the Simple command **`print`** corresponds to SML operation code **`11`** (write). The operand is a memory location containing the data on which the operation code performs its task (e.g., operation code **`10`** reads a value from the keyboard and stores it in the memory location specified by the operand). The compiler searches **`symbolTable`** to determine the Simpletron memory location for each symbol so the corresponding location can be used to complete the SML instructions.

The compilation of each Simple statement is based on its command. For example, after the line number in a **`rem`** statement is inserted in the symbol table, the remainder of the statement is ignored by the compiler because a remark is for documentation purposes only. The **`input`**, **`print`**, **`goto`** and **`end`** statements correspond to the SML *read*, *write*, *branch* (to a specific location) and *halt* instructions. Statements containing these Simple commands are converted directly to SML (note that a **`goto`** statement may contain an unresolved reference if the specified line number refers to a statement further into the Simple program file; this is sometimes called a forward reference).

When a **goto** statement is compiled with an unresolved reference, the SML instruction must be *flagged* to indicate that the second pass of the compiler must complete the instruction. The flags are stored in 100-element array **flags** of type **int** in which each element is initialized to **-1**. If the memory location to which a line number in the Simple program refers is not yet known (i.e., it is not in the symbol table), the line number is stored in array **flags** in the element with the same subscript as the incomplete instruction. The operand of the incomplete instruction is set to **00** temporarily. For example, an unconditional branch instruction (making a forward reference) is left as **+4000** until the second pass of the compiler. The second pass of the compiler is described shortly.

Compilation of **if/goto** and **let** statements is more complicated than other statements—they are the only statements that produce more than one SML instruction. For an **if/goto**, the compiler produces code to test the condition and to branch to another line if necessary. The result of the branch could be an unresolved reference. Each of the relational and equality operators can be simulated using SML's *branch zero* and *branch negative* instructions (or a combination of both).

For a **let** statement, the compiler produces code to evaluate an arbitrarily complex arithmetic expression consisting of integer variables and/or constants. Expressions should separate each operand and operator with spaces. Exercises 15.12 and 15.13 presented the infix-to-postfix conversion algorithm and the postfix evaluation algorithm used by compilers to evaluate expressions. Before proceeding with your compiler, you should complete each of these exercises. When a compiler encounters an expression, it converts the expression from infix notation to postfix notation and then evaluates the postfix expression.

How is it that the compiler produces the machine language to evaluate an expression containing variables? The postfix evaluation algorithm contains a "hook" where the compiler can generate SML instructions rather than actually evaluating the expression. To enable this "hook" in the compiler, the postfix evaluation algorithm must be modified to search the symbol table for each symbol it encounters (and possibly insert it), determine the symbol's corresponding memory location and *push the memory location onto the stack (instead of the symbol)*. When an operator is encountered in the postfix expression, the two memory locations at the top of the stack are popped and machine language for effecting the operation is produced using the memory locations as operands. The result of each subexpression is stored in a temporary location in memory and pushed back onto the stack so the evaluation of the postfix expression can continue. When postfix evaluation is complete, the memory location containing the result is the only location left on the stack. This is popped and SML instructions are generated to assign the result to the variable at the left of the **let** statement.

Second Pass

The second pass of the compiler performs two tasks: resolve any unresolved references and output the SML code to a file. Resolution of references occurs as follows:

a) Search the **flags** array for an unresolved reference (i.e., an element with a value other than **-1**).
b) Locate the object in array **symbolTable**, containing the symbol stored in the **flags** array (be sure that the type of the symbol is **'L'** for line number).
c) Insert the memory location from member **location** into the instruction with the unresolved reference (remember that an instruction containing an unresolved reference has operand **00**).
d) Repeat steps 1, 2 and 3 until the end of the **flags** array is reached.

After the resolution process is complete, the entire array containing the SML code is output to a disk file with one SML instruction per line. This file can be read by the Simpletron for execution (after the simulator is modified to read its input from a file). Compiling your first Simple program into an SML file and then executing that file should give you a real sense of personal accomplishment.

A Complete Example

The following example illustrates a complete conversion of a Simple program to SML as it will be performed by the Simple compiler. Consider a Simple program that inputs an integer and sums the values from 1 to that integer. The program and the SML instructions produced by the first pass of the Simple compiler are illustrated in Fig. 15.25. The symbol table constructed by the first pass is shown in Fig. 15.26.

Simple program	SML location and instruction	Description
`5 rem sum 1 to x`	*none*	**rem** ignored

Fig. 15.25 SML instructions produced after the compiler's first pass.

Simple program	SML location and instruction	Description
`10 input x`	`00 +1099`	read `x` into location `99`
`15 rem check y == x`	*none*	`rem` ignored
`20 if y == x goto 60`	`01 +2098`	load `y` (`98`) into accumulator
	`02 +3199`	sub `x` (`99`) from accumulator
	`03 +4200`	branch zero to unresolved location
`25 rem   increment y`	*none*	`rem` ignored
`30 let y = y + 1`	`04 +2098`	load `y` into accumulator
	`05 +3097`	add `1` (`97`) to accumulator
	`06 +2196`	store in temporary location `96`
	`07 +2096`	load from temporary location `96`
	`08 +2198`	store accumulator in `y`
`35 rem   add y to total`	*none*	`rem` ignored
`40 let t = t + y`	`09 +2095`	load `t` (`95`) into accumulator
	`10 +3098`	add `y` to accumulator
	`11 +2194`	store in temporary location `94`
	`12 +2094`	load from temporary location `94`
	`13 +2195`	store accumulator in `t`
`45 rem   loop y`	*none*	`rem` ignored
`50 goto 20`	`14 +4001`	branch to location `01`
`55 rem   output result`	*none*	`rem` ignored
`60 print t`	`15 +1195`	output `t` to screen
`99 end`	`16 +4300`	terminate execution

Fig. 15.25 SML instructions produced after the compiler's first pass.

Symbol	Type	Location
`5`	`L`	`00`
`10`	`L`	`00`
`'x'`	`V`	`99`
`15`	`L`	`01`
`20`	`L`	`01`
`'y'`	`V`	`98`
`25`	`L`	`04`
`30`	`L`	`04`
`1`	`C`	`97`
`35`	`L`	`09`
`40`	`L`	`09`
`'t'`	`V`	`95`

Fig. 15.26 Symbol table for program of Fig. 15.25 (part 1 of 2).

Symbol	Type	Location
`45`	`L`	`14`
`50`	`L`	`14`
`55`	`L`	`15`
`60`	`L`	`15`
`99`	`L`	`16`

Fig. 15.26 Symbol table for program of Fig. 15.25 (part 2 of 2).

Most Simple statements convert directly to single SML instructions. The exceptions in this program are remarks, the **`if`**/**`goto`** statement in line **`20`** and the **`let`** statements. Remarks do not translate into machine language. However, the line number for a remark is placed in the symbol table in case the line number is referenced in a **`goto`** statement or an **`if/goto`** statement. Line **`20`** of the program specifies that if the condition **`y == x`** is true, program control is transferred to line **`60`**. Because line **`60`** appears later in the program, the first pass of the compiler has not as yet placed **`60`** in the symbol table (statement line numbers are placed in the symbol table only when they appear as the first token in a statement). Therefore, it is not possible at this time to determine the operand of the SML *branch zero* instruction at location **`03`** in the array of SML instructions. The compiler places **`60`** in location **`03`** of the **`flags`** array to indicate that the second pass completes this instruction.

We must keep track of the next instruction location in the SML array because there is not a one-to-one correspondence between Simple statements and SML instructions. For example, the **`if`**/**`goto`** statement of line **`20`** compiles into three SML instructions. Each time an instruction is produced, we must increment the *instruction counter* to the next location in the SML array. Note that the size of Simpletron's memory could present a problem for Simple programs with many statements, variables and constants. It is conceivable that the compiler will run out of memory. To test for this case, your program should contain a *data counter* to keep track of the location at which the next variable or constant will be stored in the SML array. If the value of the instruction counter is larger than the value of the data counter, the SML array is full. In this case, the compilation process should terminate and the compiler should print an error message indicating that it ran out of memory during compilation. This serves to emphasize that although the programmer is freed from the burdens of managing memory by the compiler, the compiler itself must carefully determine the placement of instructions and data in memory, and must check for such errors as memory being exhausted during the compilation process.

A Step-by-Step View of the Compilation Process

Let us now walk through the compilation process for the Simple program in Fig. 15.25. The compiler reads the first line of the program

```
5 rem sum 1 to x
```

into memory. The first token in the statement (the line number) is determined using **`strtok`** (see Chapters 5 and 16 for a discussion of C++'s string manipulation functions). The token returned by **`strtok`** is converted to an integer using **`atoi`** so the symbol **`5`** can be located in the symbol table. If the symbol is not found, it is inserted in the symbol table. Since we are at the beginning of the program and this is the first line, no symbols are in the table yet. So, **`5`** is inserted into the symbol table as type **`L`** (line number) and assigned the first location in SML array (**`00`**). Although this line is a remark, a space in the symbol table is still allocated for the line number (in case it is referenced by a **`goto`** or an **`if`**/**`goto`**). No SML instruction is generated for a **`rem`** statement, so the instruction counter is not incremented.

The statement

```
10 input x
```

is tokenized next. The line number **`10`** is placed in the symbol table as type **`L`** and assigned the first location in the SML array (**`00`** because a remark began the program so the instruction counter is currently **`00`**). The command **`input`** indicates that the next token is a variable (only a variable can appear in an **`input`** statement). Because **`input`** corresponds directly to an SML operation code, the compiler has to determine the location of **`x`** in the SML array. Symbol **`x`** is not found in the symbol table. So, it is inserted into the symbol table as the ASCII representation of **`x`**, given type **`V`**, and assigned location **`99`** in the SML array (data storage begins at **`99`** and is allocated backwards). SML code can now be generated for this statement. Operation code **`10`** (the SML read operation code) is multiplied by 100, and the location of **`x`** (as determined in the symbol table) is added to complete the instruction. The instruction is then stored in the SML array at location **`00`**. The instruction counter is incremented by 1 because a single SML instruction was produced.

The statement

```
15 rem   check y == x
```

is tokenized next. The symbol table is searched for line number **15** (which is not found). The line number is inserted as type **L** and assigned the next location in the array, **01** (remember that **rem** statements do not produce code, so the instruction counter is not incremented).

The statement

```
20 if y == x goto 60
```

is tokenized next. Line number **20** is inserted in the symbol table and given type **L** with the next location in the SML array **01**. The command **if** indicates that a condition is to be evaluated. The variable **y** is not found in the symbol table, so it is inserted and given the type **V** and the SML location **98**. Next, SML instructions are generated to evaluate the condition. Since there is no direct equivalent in SML for the **if/goto**, it must be simulated by performing a calculation using **x** and **y** and branching based on the result. If **y** is equal to **x**, the result of subtracting **x** from **y** is zero, so the *branch zero* instruction can be used with the result of the calculation to simulate the **if/goto** statement. The first step requires that **y** be loaded (from SML location **98**) into the accumulator. This produces the instruction **01 +2098**. Next, **x** is subtracted from the accumulator. This produces the instruction **02 +3199**. The value in the accumulator may be zero, positive or negative. Since the operator is **==**, we want to *branch zero.* First, the symbol table is searched for the branch location (**60** in this case), which is not found. So, **60** is placed in the **flags** array at location **03**, and the instruction **03 +4200** is generated (we cannot add the branch location, because we have not assigned a location to line **60** in the SML array yet). The instruction counter is incremented to **04**.

The compiler proceeds to the statement

```
25 rem   increment y
```

The line number **25** is inserted in the symbol table as type **L** and assigned SML location **04**. The instruction counter is not incremented.

When the statement

```
30 let y = y + 1
```

is tokenized, the line number **30** is inserted in the symbol table as type **L** and assigned SML location **04**. Command **let** indicates that the line is an assignment statement. First, all the symbols on the line are inserted in the symbol table (if they are not a lready there). The integer **1** is added to the symbol table as type **C** and assigned SML location **97**. Next, the right side of the assignment is converted from infix to postfix notation. Then the postfix expression (**y 1 +**) is evaluated. Symbol **y** is located in the symbol table and its corresponding memory location is pushed onto the stack. Symbol **1** is also located in the symbol table and its corresponding memory location is pushed onto the stack. When the operator **+** is encountered, the postfix evaluator pops the stack into the right operand of the operator, pops the stack again into the left operand of the operator and then produces the SML instructions

04 +2098 *(load* **y***)*
05 +3097 *(add* **1***)*

The result of the expression is stored in a temporary location in memory (**96**) with instruction

06 +2196 *(store temporary)*

and the temporary location is pushed on the stack. Now that the expression has been evaluated, the result must be stored in **y** (i.e., the variable on the left side of **=**). So, the temporary location is loaded into the accumulator and the accumulator is stored in **y** with the instructions

07 +2096 *(load temporary)*
08 +2198 *(store* **y***)*

The reader will immediately notice that SML instructions appear to be redundant. We will discuss this issue shortly.

When the statement

```
35 rem   add y to total
```

is tokenized, line number **35** is inserted in the symbol table as type **L** and assigned location **09**.

The statement

```
40 let t = t + y
```

is similar to line **30**. The variable **t** is inserted in the symbol table as type **V** and assigned SML location **95**. The instructions follow the same logic and format as line **30**, and the instructions **09 +2095**, **10 +3098**, **11 +2194**, **12 +2094** and **13 +2195** are generated. Note that the result of **t + y** is assigned to temporary location **94** before being assigned to **t** (**95**). Once again, the reader will note that the instructions in memory locations **11** and **12** appear to be redundant. Again, we will discuss this shortly.

The statement

```
45 rem   loop y
```

is a remark, so line **45** is added to the symbol table as type **L** and assigned SML location **14**.

The statement

```
50 goto 20
```

transfers control to line **20**. Line number **50** is inserted in the symbol table as type **L** and assigned SML location **14**. The equivalent of **goto** in SML is the *unconditional branch* (**40**) instruction that transfers control to a specific SML location. The compiler searches the symbol table for line **20** and finds that it corresponds to SML location **01**. The operation code (**40**) is multiplied by 100, and location **01** is added to it to produce the instruction **14 +4001**.

The statement

```
55 rem   output result
```

is a remark, so line **55** is inserted in the symbol table as type **L** and assigned SML location **15**.

The statement

```
60 print t
```

is an output statement. Line number **60** is inserted in the symbol table as type **L** and assigned SML location **15**. The equivalent of **print** in SML is operation code **11** (*write*). The location of **t** is determined from the symbol table and added to the result of the operation code multiplied by 100.

The statement

```
99 end
```

is the final line of the program. Line number **99** is stored in the symbol table as type **L** and assigned SML location **16**. The **end** command produces the SML instruction **+4300** (**43** is *halt* in SML), which is written as the final instruction in the SML memory array.

This completes the first pass of the compiler. We now consider the second pass. The **flags** array is searched for values other than **-1**. Location **03** contains **60**, so the compiler knows that instruction **03** is incomplete. The compiler completes the instruction by searching the symbol table for **60**, determining its location and adding the location to the incomplete instruction. In this case, the search determines that line **60** corresponds to SML location **15**, so the completed instruction **03 +4215** is produced, replacing **03 +4200**. The Simple program has now been compiled successfully.

To build the compiler, you will have to perform each of the following tasks:

a) Modify the Simpletron simulator program you wrote in Exercise 5.19 to take its input from a file specified by the user (see Chapter 14). The simulator should output its results to a disk file in the same format as the screen output. Convert the simulator to be an object-oriented program. In particular, make each part of the hardware an object. Arrange the instruction types into a class hierarchy using inheritance. Then execute the program polymorphically by telling each instruction to execute itself with an **executeInstruction** message.

b) Modify the infix-to-postfix conversion algorithm of Exercise 15.12 to process multi-digit integer operands and single-letter variable name operands. (Hint: Standard library function **strtok** can be used to locate each constant and variable in an expression, and constants can be converted from strings to integers using standard library function **atoi**.) (Note: The data representation of the postfix expression must be altered to support variable names and integer constants.)

c) Modify the postfix evaluation algorithm to process multidigit integer operands and variable name operands. Also, the algorithm should now implement the "hook" discussed above so that SML instructions are produced rather than directly evaluating the expression. (Hint: Standard library function **`strtok`** can be used to locate each constant and variable in an expression, and constants can be converted from strings to integers using standard library function **`atoi`**.) (*Note:* The data representation of the postfix expression must be altered to support variable names and integer constants.)

d) Build the compiler. Incorporate parts (b) and (c) for evaluating expressions in **`let`** statements. Your program should contain a function that performs the first pass of the compiler and a function that performs the second pass of the compiler. Both functions can call other functions to accomplish their tasks. Make your compiler as object oriented as possible.

```
// STACKND.H
// Definition of template class StackNode
#ifndef STACKND_H
#define STACKND_H

template < class T > class Stack;

template < class T >
class StackNode {
   friend class Stack< T >;
public:
   StackNode( const T & = 0, StackNode * = 0 );
   T getData() const;
private:
   T data;
   StackNode *nextPtr;
};

// Member function definitions for class StackNode
template < class T >
StackNode< T >::StackNode( const T &d, StackNode< T > *ptr )
{
   data = d;
   nextPtr = ptr;
}

template < class T >
T StackNode< T >::getData() const { return data; }

#endif
```

```
// STACK.H
// Definition of class Stack
#ifndef STACK_H
#define STACK_H

#include <iostream>
using std::cout;

#include <cassert>
#include "stacknd.h"

template < class T >
class Stack {

public:
   Stack();              // default constructor
   ~Stack();             // destructor
   void push( T & );     // insert item in stack
   T pop();              // remove item from stack
   bool isEmpty() const;  // is the stack empty?
   T stackTop() const;   // return the top element of stack
   void print() const;   // output the stack
private:
   StackNode< T > *topPtr;    // pointer to fist StackNode
};

// Member function definitions for class Stack
template < class T >
Stack< T >::Stack() { topPtr = 0; }

template < class T >
Stack< T >::~Stack()
{
   StackNode< T > *tempPtr, *currentPtr = topPtr;

   while ( currentPtr != 0 ) {
      tempPtr = currentPtr;
      currentPtr = currentPtr -> nextPtr;
      delete tempPtr;
   }
}

template < class T >
void Stack< T >::push( T &d )
{
   StackNode< T > *newPtr = new StackNode< T >( d, topPtr );

   assert( newPtr != 0 );  // was memory allocated?
   topPtr = newPtr;
}

template < class T >
T Stack< T >::pop()
{
   assert( !isEmpty() );

   StackNode< T > *tempPtr = topPtr;

   topPtr = topPtr -> nextPtr;
   T poppedValue = tempPtr -> data;
   delete tempPtr;
   return poppedValue;
}

template < class T >
bool Stack< T >::isEmpty() const { return topPtr == 0; }

template < class T >
T Stack< T >::stackTop() const
{ return !isEmpty() ? topPtr -> data : 0; }

template < class T >
void Stack< T >::print() const
{
   StackNode< T > *currentPtr = topPtr;


   if ( isEmpty() )          // Stack is empty
      cout << "Stack is empty\n";
   else {                   // Stack is not empty
      cout << "The stack is:\n";

      while ( currentPtr != 0 ) {
         cout << currentPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
      }

      cout << '\n';
   }
}

#endif
```

```
// compiler.h

const int MAXIMUM = 81;            // maximum length for lines
const int SYMBOLTABLESIZE = 100;   // maximum size of symbol table
const int MEMORYSIZE = 100;        // maximum Simpletron memory

// Definition of structure for symbol table entries
struct TableEntry {
   int location;               // SML memory location 00 to 99
   char type;                  // 'C' = constant, 'V' = variable,
   int symbol;                 // or 'L' = line number
};

// Function prototypes for compiler functions
int checkSymbolTable( TableEntry *, int, char );
int checkOperand( TableEntry *, char *, int *, int *, int * );
void addToSymbolTable( char, int, int, TableEntry *, int );
void addLineToFlags( int, int, int *, int *, const int * );
void compile( ifstream &, char * );
void printOutput( const int [], char * );
void lineNumber( char *, TableEntry *, int, int );
void initArrays( int *, int *, TableEntry * );
void firstPass( int *, int *, TableEntry *, ifstream & );
void secondPass( int *, int *, TableEntry * );
void separateToken( char *, int *, int *, TableEntry *, int *, int * );
void keyWord( char *, int *, int *, TableEntry *, int *, int * );
void keyLet( char *, int *, TableEntry *, int *, int * );
void keyInput( char *, int *, TableEntry *, int *, int * );
void keyPrint( char *, int *, TableEntry *, int *, int * );
void keyGoto( char *, int *, int *, TableEntry *, int * );
void keyIfGoto( char *, int *, int *, TableEntry *, int *, int * );
void evaluateExpression( int, int, char *, int *, TableEntry *,
                         int *, int *, char * );
int createLetSML( int, int, int *, int *, int *, char );
void infixToPostfix( char *, char *, int, TableEntry *, int *, int *, int * );
void evaluatePostfix( char *, int *, int *, int *, int, TableEntry * );
bool isOperator( char );
bool precedence( char, char );

// Simpletron Machine Language (SML) Operation Codes
enum SMLOperationCodes { READ = 10, WRITE = 11, LOAD = 20, STORE = 21, ADD = 30,
                         SUBTRACT = 31, DIVIDE = 32, MULTIPLY = 33, BRANCH = 40,
                         BRANCHNEG = 41, BRANCHZERO = 42, HALT = 43 };
```

```
// Exercise 15.27 Solution
// Non-optimized version.
#include <iostream>

```

```
using std::cout;
using std::cin;
using std::ios;
using std::cerr;

#include <iomanip>

using std::setw;

#include <fstream>

using std::ifstream;
using std::ofstream;

#include <cstring>
#include <cctype>
#include <cstdlib>
#include "stack.h"

#include "compiler.h"

int main()
{
   char inFileName[ 15 ] = "", outFileName[ 15 ] = "";
   int last = 0;

   cout << "Enter Simple file to be compiled: ";
   cin >> setw( 15 ) >> inFileName;

   while ( isalnum( inFileName[ last ] ) != 0 ) {
      outFileName[ last ] = inFileName[ last ];
      last++;            // note the last occurance
   }

   outFileName[ last ] = '\0';     // append a NULL character
   strcat( outFileName, ".sml" );  // add .sml to name

   ifstream inFile( inFileName, ios::in );

   if ( inFile )
      compile( inFile, outFileName );
   else
      cerr << "File not opened. Program execution terminating.\n";

   return 0;
}

// compile function calls the first pass and the second pass
void compile( ifstream &input, char *outFileName )
{
   TableEntry symbolTable[ SYMBOLTABLESIZE ]; // symbol table
   int flags[ MEMORYSIZE ];           // array for forward references
   int machineArray[ MEMORYSIZE ];    // array for SML instructions

   initArrays( flags, machineArray, symbolTable );

   firstPass( flags, machineArray, symbolTable, input );
   secondPass( flags, machineArray, symbolTable );


   printOutput( machineArray, outFileName );
}

```

```
// firstPass constructs the symbol table, creates SML, and flags unresolved
// references for goto and if/goto statements.
void firstPass( int flags[], int machineArray[], TableEntry symbolTable[],
                ifstream &input )
{
   char array[ MAXIMUM ];              // array to copy a Simple line
   int n = MAXIMUM;                   // required for fgets()
   int dataCounter = MEMORYSIZE - 1; // 1st data location in machineArray
   int instCounter = 0;              // 1st instruction location in machineArray

   input.getline( array, n );

   while ( !input.eof() ) {
      separateToken( array, flags, machineArray, symbolTable, &dataCounter,
                     &instCounter );
      input.getline( array, n );
   }
}

// Separate Tokens tokenizes a Simple statement, process the line number,
// and passes the next token to keyWord for processing.
void separateToken( char array[], int flags[], int machineArray[],
                    TableEntry symbolTable[], int *dataCounterPtr,
                    int *instCounterPtr )
{
   char *tokenPtr = strtok( array, " " );     // tokenize line
   lineNumber( tokenPtr, symbolTable, *instCounterPtr, *dataCounterPtr );
   tokenPtr = strtok( 0, " \n" );      // get next token
   keyWord( tokenPtr, flags, machineArray, symbolTable, dataCounterPtr,
            instCounterPtr );
}

// checkSymbolTable searches the symbol table and returns
// the symbols SML location or a -1 if not found.
int checkSymbolTable( TableEntry symbolTable[], int symbol, char type )
{
   for ( int loop = 0; loop < SYMBOLTABLESIZE; ++loop )
      if ( ( symbol == symbolTable[ loop ].symbol ) &&
           ( type == symbolTable[ loop ].type ) )
         return symbolTable[ loop ].location;  // return SML location

   return -1;                                 // symbol not found
}

// lineNumber processes line numbers
void lineNumber( char *tokenPtr, TableEntry symbolTable[],
                 int instCounter, int dataCounter )
{
   const char type = 'L';
   int symbol;

   if ( isdigit( tokenPtr[ 0 ] ) ) {
      symbol = atoi( tokenPtr );

      if ( -1 == checkSymbolTable( symbolTable, symbol, type ) )
         addToSymbolTable( type, symbol, dataCounter, symbolTable,
                           instCounter );
   }

}

// keyWord determines the key word type and calls the appropriate function
void keyWord( char *tokenPtr, int flags[], int machineArray[],
                TableEntry symbolTable[], int *dataCounterPtr,
                int *instCounterPtr )
{
   if ( strcmp( tokenPtr, "rem" ) == 0 )
      ; // no instructions are generated by comments
   else if ( strcmp( tokenPtr, "input" ) == 0 ) {
      tokenPtr = strtok( 0, " " );  // assign pointer to next token
      keyInput( tokenPtr, machineArray, symbolTable, dataCounterPtr,
                instCounterPtr );
   }
   else if ( strcmp( tokenPtr, "print" ) == 0 ) {
      tokenPtr = strtok( 0, " " );  // assign pointer to next token
      keyPrint( tokenPtr, machineArray, symbolTable, dataCounterPtr,
                instCounterPtr );
   }
   else if ( strcmp( tokenPtr, "goto" ) == 0 ) {
      tokenPtr = strtok( 0, " " );  // assign pointer to next token
      keyGoto( tokenPtr, flags, machineArray, symbolTable, instCounterPtr );
   }
   else if ( strcmp( tokenPtr, "if" ) == 0 ) {
      tokenPtr = strtok( 0, " " );  // assign pointer to next token
      keyIfGoto( tokenPtr, flags, machineArray, symbolTable, dataCounterPtr,
                 instCounterPtr );
   }
   else if ( strcmp( tokenPtr, "end" ) == 0 ) {
      machineArray[ *instCounterPtr ] = HALT * 100;
      ++( *instCounterPtr );
      tokenPtr = 0;       // assign tokenPtr to 0
   }
   else if ( strcmp( tokenPtr, "let" ) == 0 ) {
      tokenPtr = strtok( 0, " " );  // assign pointer to next token
      keyLet( tokenPtr, machineArray, symbolTable, dataCounterPtr,
              instCounterPtr );
   }
}

// keyInput process input keywords
void keyInput( char *tokenPtr, int machineArray[], TableEntry symbolTable[],
               int *dataCounterPtr, int *instCounterPtr )
{
   const char type = 'V';

   machineArray[ *instCounterPtr ] = READ * 100;
   int symbol = tokenPtr[ 0 ];
   int tableTest = checkSymbolTable( symbolTable, symbol, type );

   if ( -1 == tableTest ) {
      addToSymbolTable( type, symbol, *dataCounterPtr, symbolTable,
                        *instCounterPtr );
      machineArray[ *instCounterPtr ] += *dataCounterPtr;
      --( *dataCounterPtr );
   }
   else
      machineArray[ *instCounterPtr ] += tableTest;

   ++( *instCounterPtr );
}

// keyPrint process print keywords
void keyPrint( char *tokenPtr, int machineArray[], TableEntry symbolTable[],
               int *dataCounterPtr, int *instCounterPtr )
{
   const char type = 'V';

   machineArray[ *instCounterPtr ] = WRITE * 100;
   int symbol = tokenPtr[ 0 ];
   int tableTest = checkSymbolTable( symbolTable, symbol, type );

   if ( -1 == tableTest ) {
      addToSymbolTable( type, symbol, *dataCounterPtr, symbolTable,
                        *instCounterPtr );
      machineArray[*instCounterPtr] += *dataCounterPtr;
      --( *dataCounterPtr );
   }
   else
      machineArray[ *instCounterPtr ] += tableTest;

   ++( *instCounterPtr );
}

// keyGoto process goto keywords
void keyGoto( char *tokenPtr, int flags[], int machineArray[],
              TableEntry symbolTable[], int *instCounterPtr )
{
   const char type = 'L';

   machineArray[*instCounterPtr] = BRANCH * 100;
   int symbol = atoi( tokenPtr );
   int tableTest = checkSymbolTable( symbolTable, symbol, type );
   addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
   ++( *instCounterPtr );
}

// keyIfGoto process if/goto commands
void keyIfGoto( char *tokenPtr, int flags[], int machineArray[],
                TableEntry symbolTable[], int *dataCounterPtr,
                int *instCounterPtr )
{
   int operand1Loc = checkOperand( symbolTable, tokenPtr, dataCounterPtr,
                                   instCounterPtr, machineArray );

   char *operatorPtr = strtok( 0, " " );   // get the operator

   tokenPtr = strtok( 0, " " );  // get the right operand of comparison operator

   int operand2Loc = checkOperand( symbolTable, tokenPtr, dataCounterPtr,
                                   instCounterPtr, machineArray );

   tokenPtr = strtok( 0, " " );  // read in the goto keyword

   char *gotoLinePtr = strtok( 0, " " );  // read in the goto line number

   evaluateExpression( operand1Loc, operand2Loc, operatorPtr, machineArray,
                       symbolTable, instCounterPtr, flags, gotoLinePtr );
}

// checkOperand ensures that the operands of an if/goto statement are
// in the symbol table.
int checkOperand( TableEntry symbolTable[], char *symPtr, int *dataCounterPtr,
                  int *instCounterPtr, int machineArray[] )
{
   char type;
   int tableTest, operand, temp;

   if ( isalpha( symPtr[ 0 ] ) ) {
      type = 'V';
      operand = symPtr[ 0 ];
      tableTest = checkSymbolTable( symbolTable, operand, type );

      if ( tableTest == -1 ) {
         addToSymbolTable( type, operand, *dataCounterPtr, symbolTable,
                           *instCounterPtr );
         temp = *dataCounterPtr;
         --( *dataCounterPtr );
         return temp;
      }
      else
         return tableTest;
   }
   // if the symbol is a digit or a signed digit
   else if ( isdigit( symPtr[ 0 ] ) ||
            ( ( symPtr[ 0 ] == '-' || symPtr[ 0 ] == '+' ) &&
            isdigit( symPtr[ 1 ] ) != 0 ) ) {
      type = 'C';
      operand = atoi( symPtr );
      tableTest = checkSymbolTable( symbolTable, operand, type );

      if ( tableTest == -1 ) {
         addToSymbolTable( type, operand, *dataCounterPtr, symbolTable,
                          *instCounterPtr );
         machineArray[ *dataCounterPtr ] = operand;
         temp = *dataCounterPtr;
         --( *dataCounterPtr );
         return temp ;
      }
      else
         return tableTest;
   }

   return 0;       // default return for compilation purposes
}

// evaluateExpression creates SML for conditional operators
void evaluateExpression( int operator1Loc, int operator2Loc, char *operandPtr,
                         int machineArray[], TableEntry symbolTable[],
                         int *instCounterPtr, int flags[], char *gotoLinePtr )
{
   const char type = 'L';
   int tableTest, symbol;

   if ( strcmp( operandPtr, "==" ) == 0 ) {
      machineArray[ *instCounterPtr ] = LOAD * 100;
      machineArray[ *instCounterPtr ] += operator1Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = SUBTRACT * 100;
      machineArray[ *instCounterPtr ] += operator2Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = BRANCHZERO * 100;

      symbol = atoi( gotoLinePtr );
      tableTest = checkSymbolTable( symbolTable, symbol, type );
      addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
      ++( *instCounterPtr );
   }
   else if ( strcmp( operandPtr, "!=" ) == 0 ) {
      machineArray[ *instCounterPtr ] = LOAD * 100;
      machineArray[ *instCounterPtr ] += operator2Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = SUBTRACT * 100;
      machineArray[ *instCounterPtr ] += operator1Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = BRANCHNEG * 100;

      symbol = atoi( gotoLinePtr );
      tableTest = checkSymbolTable( symbolTable, symbol, type );

      addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );

      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = LOAD * 100;
      machineArray[ *instCounterPtr ] += operator1Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = SUBTRACT * 100;
      machineArray[ *instCounterPtr ] += operator2Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = BRANCHNEG * 100;

      symbol = atoi( gotoLinePtr );
      tableTest = checkSymbolTable( symbolTable, symbol, type );

      addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );

      ++( *instCounterPtr );
   }
   else if ( strcmp( operandPtr, ">" ) == 0 ) {
      machineArray[ *instCounterPtr ] = LOAD * 100;
      machineArray[ *instCounterPtr ] += operator2Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = SUBTRACT * 100;
      machineArray[ *instCounterPtr ] += operator1Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = BRANCHNEG * 100;

      symbol = atoi( gotoLinePtr );
      tableTest = checkSymbolTable( symbolTable, symbol, type );

      addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
      ++( *instCounterPtr );
   }
   else if ( strcmp( operandPtr, "<" ) == 0 ) {
      machineArray[ *instCounterPtr ] = LOAD * 100;
      machineArray[ *instCounterPtr ] += operator1Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = SUBTRACT * 100;
      machineArray[ *instCounterPtr ] += operator2Loc;
      ++( *instCounterPtr );

      machineArray[ *instCounterPtr ] = BRANCHNEG * 100;
```

```
         symbol = atoi( gotoLinePtr );
         tableTest = checkSymbolTable( symbolTable, symbol, type );

         addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
         ++( *instCounterPtr );
      }
      else if ( strcmp( operandPtr, ">=" ) == 0 ) {
         machineArray[ *instCounterPtr ] = LOAD * 100;
         machineArray[ *instCounterPtr ] += operator2Loc;
         ++( *instCounterPtr );

         machineArray[ *instCounterPtr ] = SUBTRACT * 100;
         machineArray[ *instCounterPtr ] += operator1Loc;
         ++( *instCounterPtr );

         machineArray[ *instCounterPtr ] = BRANCHNEG * 100;

         symbol = atoi( gotoLinePtr );
         tableTest = checkSymbolTable( symbolTable, symbol, type );

         addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
         ++( *instCounterPtr );

         machineArray[ *instCounterPtr ] = BRANCHZERO * 100;

         addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
         ++( *instCounterPtr );
      }
      else if ( strcmp( operandPtr, "<=" ) == 0 ) {
         machineArray[ *instCounterPtr ] = LOAD * 100;
         machineArray[ *instCounterPtr ] += operator1Loc;
         ++( *instCounterPtr );

         machineArray[ *instCounterPtr ] = SUBTRACT * 100;
         machineArray[ *instCounterPtr ] += operator2Loc;
         ++( *instCounterPtr );

         machineArray[ *instCounterPtr ] = BRANCHNEG * 100;

         symbol = atoi( gotoLinePtr );
         tableTest = checkSymbolTable( symbolTable, symbol, type );

         addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
         ++( *instCounterPtr );

         machineArray[ *instCounterPtr ] = BRANCHZERO * 100;

         addLineToFlags( tableTest, symbol, flags, machineArray, instCounterPtr );
         ++( *instCounterPtr );
      }
   }

   // secondPass resolves incomplete SML instructions for forward references
   void secondPass( int flags[], int machineArray[], TableEntry symbolTable[] )
   {
      const char type = 'L';

      for ( int loop = 0; loop < MEMORYSIZE; ++loop ) {
         if ( flags[ loop ] != -1 ) {
            int symbol = flags[ loop ];
            int flagLocation = checkSymbolTable( symbolTable, symbol, type );
            machineArray[ loop ] += flagLocation;
         }
      }
   }

   // keyLet processes the keyword let
   void keyLet( char *tokenPtr, int machineArray[], TableEntry symbolTable[],
                int *dataCounterPtr, int *instCounterPtr )
   {
      const char type = 'V';
      char infixArray[ MAXIMUM ] = "", postfixArray[ MAXIMUM ] = "";
      int tableTest, symbol, location;
      static int subscript = 0;

      symbol = tokenPtr[ 0 ];
      tableTest = checkSymbolTable( symbolTable, symbol, type );

      if ( -1 == tableTest ) {
         addToSymbolTable( type, symbol, *dataCounterPtr, symbolTable,
                           *instCounterPtr );
         location = *dataCounterPtr;
         --( *dataCounterPtr );
      }
      else
         location = tableTest;

      tokenPtr = strtok( 0, " " );     // grab equal sign
      tokenPtr = strtok( 0, " " );     // get next token

      while ( tokenPtr != 0 ) {
         checkOperand( symbolTable, tokenPtr, dataCounterPtr,
                       instCounterPtr, machineArray );
         infixArray[ subscript ] = tokenPtr[ 0 ];
         ++subscript;
         tokenPtr = strtok( 0, " " );  // get next token
      }

      infixArray[ subscript ] = '\0';

      infixToPostfix( infixArray, postfixArray, location, symbolTable,
                      instCounterPtr, dataCounterPtr, machineArray );

      subscript = 0;    // reset static subscript when done
   }

   void addToSymbolTable( char type, int symbol, int dataCounter,
                          TableEntry symbolTable[], int instCounter )
   {
      static int symbolCounter = 0;

      symbolTable[ symbolCounter ].type = type;
      symbolTable[ symbolCounter ].symbol = symbol;

      if ( type == 'V' || type == 'C' )
         symbolTable[ symbolCounter ].location = dataCounter;
      else
         symbolTable[ symbolCounter ].location = instCounter;

      ++symbolCounter;
   }

   void addLineToFlags( int tableTest, int symbol, int flags[],
                         int machineArray[], const int *instCounterPtr )
   {
      if ( tableTest == -1 )
         flags[ *instCounterPtr ] = symbol;
      else
         machineArray[ *instCounterPtr ] += tableTest;
   }

   void printOutput( const int machineArray[], char *outFileName )
   {
      ofstream output( outFileName, ios::out );

      if ( !output )
         cerr << "File was not opened.\n";
      else                  // output every memory cell
         for ( int loop = 0; loop <= MEMORYSIZE - 1; ++loop )
            output << machineArray[ loop ] << '\n';
   }

   void initArrays( int flags[], int machineArray[], TableEntry symbolTable[] )
   {
      TableEntry initEntry = { 0, 0, -1 };

      for ( int loop = 0; loop < MEMORYSIZE; ++loop ) {
         flags[ loop ] = -1;
         machineArray[ loop ] = 0;
         symbolTable[ loop ] = initEntry;
      }
   }

   //////////////////////////////////////////////////////////////////////////////
   // INFIX TO POSTFIX CONVERSION and POSTFIX EVALUATION FOR THE LET STATEMENT //
   //////////////////////////////////////////////////////////////////////////////

   // infixToPostfix converts an infix expression to a postfix expression
   void infixToPostfix( char infix[], char postfix[], int getsVariable,
                        TableEntry symbolTable[], int *instCounterPtr,
                        int *dataCounterPtr, int machineArray[] )
   {
      Stack< int > intStack;
      int infixCount, postfixCount, popValue;
      bool higher;
      int leftParen = '(';   // made int

      // push a left paren onto the stack and add a right paren to infix
      intStack.push( leftParen );
      strcat( infix, ")" );

      // convert the infix expression to postfix
      for ( infixCount = 0, postfixCount = 0; intStack.stackTop();
            ++infixCount ) {

         if ( isalnum( infix[ infixCount ] ) )
            postfix[ postfixCount++ ] = infix[ infixCount ];
         else if ( infix[ infixCount ] == '(' )
            intStack.push( leftParen );
         else if ( isOperator( infix[ infixCount ] ) ) {
            higher = true;   // used to store value of precedence test

            while ( higher ) {
               if ( isOperator( static_cast< char >( intStack.stackTop() ) ) )
                  if ( precedence( static_cast< char >( intStack.stackTop() ),
                       infix[ infixCount ] ) )

                     postfix[ postfixCount++ ] =
                                        static_cast< char >( intStack.pop() );
                  else
                     higher = false;
               else
                  higher = false;
            }

            // See chapter 21 for a discussion of reinterpret_cast
            intStack.push( reinterpret_cast< int & > ( infix[ infixCount ] ) );
         }
         else if ( infix[ infixCount ] == ')' )
            while ( ( popValue = intStack.pop() ) != '(' )
               postfix[ postfixCount++ ] = static_cast< char >( popValue );
      }

      postfix[ postfixCount ] = '\0';

      evaluatePostfix( postfix, dataCounterPtr, instCounterPtr,
                       machineArray, getsVariable, symbolTable );
   }

   // check if c is an operator
   bool isOperator( char c )
   {
      if ( c == '+' || c == '-' || c == '*' || c == '/' || c == '^' )
         return true;
      else
         return false;
   }

   // If the precedence of operator1 is >= operator2,
   bool precedence( char operator1, char operator2 )
   {
      if ( operator1 == '^' )
         return true;
      else if ( operator2 == '^' )
         return false;
      else if ( operator1 == '*' || operator1 == '/' )
         return true;
      else if ( operator1 == '+' || operator1 == '-' )
         if ( operator2 == '*' || operator2 == '/' )
            return false;
         else
            return true;

      return false;
   }

   // evaluate postfix expression and produce code
   void evaluatePostfix( char *expr, int *dataCounterPtr,
                         int *instCounterPtr, int machineArray[],
                         int getsVariable, TableEntry symbolTable[] )
   {
      Stack< int > intStack;
      int popRightValue, popLeftValue, accumResult, symbolLocation, symbol;
      char type, array[ 2 ] = "";
      int i;

      strcat( expr, ")" );
```

```

   for ( i = 0; expr[ i ] != ')'; ++i )
      if ( isdigit( expr[ i ] ) ) {
         type = 'C';
         array[ 0 ] = expr[ i ];
         symbol = atoi( array );

         symbolLocation = checkSymbolTable( symbolTable, symbol, type );
         intStack.push( symbolLocation );
      }
      else if ( isalpha( expr[ i ] ) ) {
         type = 'V';
         symbol = expr[ i ];
         symbolLocation = checkSymbolTable( symbolTable, symbol, type );
         intStack.push( symbolLocation );
      }
      else {
         popRightValue = intStack.pop();
         popLeftValue = intStack.pop();
         accumResult = createLetSML( popRightValue, popLeftValue, machineArray,
                                     instCounterPtr, dataCounterPtr,
                                     expr[ i ] );
         intStack.push( accumResult );
      }

      machineArray[ *instCounterPtr ] = LOAD * 100;
      machineArray[ *instCounterPtr ] += intStack.pop();
      ++( *instCounterPtr );
      machineArray[ *instCounterPtr ] = STORE * 100;
      machineArray[ *instCounterPtr ] += getsVariable;
      ++( *instCounterPtr );
}

int createLetSML( int right, int left, int machineArray[],
                  int *instCounterPtr, int *dataCounterPtr, char oper )
{
   int location;

   switch( oper ) {
      case '+':
         machineArray[ *instCounterPtr ] = LOAD * 100;
         machineArray[ *instCounterPtr ] += left;
         ++( *instCounterPtr );
         machineArray[ *instCounterPtr ] = ADD * 100;
         machineArray[ *instCounterPtr ] += right;
         ++( *instCounterPtr );
         machineArray[ *instCounterPtr ] = STORE * 100;
         machineArray[ *instCounterPtr ] += *dataCounterPtr;
         location = *dataCounterPtr;
         --( *dataCounterPtr );
         ++( *instCounterPtr );
         return location;
      case '-':
         machineArray[ *instCounterPtr ] = LOAD * 100;
         machineArray[ *instCounterPtr ] += left;
         ++( *instCounterPtr );
         machineArray[ *instCounterPtr ] = SUBTRACT * 100;
         machineArray[ *instCounterPtr ] += right;

         ++( *instCounterPtr );
         machineArray[ *instCounterPtr ] = STORE * 100;
         machineArray[ *instCounterPtr ] += *dataCounterPtr;
         location = *dataCounterPtr;
```

```
            --( *dataCounterPtr );
            ++( *instCounterPtr );
            return location;
         case '/':
            machineArray[ *instCounterPtr ] = LOAD * 100;
            machineArray[ *instCounterPtr ] += left;
            ++( *instCounterPtr );
            machineArray[ *instCounterPtr ] = DIVIDE * 100;
            machineArray[ *instCounterPtr ] += right;
            ++( *instCounterPtr );
            machineArray[ *instCounterPtr ] = STORE * 100;
            machineArray[ *instCounterPtr ] += *dataCounterPtr;
            location = *dataCounterPtr;
            --( *dataCounterPtr );
            ++( *instCounterPtr );
            return location;
         case '*':
            machineArray[ *instCounterPtr ] = LOAD * 100;
            machineArray[ *instCounterPtr ] += left;
            ++( *instCounterPtr );
            machineArray[ *instCounterPtr ] = MULTIPLY * 100;
            machineArray[ *instCounterPtr ] += right;
            ++( *instCounterPtr );
            machineArray[ *instCounterPtr ] = STORE * 100;
            machineArray[ *instCounterPtr ] += *dataCounterPtr;
            location = *dataCounterPtr;
            --( *dataCounterPtr );
            ++( *instCounterPtr );
            return location;
         default:
            cerr << "ERROR: operator not recognized.\n";
            break;
      }

      return 0;    // default return
   }
```

15.28 (*Optimizing the Simple Compiler*) When a program is compiled and converted into SML, a set of instructions is generated. Certain combinations of instructions often repeat themselves, usually in triplets called*productions*. A production normally consists of three instructions such as*load*, *add* and *store*. For example, Fig. 15.27 illustrates five of the SML instructions that were produced in the compilation of the program in Fig. 15.25 The first three instructions are the production that adds`1` to `y`. Note that instructions `06` and `07` store the accumulator value in temporary location `96` and then load the value back into the accumulator so instruction `08` can store the value in location`98`. Often a production is followed by a load instruction for the same location that was just stored. This code can be *optimized* by eliminating the store instruction and the subsequent load instruction that operate on the same memory location, thus enabling the Simpletron to execute the program faster. Figure 15.28 illustrates the optimized SML for the programof Fig. 15.25. Note that there are four fewer instructions in the optimized code—a memory-space savings of 25%.

Modify the compiler to provide an option for optimizing the Simpletron Machine Language code it produces. Manually compare the nonoptimized code with the optimized code, and calculate the percentage reduction.

15.29 (*Modifications to the Simple compiler*) Perform the following modifications to the Simple compiler. Some of these modifications may also require modifications to the Simpletron Simulator program written in Exercise 5.19.

a) Allow the modulus operator (`%`) to be used in `let` statements. Simpletron Machine Language must be modified to include a modulus instruction.
b) Allow exponentiation in a `let` statement using `^` as the exponentiation operator. Simpletron Machine Language must be modified to include an exponentiation instruction.

```
04   +2098   (load)
05   +3097   (add)
```

Fig. 15.27 Nonoptimized code from the program of Fig. 15.25.

```
06   +2196  (store)
07   +2096  (load)
08   +2198  (store)
```

Fig. 15.27 Nonoptimized code from the program of Fig. 15.25.

Simple program	SML location and instruction	Description
`5 rem sum 1 to x`	*none*	**rem** ignored
`10 input x`	`00 +1099`	read **x** into location **99**
`15 rem   check y == x`	*none*	**rem** ignored
`20 if y == x goto 60`	`01 +2098`	load **y** (**98**) into accumulator
	`02 +3199`	sub **x** (**99**) from accumulator
	`03 +4211`	branch to location **11** if zero
`25 rem   increment y`	*none*	**rem** ignored
`30 let y = y + 1`	`04 +2098`	load **y** into accumulator
	`05 +3097`	add **1** (**97**) to accumulator
	`06 +2198`	store accumulator in **y** (**98**)
`35 rem   add y to total`	*none*	**rem** ignored
`40 let t = t + y`	`07 +2096`	load **t** from location (**96**)
	`08 +3098`	add **y** (**98**) accumulator
	`09 +2196`	store accumulator in **t** (**96**)
`45 rem   loop y`	*none*	**rem** ignored
`50 goto 20`	`10 +4001`	branch to location **01**
`55 rem   output result`	*none*	**rem** ignored
`60 print t`	`11 +1196`	output **t** (**96**) to screen
`99 end`	`12 +4300`	terminate execution

Fig. 15.28 Optimized code for the program of Fig. 15.25.

c) Allow the compiler to recognize uppercase and lowercase letters in Simple statements (e.g., **'A'** is equivalent to **'a'**). No modifications to the Simulator are required.

d) Allow **input** statements to read values for multiple variables such as **input x, y**. No modifications to the Simpletron Simulator are required.

e) Allow the compiler to output multiple values in a single **print** statement such as **print a, b, c**. No modifications to the Simpletron Simulator are required.

f) Add syntax-checking capabilities to the compiler so error messages are output when syntax errors are encountered in a Simple program. No modifications to the Simpletron Simulator are required.

g) Allow arrays of integers. No modifications to the Simpletron Simulator are required.

h) Allow subroutines specified by the Simple commands **gosub** and **return**. Command **gosub** passes program control to a subroutine, and command **return** passes control back to the statement after the **gosub**. This is similar to a function call in C++. The same subroutine can be called from many **gosub** commands distributed throughout a program. No modifications to the Simpletron Simulator are required.

i) Allow repetition structures of the form

```
for x = 2 to 10 step 2
   Simple statements
next
```

This **for** statement loops from **2** to **10** with an increment of **2**. The **next** line marks the end of the body of the **for** line. No modifications to the Simpletron Simulator are required.

j) Allow repetition structures of the form

```
for x = 2 to 10
   Simple statements
next
```

This **`for`** statement loops from **`2`** to **`10`** with a default increment of **`1`**. No modifications to the Simpletron Simulator are required.

k) Allow the compiler to process string input and output. This requires the Simpletron Simulator to be modified to process and store string values. (Hint: Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII decimal equivalent of a character. Add a machine language instruction that will print a string beginning at a certain Simpletron memory location. The first half of the word at that location is a count of the number of characters in the string (i.e., the length of the string). Each succeeding half word contains one ASCII character expressed as two decimal digits. The machine language instruction checks the length and prints the string by translating each two-digit number into its equivalent character.)

l) Allow the compiler to process floating-point values in addition to integers. The Simpletron Simulator must also be modified to process floating-point values.

15.30 (*A Simple interpreter*) An interpreter is a program that reads a high-level language program statement, determines the operation to be performed by the statement and executes the operation immediately. The high-level language program is not converted into machine language first. Interpreters execute slowly because each statement encountered in the program must first be deciphered. If statements are contained in a loop, the statements are deciphered each time they are encountered in the loop. Early versions of the BASIC programming language were implemented as interpreters.

Write an interpreter for the Simple language discussed in Exercise 15.26. The program should use the infix-to-postfix converter developed in Exercise 15.12 and the postfix evaluator developed in Exercise 15.13 to evaluate expressions in a **`let`** statement. The same restrictions placed on the Simple language in Exercise 15.26 should be adhered to in this program. Test the interpreter with the Simple programs written in Exercise 15.26. Compare the results of running these programs in the interpreter with the results of compiling the Simple programs and running them in the Simpletron Simulator built in Exercise 5.19.

15.31 *(Insert/Delete Anywhere in a Linked List)* Our linked list class template allowed insertions and deletions at only the front and the back of the linked list. These capabilities were convenient for us when we used private inheritance and composition to produce a stack class template and a queue class template with a minimal amount of code by reusing the list class template. Actually, linked lists are more general that those we provided. Modify the linked list class template we developed in this chapter to handle insertions and deletions anywhere in the list.

```
// LIST.H
// Template List class definition
// Added copy constructor to member functions (not included in chapter).
#ifndef LIST_H
#define LIST_H

#include <iostream>

using std::cout;

#include <cassert>
#include "listnd.h"

template< class NODETYPE >
class List {
public:
   List();                            // default constructor
   List( const List< NODETYPE > & );  // copy constructor
   ~List();                           // destructor
   void insertAtFront( const NODETYPE & );
   void insertAtBack( const NODETYPE & );
   bool removeFromFront( NODETYPE & );
   bool removeFromBack( NODETYPE & );
   void insertInOrder( const NODETYPE & );
   bool isEmpty() const;
   void print() const;
protected:
   ListNode< NODETYPE > *firstPtr;  // pointer to first node
   ListNode< NODETYPE > *lastPtr;   // pointer to last node

   // Utility function to allocate a new node
   ListNode< NODETYPE > *getNewNode( const NODETYPE & );
};

// Default constructor
template< class NODETYPE >
List< NODETYPE >::List() { firstPtr = lastPtr = 0; }

// Copy constructor
template< class NODETYPE >
List< NODETYPE >::List( const List<NODETYPE> &copy )
{
   firstPtr = lastPtr = 0;  // initialize pointers

   ListNode< NODETYPE > *currentPtr = copy.firstPtr;

   while ( currentPtr != 0 ) {
      insertAtBack( currentPtr -> data );
      currentPtr = currentPtr -> nextPtr;
   }
}

// Destructor
template< class NODETYPE >
List< NODETYPE >::~List()
{
   if ( !isEmpty() ) {    // List is not empty
      cout << "Destroying nodes ...\n";

      ListNode< NODETYPE > *currentPtr = firstPtr, *tempPtr;

      while ( currentPtr != 0 ) {  // delete remaining nodes
         tempPtr = currentPtr;
         cout << tempPtr -> data << ' ';
         currentPtr = currentPtr -> nextPtr;
         delete tempPtr;
      }
   }

   cout << "\nAll nodes destroyed\n\n";
}

// Insert a node at the front of the list
template< class NODETYPE >
void List< NODETYPE >::insertAtFront( const NODETYPE &value )
{
   ListNode<NODETYPE> *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {          // List is not empty
      newPtr -> nextPtr = firstPtr;
      firstPtr = newPtr;
   }
}

// Insert a node at the back of the list
template< class NODETYPE >
```

```
void List< NODETYPE >::insertAtBack( const NODETYPE &value )
{
   ListNode< NODETYPE > *newPtr = getNewNode( value );

   if ( isEmpty() )  // List is empty
      firstPtr = lastPtr = newPtr;
   else {           // List is not empty
      lastPtr -> nextPtr = newPtr;
      lastPtr = newPtr;
   }
}

// Delete a node from the front of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromFront( NODETYPE &value )
{
   if ( isEmpty() )             // List is empty
      return false;             // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = firstPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else
         firstPtr = firstPtr -> nextPtr;

      value = tempPtr -> data;  // data being removed
      delete tempPtr;
      return true;              // delete successful
   }
}

// Delete a node from the back of the list
template< class NODETYPE >
bool List< NODETYPE >::removeFromBack( NODETYPE &value )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful
   else {
      ListNode< NODETYPE > *tempPtr = lastPtr;

      if ( firstPtr == lastPtr )
         firstPtr = lastPtr = 0;
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr;

         while ( currentPtr -> nextPtr != lastPtr )
            currentPtr = currentPtr -> nextPtr;

         lastPtr = currentPtr;
         currentPtr -> nextPtr = 0;
      }

      value = tempPtr -> data;
      delete tempPtr;
      return true;   // delete successful
   }
}

// Is the List empty?
template< class NODETYPE >
bool List< NODETYPE >::isEmpty() const { return firstPtr == 0; }
```

```

// Return a pointer to a newly allocated node
template< class NODETYPE >
ListNode< NODETYPE > *List< NODETYPE >::getNewNode( const NODETYPE &value )
{
   ListNode< NODETYPE > *ptr = new ListNode< NODETYPE >( value );
   assert( ptr != 0 );
   return ptr;
}

// Display the contents of the List
template< class NODETYPE >
void List< NODETYPE >::print() const
{
   if ( isEmpty() ) {
      cout << "The list is empty\n\n";
      return;
   }

   ListNode< NODETYPE > *currentPtr = firstPtr;

   cout << "The list is: ";

   while ( currentPtr != 0 ) {
      cout << currentPtr -> data << ' ';
      currentPtr = currentPtr -> nextPtr;
   }

   cout << "\n\n";
}

template< class NODETYPE >
void List< NODETYPE >::insertInOrder( const NODETYPE &value )
{
   if ( isEmpty() ) {
      ListNode< NODETYPE > *newPtr = getNewNode( value );
      firstPtr = lastPtr = newPtr;
   }
   else {
      if ( firstPtr -> data > value )
         insertAtFront( value );
      else if ( lastPtr -> data < value )
         insertAtBack( value );
      else {
         ListNode< NODETYPE > *currentPtr = firstPtr -> nextPtr,
                              *previousPtr = firstPtr,
                              *newPtr = getNewNode( value );

         while ( currentPtr != lastPtr && currentPtr -> data < value ) {
            previousPtr = currentPtr;
            currentPtr = currentPtr -> nextPtr;
         }

         previousPtr -> nextPtr = newPtr;
         newPtr -> nextPtr = currentPtr;
      }
   }
}

#endif
```

```
// LISTND.H
// ListNode template definition
#ifndef LISTND_H
#define LISTND_H

template< class T > class List;  // forward declaration

template< class NODETYPE >
class ListNode {
   friend class List< NODETYPE >; // make List a friend
public:
   ListNode( const NODETYPE & );  // constructor
   NODETYPE getData() const;    // return the data in the node
   void setNextPtr( ListNode *nPtr ) { nextPtr = nPtr; }
   ListNode *getNextPtr() const { return nextPtr; }
private:
   NODETYPE data;               // data
   ListNode *nextPtr;           // next node in the list
};

// Constructor
template< class NODETYPE >
ListNode< NODETYPE >::ListNode( const NODETYPE &info )
{
   data = info;
   nextPtr = 0;
}

// Return a copy of the data in the node
template< class NODETYPE >
NODETYPE ListNode< NODETYPE >::getData() const { return data; }

#endif
```

```
// LIST2.H
// Template List class definition
// NOTE: This solution only provides the delete anywhere operation.

#ifndef LIST2_H
#define LIST2_H

#include <cassert>
#include "listnd.h"
#include "list.h"

template< class NODETYPE >
class List2 : public List< NODETYPE > {
public:
   bool deleteNode( const NODETYPE &, NODETYPE & );
};

// Delete a node from anywhere in the list
template< class NODETYPE >
bool List2< NODETYPE >::deleteNode( const NODETYPE &val, NODETYPE &deletedVal )
{
   if ( isEmpty() )
      return false;   // delete unsuccessful

   else {
      if ( firstPtr -> getData() == val ) {
```

```cpp
            removeFromFront( deletedVal );
            return true;  // delete successful
         }
         else if ( lastPtr -> getData() == val ) {
            removeFromBack( deletedVal );
            return true;  // delete successful
         }
         else {
            ListNode< NODETYPE > *currentPtr = firstPtr -> getNextPtr(),
                                 *previousPtr = firstPtr;

            while ( currentPtr != lastPtr && currentPtr -> getData() < val ) {
               previousPtr = currentPtr;
               currentPtr = currentPtr -> getNextPtr();
            }

            if ( currentPtr -> getData() == val ) {
               ListNode< NODETYPE > *tempPtr = currentPtr;
               deletedVal = currentPtr -> getData();
               previousPtr -> setNextPtr( currentPtr -> getNextPtr() );
               delete tempPtr;
               return true;  // delete successful
            }
            else
               return false;  // delete unsuccessful
         }
      }
   }

#endif
```

```cpp
// Exercise 15.31 solution
#include <iostream>

using std::cout;
using std::cin;

#include <cstdlib>
#include <ctime>
#include "list2.h"

int main()
{
   srand( time( 0 ) );  // randomize the random number generator

   List2< int > intList;

   for ( int i = 1; i <= 10; ++i )
      intList.insertInOrder( rand() % 101 );

   intList.print();

   int value, deletedValue;

   cout << "Enter an integer to delete (-1 to end): ";
   cin >> value;

   while ( value != -1 ) {
      if ( intList.deleteNode( value, deletedValue ) ) {
         cout << deletedValue << " was deleted from the list\n";
         intList.print();
      }
      else
         cout << "Element was not found";

      cout << "Enter an integer to delete (-1 to end): ";
      cin >> value;
   }

   return 0;
}
```

```
The list is: 0 3 20 35 39 51 61 62 64 82

Enter an integer to delete (-1 to end): 8
Element was not foundEnter an integer to delete (-1 to end): 82
82 was deleted from the list
The list is: 0 3 20 35 39 51 61 62 64

Enter an integer to delete (-1 to end): -1
Destroying nodes ...
0 3 20 35 39 51 61 62 64
All nodes destroyed
```

15.32 *(List and Queues without Tail Pointers)* Our implementation of a linked list (Fig. 15.3) used both a **firstPtr** and a **lastPtr**. The **lastPtr** was useful for the **insertAtBack** and **removeFromBack** member functions of the **List** class. The **insertAtBack** function corresponds to the **enqueue** member function of the **Queue** class. Rewrite the **List** class so that it does not use a **lastPtr**. Thus, any operations on the tail of a list must begin searching the list from the front. Does this affect our implementation of the **Queue** class (Fig. 15.12)?

15.33 Use the composition version of the stack program (Fig. 15.11) to form a complete working stack program. Modify this program to **inline** the member functions. Compare the two approaches. Summarize the advantages and disadvantages of inlining member functions.

15.34 *(Performance of Binary Tree Sorting and Searching)* One problem with the binary tree sort is that the order in which the data are inserted affects the shape of the tree—for the same collection of data, different orderings can yield binary trees of dramatically different shapes. The performance of the binary tree sorting and searching algorithms is sensitive to the shape of the binary tree. What shape would a binary tree have if its data were inserted in increasing order? in decreasing order? What shape should the tree have to achieve maximal searching performance?

15.35 *(Indexed Lists)* As presented in the text, linked lists must be searched sequentially. For large lists, this can result in poor performance. A common technique for improving list searching performance is to create and maintain an index to the list. An index is a set of pointers to various key places in the list. For example, an application that searches a large list of names could improve performance by creating an index with 26 entries—one for each letter of the alphabet. A search operation for a last name beginning with 'Y' would then first search the index to determine where the 'Y' entries begin, and then "jump into" the list at that point and search linearly until the desired name is found. This would be much faster than searching the linked list from the beginning. Use the **List** class of Fig. 15.3 as the basis of an **IndexedList** class. Write a program that demonstrates the operation of indexed lists. Be sure to include member functions **insertInIndexedList**, **searchIndexedList** and **deleteFromIndexedList**.

16

Bits, Characters, Strings and Structures Solutions

Solutions

16.6 Provide the definition for each of the following structures and unions:

a) Structure **Inventory** containing character array **partName[30]**, integer **partNumber**, floating-point **price**, integer **stock** and integer **reorder**.

ANS:

```
struct Inventory {
   char partName[ 30 ];
   int partNumber;
   double price;
   int stock;
   int reorder;
};
```

b) A structure called **Address** that contains character arrays **streetAddress[25]**, **city[20]**, **state[3]** and **zipCode[6]**.

ANS:

```
struct Address {
   char streetAddress[ 25 ];
   char city[ 20 ];
   char state[ 3 ];
   char zipCode[ 6 ];
};
```

c) Structure **Student** that contains arrays **firstName[15]** and **lastName[15]**, and variable **homeAddress** of type **struct Address** from part (b).

ANS:

```
struct Student {
   char firstName[ 15 ];
   char lastName[ 15 ];
   struct Address homeAddress;
};
```

d) Structure **Test** containing 16 bit fields with widths of 1 bit. The names of the bit fields are the letters **a** to **p**.

ANS:
```
struct Test {
   unsigned a:1, b:1, c:1, d:1, e:1, f:1, g:1, h:1,
            i:1, j:1, k:1, l:1, m:1, n:1, o:1, p:1;
};
```

16.7 Consider the following structure definitions and variable declarations,

```
struct Customer {
   char lastName[ 15 ];
   char firstName[ 15 ];
   int customerNumber;

   struct {
      char phoneNumber[ 11 ];
      char address[ 50 ];
      char city[ 15 ];
      char state[ 3 ];
      char zipCode[ 6 ];
   } personal;
} customerRecord, *customerPtr;

customerPtr = &customerRecord;
```

Write a separate expression that accesses the structure members in each of the following parts:

a) Member `lastName` of structure `customerRecord`.
ANS: `customerRecord.lastName`
b) Member `lastName` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->lastName`
c) Member `firstName` of structure `customerRecord`.
ANS: `customerRecord.firstName`
d) Member `firstName` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->firstName`
e) Member `customerNumber` of structure `customerRecord`.
ANS: `customerRecord.customerNumber`
f) Member `customerNumber` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->customerNumber`
g) Member `phoneNumber` of member `personal` of structure `customerRecord`.
ANS: `customerRecord.personal.phoneNumber`
h) Member `phoneNumber` of member `personal` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->personal.phoneNumber`
i) Member `address` of member `personal` of structure `customerRecord`.
ANS: `customerRecord.personal.address`
j) Member `address` of member `personal` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->personal.address`
k) Member `city` of member `personal` of structure `customerRecord`.
ANS: `customerRecord.personal.city`
l) Member `city` of member `personal` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->personal.city`
m) Member `state` of member `personal` of structure `customerRecord.`
ANS: `customerRecord.personal.state`
n) Member `state` of member `personal` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->personal.state`
o) Member `zipCode` of member `personal` of structure `customerRecord`.
ANS: `customerRecord.personal.zipCode`
p) Member `zipCode` of member `personal` of the structure pointed to by `customerPtr`.
ANS: `customerPtr->personal.zipCode`

16.8 Modify the program of Fig. 16.14 to shuffle the cards using a high-performance shuffle as shown in Fig. 16.2. Print the resulting deck in two column format as in Fig. 16.3. Precede each card with its color.

```
// Exercise 16.8 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

#include <cstdlib>
#include <ctime>

struct bitCard {
   unsigned face : 4;
   unsigned suit : 2;
   unsigned color : 1;
};

typedef struct bitCard Card;

void fillDeck( Card * const );
void shuffle( Card * const );
void deal( const Card * const );

int main()
{
   Card deck[ 52 ];

   srand( 0 );

   fillDeck( deck );
   shuffle( deck );
   deal( deck );

   return 0;
}

void fillDeck( Card * const wDeck )
{
   for ( int i = 0; i < 52; ++i ) {
      wDeck[ i ].face = i % 13;
      wDeck[ i ].suit = i / 13;
      wDeck[ i ].color = i / 26;
   }
}

void shuffle( Card * const wDeck )
{
   int j;
   Card temp;

   for ( int i = 0; i < 52; ++i ) {
      j = rand() % 52;

      temp = wDeck[ i ];
      wDeck[ i ] = wDeck[ j ];
      wDeck[ j ] = temp;
   }
}

void deal( const Card * const wDeck2 )
{
   char *face[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
                    "Eight", "Nine", "Ten", "Jack", "Queen", "King" },
        *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades" },
        *color[] = { "Red", "Black" };

   for ( int i = 0; i < 52; ++i ) {

      cout << setw( 5 ) << color[ wDeck2[ i ].color ] << ": "
           << setw( 8 ) << face[ wDeck2[ i ].face ] << " of "
           << setiosflags( ios::left ) << setw( 8 )
           << suit[ wDeck2[ i ].suit ] << resetiosflags( ios::left );

      cout.put( ( i + 1 ) % 2 ? '\t' : '\n' );
   }
}
```

```
Black:     King of Clubs        Black:    Three of Clubs
Black:     Five of Clubs          Red:      Ten of Hearts
Black:    Deuce of Clubs          Red:     Four of Hearts
  Red:      Six of Hearts       Black:     Nine of Clubs
Black:    Seven of Clubs          Red:     Five of Hearts
Black:    Deuce of Spades         Red:    Eight of Diamonds
  Red:     Nine of Hearts         Red:     Nine of Diamonds
Black:    Three of Spades       Black:     Nine of Spades
  Red:    Seven of Diamonds       Red:     King of Diamonds
Black:      Six of Clubs          Red:     Jack of Diamonds
  Red:    Queen of Hearts       Black:      Ten of Clubs
  Red:    Deuce of Hearts         Red:      Six of Diamonds
Black:     Five of Spades       Black:     Four of Clubs
Black:     Jack of Clubs          Red:    Three of Diamonds
  Red:     Jack of Hearts         Red:      Ace of Hearts
  Red:      Ten of Diamonds     Black:    Eight of Clubs
Black:    Queen of Spades       Black:      Six of Spades
  Red:    Seven of Hearts       Black:      Ace of Spades
Black:    Seven of Spades         Red:     King of Hearts
Black:      Ten of Spades       Black:     Four of Spades
  Red:    Queen of Diamonds     Black:     Jack of Spades
Black:      Ace of Clubs        Black:     King of Spades
Black:    Eight of Spades       Black:    Queen of Clubs
  Red:    Eight of Hearts         Red:      Ace of Diamonds
  Red:     Four of Diamonds       Red:    Three of Hearts
  Red:     Five of Diamonds       Red:    Deuce of Diamonds
```

16.9 Write a program that right shifts an integer variable 4 bits. The program should print the integer in bits before and after the shift operation. Does your system place zeros or ones in the vacated bits?

```
// Exercise 16.9 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;

void displayBits( unsigned );

int main()
{
   unsigned val;

   cout << "Enter an integer: ";
   cin >> val;

   cout << "Before right shifting 4 bits is:\n";
   displayBits( val );
   cout << "After right shifting 4 bits is:\n";
   displayBits( val >> 4 );

   return 0;
}

void displayBits( unsigned value )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1 << SHIFT;
   cout << setw( 7 ) << value << " = ";

   for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
      cout << ( value & MASK ? '1' : '0' );
      value <<= 1;

      if ( c % 8 == 0 )
      cout << ' ';
   }

   cout << endl;
}
```

```
Enter an integer: 888
Before right shifting 4 bits is:
    888 = 00000000 00000000 00000011 01111000
After right shifting 4 bits is:
     55 = 00000000 00000000 00000000 00110111
```

16.10 If your computer uses 4-byte integers, modify the program of Fig. 16.5 so that it works with 4-byte integers.

16.11 Left shifting an **`unsigned`** integer by 1 bit is equivalent to multiplying the value by 2. Write function **`power2`** that takes two integer arguments **`number`** and **`pow`** and calculates

$$\texttt{number * 2}^{\texttt{pow}}$$

Use a shift operator to calculate the result. The program should print the values as integers and as bits.

```
// Exercise 16.11 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;
using std::setprecision;

void displayBits( unsigned );
unsigned power2( unsigned, unsigned );

int main()
{
   unsigned number, pow, result;

   cout << "Enter two integers: ";
   cin >> number >> pow;

   cout << "number:\n";
   displayBits( number );
   cout << "\npower:\n";
   displayBits( pow );
   result = power2( number, pow );
   cout << '\n' << number << " * 2^" << pow << " = " << result << '\n';
   displayBits( result );

   return 0;
}

unsigned power2( unsigned n, unsigned p )  { return n << p; }

void displayBits( unsigned value )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1 << SHIFT;
   cout << setw( 7 ) << value << " = ";

   for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
      cout << ( value & MASK ? '1' : '0' );
      value <<= 1;

      if ( c % 8 == 0 )
      cout << ' ';
   }

   cout << endl;
}
```

```
Enter an integer: 888
Before right shifting 4 bits is:
    888 = 00000000 00000000 00000011 01111000
After right shifting 4 bits is:
     55 = 00000000 00000000 00000000 00110111
```

```
Enter two integers: 3 4
number:
      3 = 00000000 00000000 00000000 00000011

power:
      4 = 00000000 00000000 00000000 00000100

3 * 2^4 = 48
     48 = 00000000 00000000 00000000 00110000
```

16.12 The left-shift operator can be used to pack two character values into a 2-byte unsigned integer variable. Write a program that inputs two characters from the keyboard and passes them to function **packCharacters**. To pack two characters into an **unsigned** integer variable, assign the first character to the **unsigned** variable, shift the **unsigned** variable left by 8 bit positions and combine the **unsigned** variable with the second character using the bitwise inclusive-OR operator. The program should output the characters in their bit format before and after they are packed into the **unsigned** integer to prove that the characters are in fact packed correctly in the **unsigned** variable.

```cpp
// Exercise 16.12 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;

unsigned packCharacters( char, char );
void displayBits( unsigned );

int main()
{
   char a, b;
   unsigned result;

   cout << "Enter two characters: ";
   cin.setf( ios::skipws );
   cin >> a >> b;

   cout << '\'' << a << '\'' << " in bits as an unsigned integers is:\n";
   displayBits( a );

   cout << '\'' << b << '\'' << " in bits as an unsigned integers is:\n";
   displayBits( b );

   result = packCharacters( a, b );

   cout << "\n\'" << a << '\'' << " and " << '\'' << b << '\''
        << " packed in an unsigned integer:\n";
   displayBits( result );

   return 0;
}

unsigned packCharacters( char x, char y )
{
   unsigned pack = x;

   pack <<= 8;
   pack |= y;
   return pack;
}

void displayBits( unsigned value )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1 << SHIFT;
   cout << setw( 7 ) << value << " = ";

   for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
      cout << ( value & MASK ? '1' : '0' );
      value <<= 1;

      if ( c % 8 == 0 )
      cout << ' ';
   }

   cout << endl;
}
```

```
Enter two characters: T N
'T' in bits as an unsigned integers is:
     84 = 00000000 00000000 00000000 01010100
'N' in bits as an unsigned integers is:
     78 = 00000000 00000000 00000000 01001110

'T' and 'N' packed in an unsigned integer:
  21582 = 00000000 00000000 01010100 01001110
```

16.13 Using the right-shift operator, the bitwise AND operator and a mask, write function **unpackCharacters** that takes the **unsigned** integer from Exercise 16.12 and unpacks it into two characters. To unpack two characters from an **unsigned** 2-byte integer, combine the unsigned integer with the mask **65280** (**1111111100000000**) and right shift the result 8 bits. Assign the resulting value to a **char** variable. Then, combine the **unsigned** integer with the mask **255** (**00000000 11111111**). Assign the result to another **char** variable. The program should print the **unsigned** integer in bits before it is unpacked, and then print the characters in bits to confirm that they were unpacked correctly.

```
// Exercise 16.13 Solution
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::setw;

void unpackCharacters( char * const, char * const, unsigned );
void displayBits( unsigned );
```

```cpp

int main()
{
   char a, b;
   unsigned packed = 16706;

   cout << "The packed character representation is:\n";
   displayBits( packed );
   unpackCharacters( &a, &b, packed );
   cout << "\nThe unpacked characters are \'" << a << "\' and \'" << b
        << "\'\n";
   displayBits( a );
   displayBits( b );

   return 0;
}

void unpackCharacters( char * const aPtr, char * const bPtr, unsigned pack )
{
   unsigned mask1 = 65280, mask2 = 255;

   *aPtr = static_cast< char >( ( pack & mask1 ) >> 8 );
   *bPtr = static_cast< char >( pack & mask2 );
}

void displayBits( unsigned value )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1 << SHIFT;
   cout << setw( 7 ) << value << " = ";

   for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
      cout << ( value & MASK ? '1' : '0' );
      value <<= 1;

      if ( c % 8 == 0 )
         cout << ' ';
   }

   cout << endl;
}
```

```
The packed character representation is:
  16706 = 00000000 00000000 01000001 01000010

The unpacked characters are 'A' and 'B'
     65 = 00000000 00000000 00000000 01000001
     66 = 00000000 00000000 00000000 01000010
```

16.14 If your system uses 4-byte integers, rewrite the program of Exercise 16.12 to pack 4 characters.

16.15 If your system uses 4-byte integers, rewrite the function `unpackCharacters` of Exercise 16.13 to unpack 4 characters. Create the masks you need to unpack the 4 characters by left shifting the value 255 in the mask variable by 8 bits 0, 1, 2 or 3 times (depending on the byte you are unpacking).

16.16 Write a program that reverses the order of the bits in an `unsigned` integer value. The program should input the value from the user and call function `reverseBits` to print the bits in reverse order. Print the value in bits both before and after the bits are reversed to confirm that the bits are reversed properly.

```
// Exercise 16.16 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;

void reverseBits( unsigned * const );
void displayBits( unsigned );

int main()
{
   unsigned a;

   cout << "Enter an unsigned integer: ";
   cin >> a;

   cout << "\nBefore bits are reversed:\n";
   displayBits( a );
   reverseBits( &a );
   cout << "\nAfter bits are reversed:\n";
   displayBits( a );

   return 0;
}

void reverseBits( unsigned * const vPtr )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1;
   unsigned value = *vPtr;

   for ( int i = 0; i <= 15; ++i ) {
      *vPtr <<= 1;
      *vPtr |= ( value & MASK );
      value >>= 1;
   }
}
void displayBits( unsigned value )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1 << SHIFT;
   cout << setw( 7 ) << value << " = ";

   for ( unsigned c = 1; c <= SHIFT + 1; c++ ) {
      cout << ( value & MASK ? '1' : '0' );
      value <<= 1;

      if ( c % 8 == 0 )
         cout << ' ';
   }

   cout << endl;
}
```

```
Enter an unsigned integer: 330

Before bits are reversed:
    330 = 00000000 00000000 00000001 01001010

After bits are reversed:
21648000 = 00000001 01001010 01010010 10000000
```

16.17 Write a program that demonstrates passing an array by value. (Hint: use a `struct`). Prove that a copy was passed by modifying the array copy in the called function.

16.18 Write a program that inputs a character from the keyboard and tests the character with each of the functions in the character handling library. The program should print the value returned by each function.

```
// Exercise 16.18 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cctype>

int main()
{
   char c;

   cout << "Enter a character: ";
   c = static_cast< char >( cin.get() );

   cout << "isdigit(\'" << c << "\')  = " << isdigit( c )
        << "\nisalpha(\'" << c << "\')  = " << isalpha( c )
        << "\nisalnum(\'" << c << "\')  = " << isalnum( c );

   cout << "\nisxdigit(\'" << c << "\') = " << isxdigit( c )
        << "\nislower(\'" << c << "\')  = " << islower( c )
        << "\nisupper(\'" << c << "\')  = " << isupper( c )
        << "\ntolower(\'" << c << "\')  = " << tolower( c )
        << "\ntoupper(\'" << c << "\')  = " << toupper( c )
        << "\nisspace(\'" << c << "\')  = " << isspace( c );

   cout << "\niscntrl(\'" << c << "\')  = " << iscntrl( c )
        << "\nispunct(\'" << c << "\')  = " << ispunct( c )
        << "\nisprint(\'" << c << "\')  = " << isprint( c )
        << "\nisgraph(\'" << c << "\')  = " << isgraph( c ) << endl;

   return 0;
}
```

```
Enter a character: ~
isdigit('~')  = 0
isalpha('~')  = 0
isalnum('~')  = 0
isxdigit('~') = 0
islower('~')  = 0
isupper('~')  = 0
tolower('~')  = 126
toupper('~')  = 126
isspace('~')  = 0
iscntrl('~')  = 0
ispunct('~')  = 16
isprint('~')  = 16
isgraph('~')  = 16
```

16.19 The following program uses function `multiple` to determine if the integer entered from the keyboard is a multiple of some integer `X`. Examine function `multiple`, and then determine the value of `X`.

```cpp
// ex16_19.cpp
// This program determines if a value is a multiple of X
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

bool multiple( int );

int main()
{
   int y;

   cout << "Enter an integer between 1 and 32000: ";
   cin >> y;

   if ( multiple( y ) )
      cout << y << " is a multiple of X" << endl;
   else
      cout << y << " is not a multiple of X" << endl;

   return 0;
}

bool multiple( int num )
{
   bool mult = true;

   for ( int i = 0, mask = 1; i < 10; i++, mask <<= 1 )
      if ( ( num & mask ) != 0 ) {
         mult = false;
         break;
      }

   return mult;
}
```

ANS: Determines if the number input is a multiple of X. The value of X is 1024.

16.20 What does the following program do?

```
// ex16_20.cpp
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int mystery( unsigned );

int main()
{
   unsigned x;

   cout << "Enter an integer: ";
   cin >> x;
   cout << "The result is " << mystery( x ) << endl;

   return 0;
}

int mystery( unsigned bits )
{
   const int SHIFT = 8 * sizeof( unsigned ) - 1;
   const unsigned MASK = 1 << SHIFT;
   unsigned total = 0;

   for ( int i = 0; i < SHIFT + 1; i++, bits <<= 1 )
      if ( ( bits & MASK ) == MASK )
         ++total;

   return !( total % 2 );
}
```

ANS: The program prints **`0`** if the total number of 1s in the bit representation is odd, and prints a **`1`** if the number of bits is even.

16.21 Write a program that inputs a line of text with **`istream`** member function **`getline`** (see Chapter 11) into character array **`s[ 100 ]`**. Output the line in uppercase letters and lowercase letters.

```
// Exercise 16.21 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <cctype>

const int SIZE = 100;

int main()
{
   char s[ SIZE ];
   int i;

   cout << "Enter a line of text:\n";
   cin.getline( s, SIZE );

   cout << "\nThe line in uppercase is:\n";

   for ( i = 0; s[ i ] != '\0'; ++i )
      cout.put( static_cast< char >( toupper( s[ i ] ) ) );

   cout << "\n\nThe line in lowercase is:\n";

   for ( i = 0; s[ i ] != '\0'; ++i )
      cout.put( static_cast< char >( tolower( s[ i ] ) ) );

   return 0;
}
```

```
Enter a line of text:
CPPHTP2 Instructor's Manual

The line in uppercase is:
CPPHTP2 INSTRUCTOR'S MANUAL

The line in lowercase is:
cpphtp2 instructor's manual
```

16.22 Write a program that inputs four strings that represent integers, converts the strings to integers sums the values, and prints the total of the four values.

```
// Exercise 16.22 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <cstdlib>

const int SIZE = 6;

int main()
{
   char stringValue[ SIZE ];
   int sum = 0;

   for ( int i = 1; i <= 4; ++i ) {
      cout << "Enter an integer string: ";
      cin >> stringValue;
      sum += atoi( stringValue );
   }

   cout << "The total of the values is " << sum << endl;

   return 0;
}
```

```
Enter an integer string: 11
Enter an integer string: 22
Enter an integer string: 44
Enter an integer string: 88
The total of the values is 165
```

16.23 Write a program that inputs four strings that represent floating-point values, converts the strings to double values, sums the values and prints the total of the four values.

```
// Exercise 16.23 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;

#include <cstdlib>

const int SIZE = 15;

int main()
{
   char stringValue[ SIZE ];
   double sum = 0.0;

   for ( int i = 1; i <= 4; ++i ) {
      cout << "Enter a double point string: ";
      cin >> stringValue;
      sum += atof( stringValue );
   }

   cout.setf( ios::fixed | ios::showpoint );
   cout << "\nThe total of the values is " << setprecision( 3 ) << sum
        << endl;

   return 0;
}
```

```
Enter a double point string: 1.1
Enter a double point string: 1.2
Enter a double point string: 1.3
Enter a double point string: 1.4

The total of the values is 5.000
```

16.24 Write a program that inputs a line of text and a search string from the keyboard. Using function `strstr`, locate the first occurrence of the search string in the line of text, and assign the location to variable `searchPtr` of type `char *`. If the search string is found, print the remainder of the line of text beginning with the search string. Then, use `strstr` again to locate the next occurrence of the search string in the line of text. If a second occurrence is found, print the remainder of the line of text beginning with the second occurrence. (Hint: The second call to `strstr` should contain the expression `searchPtr + 1` as its first argument.)

```
// Exercise 16.24 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <cstring>
```

```

const int SIZE1 = 80, SIZE2 = 15;

int main()
{
   char text[ SIZE1 ], search[ SIZE2 ], *searchPtr;

   cout << "Enter a line of text:\n";
   cin.get( text, SIZE1 );
   cout << "Enter a search string: ";
   cin >> search;
   searchPtr = strstr( text, search );

   if ( searchPtr ) {
      cout << "\nThe remainder of the line beginning with\n"
           << "the first occurrence of\n\"" << search << "\":\n"
           << searchPtr << '\n';

      searchPtr = strstr( searchPtr + 1, search );

      if ( searchPtr )
         cout << "\nThe remainder of the line beginning with"
              << "\nthe second occurrence of\n\"" << search << "\":\n"
              << searchPtr << '\n';
      else
         cout << "The search string appeared only once.\n";
   }
   else
      cout << "\"" << search << "\" not found.\n";

   return 0;
}
```

```
Enter a line of text:
alphabet soup tastes good
Enter a search string: be

The remainder of the line beginning with
the first occurrence of
"be":
bet soup tastes good
The search string appeared only once.
```

16.25 Write a program based on the program of Exercise 16.24that inputs several lines of text and a search string, and uses function **strstr** to determine the total number of occurrences of the string in the lines of text. Print the result.

```
// Exercise 16.25 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;
using std::resetiosflags;
```

```

#include <cstring>
#include <cctype>

const int SIZE1 = 80, SIZE2 = 20;

int main()
{
   char text[ 3 ][ SIZE1 ], search[ SIZE2 ], *searchPtr;
   int count = 0, i;

   cout << "Enter three lines of text:\n";

   for ( i = 0; i <= 2; ++i )
      cin.getline( &text[ i ][ 0 ], SIZE1 );

   // make all characters lowercase
   for ( i = 0; i <= 2; ++i )
      for ( int j = 0; text[ i ][ j ] != '\0'; ++j ) {
         char c = static_cast< char >( tolower( text[ i ][ j ] ) );
         text[ i ][ j ] = c;
      }

   cout << "\nEnter a search string: ";
   cin >> search;

   for ( i = 0; i <= 2; ++i ) {
      searchPtr = &text[ i ][ 0 ];

      while ( searchPtr = strstr( searchPtr, search ) ) {
         ++count;
         ++searchPtr;
      }
   }

   cout << "\nThe total occurrences of \"" << search
        << "\" in the text is:" << setw( 3 ) << count << endl;

   return 0;
}
```

```
Enter three lines of text:
first line of text
second line of text
third line of text

Enter a search string: in

The total occurrences of "in" in the text is:  3
```

16.26 Write a program that inputs several lines of text and a search character and uses function **strchr** to determine the total number of occurrences of the character in the lines of text.

```
// Exercise 16.26 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
```

```

#include <iomanip>

using std::setw;

#include <cstring>
#include <cctype>

const int SIZE = 80;

int main()
{
   char text[ 3 ][ SIZE ], search, *searchPtr;
   int count = 0, i;

   cout << "Enter three lines of text:\n";

   for ( i = 0; i <= 2; ++i )
      cin.getline( &text[ i ][ 0 ], SIZE );

   // convert all letters to lowercase
   for ( i = 0; i <= 2; ++i )
      for ( int j = 0; text[ i ][ j ] != '\0'; ++j ) {
         char c = static_cast< char >( tolower( text[ i ][ j ] ) );
         text[ i ][ j ] = c;
      }

   cout << "\nEnter a search character: ";
   cin >> search;

   for ( i = 0; i <= 2; ++i ) {
      searchPtr = &text[ i ][ 0 ];

      while ( searchPtr = strchr( searchPtr, search ) ) {
         ++count;
         ++searchPtr;
      }
   }

   cout << "The total occurrences of \'" << search << "\' in the text is:"
        << setw( 3 ) << count << endl;

   return 0;
}
```

```
Enter three lines of text:
one line of text
two lines of text
three lines of text

Enter a search character: e
The total occurrences of 'e' in the text is:  9
```

16.27 Write a program based on the program of Exercise 16.26 that inputs several lines of text and uses function **strchr** to determine the total number of occurrences of each letter of the alphabet in the text. Uppercase and lowercase letters should be counted together. Store the totals for each letter in an array, and print the values in tabular format after the totals have been determined.

```
// Exercise 16.27 Solution
#include <iostream>
```

```

using std::cout;
using std::endl;
using std::cin;

#include <iomanip>

using std::setw;

#include <cstring>
#include <cctype>

const int SIZE1 = 80, SIZE2 = 26;

int main()
{
   char text[ 3 ][ SIZE1 ], *searchPtr;
   int characters[ SIZE2 ] = { 0 }, count = 0;

   cout << "Enter three lines of text:\n";

   for ( int i = 0; i <= 2; ++i )
      cin.getline( &text[ i ][ 0 ], SIZE1 );

   // convert letters to lowercase
   for ( int k = 0; k <= 2; ++k )
      for ( int j = 0; text[ k ][ j ] != '\0'; ++j ) {
         char c = static_cast< char >( tolower( text[ k ][ j ] ) );
         text[ k ][ j ] = c;
      }

   for ( int q = 0; q < SIZE2; ++q ) {
      count = 0;

      for ( int j = 0; j <= 2; ++j ) {
         searchPtr = &text[ j ][ 0 ];

         while ( searchPtr = strchr( searchPtr, 'a' + q ) ) {
            ++count;
            ++searchPtr;
         }
      }

      characters[ q ] = count;
   }

   cout << "\nThe total occurrences of each character:\n";

   for ( int w = 0; w < SIZE2; ++w )
      cout << setw( 3 ) << static_cast< char >( 'a' + w ) << ':' << setw( 3 )
           << characters[ w ] << '\n';

   return 0;
}
```

```
Enter three lines of text:
The yak ran away
A giant parrot flew by
The cat pounced on the rat

The total occurrences of each character:
  a:  9
  b:  1
  c:  2
  d:  1
  e:  5
  f:  1
  g:  1
  h:  3
  i:  1
  j:  0
  k:  1
  l:  1
  m:  0
  n:  4
  o:  3
  p:  2
  q:  0
  r:  4
  s:  0
  t:  7
  u:  1
  v:  0
  w:  2
  x:  0
  y:  3
  z:  0
```

16.28 The chart in Appendix B shows the numeric code representations for the characters in the ASCII character set. Study this chart, and then state whether each of the following is *true* or *false*:

a) The letter "**A**" comes before the letter "**B**".
b) The digit "**9**" comes before the digit "**0**".
c) The commonly used symbols for addition, subtraction, multiplication and division all come before any of the digits.
d) The digits come before the letters.
e) If a sort program sorts strings into ascending sequence, then the program will place the symbol for a right parenthesis before the symbol for a left parenthesis.

```
// Exercise 16.28 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;


const int SIZE = 20;

int main()
{
   char array[ 5 ][ SIZE ];
   int i;

   for ( i = 0; i <= 4; ++i ) {
      cout << "Enter a string: ";
      cin.getline( &array[ i ][ 0 ], SIZE );
   }

   cout << "The strings starting with 'b' are:\n";

   for ( i = 0; i <= 4; ++i )
      if ( array[ i ][ 0 ] == 'b' )
         cout << &array[ i ][ 0 ] << '\n';

   return 0;
}
```

```
Enter a string: c++
Enter a string: apple
Enter a string: burger
Enter a string: band
Enter a string: bag
The strings starting with 'b' are:
burger
band
bag
```

16.29 Write a program that reads a series of strings and prints only those strings beginning with the letter "**b**".

```
// Exercise 16.29 Solution
#include <iostream>

using std::cout;
using std::cin;


const int SIZE = 20;

int main()
{
   char array[ 5 ][ SIZE ];
   int i;

   for ( i = 0; i <= 4; ++i ) {
      cout << "Enter a string: ";
      cin.getline( &array[ i ][ 0 ], SIZE );
   }

   cout << "The strings starting with 'b' are:\n";

   for ( i = 0; i <= 4; ++i )
      if ( array[ i ][ 0 ] == 'b' )
         cout << &array[ i ][ 0 ] << '\n';

   return 0;
}
```

```
Enter a string: MOVED
Enter a string: SAW
Enter a string: RAN
Enter a string: CARVED
Enter a string: PROVED

The strings ending with "ED" are:
MOVED
CARVED
PROVED
```

16.30 Write a program that reads a series of strings and prints only those strings that end with the letters “ **ED**”Write a program that inputs an ASCII code and prints the corresponding character. Modify this program so that it generates all possible three-di git codes in the range 000 to 255 and attempts to print the corresponding characters. What happens when this program is runUsing the ASCII character chart in Appendix B as a guide, write your own versions of the character handling functions in Fig. 16.16.

```
// Exercise 16.30 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <cstring>

const int SIZE = 20;

int main()
{
   int length, i;
   char array[ 5 ][ SIZE ];

   for ( i = 0; i <= 4; ++i ) {
      cout << "Enter a string: ";
      cin.getline( &array[ i ][ 0 ], SIZE );
   }

   cout << "\nThe strings ending with \"ED\" are:\n";

   for ( i = 0; i <= 4; ++i ) {
      length = strlen( &array[ i ][ 0 ] );

      if ( strcmp( &array[ i ][ length - 2 ], "ED" ) == 0 )
         cout << &array[ i ][ 0 ] << '\n';
   }

   return 0;
}
```

```
Enter an ASCII character code (EOF to end): 44
The corresponding character is ','

Enter an ASCII character code (EOF to end): 77
The corresponding character is 'M'

Enter an ASCII character code (EOF to end): 26
The corresponding character is '

...
```

16.31 Write your own versions of the functions in Fig. 16.20 for converting strings to numbers.

```cpp
// Exercise 16.31 Solution
// NOTE: This solution is easily modified to print all
// possible three digit codes.
#include <iostream>

using std::cout;
using std::cin;


int main()
{
   int c;

   cout << "Enter an ASCII character code (EOF to end): ";
   cin >> c;

   while ( c != EOF ) {
      if ( c >= 0 && c <= 255 )
         cout << "The corresponding character is '"
              << static_cast< char > ( c ) << "\'\n";
      else
         cout << "Invalid character code\n";

      cout << "\nEnter an ASCII character code (EOF to end): ";
      cin >> c;
   }

   return 0;
}
```

```
Enter an ASCII character code (EOF to end): 44
The corresponding character is ','

Enter an ASCII character code (EOF to end): 77
The corresponding character is 'M'

Enter an ASCII character code (EOF to end): 26
The corresponding character is '

...
```

16.32 Write your own versions of the functions in Fig. 16.27 for searching strings.

```cpp
// Exercise 16.32 Solution
#include <iostream>

using std::cout;
using std::cin;

int isDigit( int );
int isAlpha( int );
int isAlNum( int );
int isLower( int );
int isUpper( int );
int isSpace( int );
int isPunct( int );
int isPrint( int );
int isGraph( int );
int toLower( int );
int toUpper( int );

int main()
{
   int v;
   char a, header[] = "According to",
           *names[] = { "isDigit ", "isAlpha ", "isAlNum ",
                        "isLower ", "isUpper ", "isSpace ",
                        "isPunct ", "isPrint ", "isGraph ",
                        "toLower ", "toUpper " },
           *names2[] = { "digit", "letter", "letter/digit", "lowercase",
                         "uppercase", "space", "punctuation", "print", "graph",
                         "converted lowercase", "converted uppercase" };
   int ( *f[] )( int ) = { isDigit, isAlpha, isAlNum, isLower,
                           isUpper, isSpace, isPunct, isPrint,
                           isGraph, toLower, toUpper };

   cout << "Enter a character: ";
   cin >> a;

   for ( int k = 0; k < 11; ++k ) {
      v = ( *f[ k ] )( static_cast< int >( a ) );
      cout.write( header, 13 );
      cout << names[ k ] << a << ( !v ? " is not a " : " is a " )
           << names2[ k ] << " character\n";
   }

   return 0;
}

// Normally these would return bool, but int return type
// is consistent with the ctype library.
int isDigit( int c )
{
   return ( c >= 48 && c <= 57 ) ? 1 : 0;
}

int isAlpha( int c )
{
   return ( ( c >= 65 && c <= 90 ) || ( c >= 97 && c <= 122 ) ) ? 1 : 0;
}

int isAlNum( int c )
{
   return ( isDigit( c ) == 1 || isAlpha( c ) == 1 ) ? 1 : 0;
}

int isLower( int c )
{
   return ( c >= 97 && c <= 122 ) ? 1 : 0;
}

int isUpper( int c )
{
   return ( c >= 65 && c <= 90 ) ? 1 : 0;
}

int isSpace( int c )
{
   return ( ( c == 32 ) || ( c >= 9 && c <= 13 ) ) ? 1 : 0;
}

int isPunct( int c )
{
   return ( isAlNum( c ) == 0 && isSpace( c ) == 0 ) ? 1 : 0;
}

int isPrint( int c )
{
   return ( c >= 32 && c <= 126 ) ? 1 : 0;
}

int isGraph( int c )
{
   return ( c >= 33 && c <= 126 ) ? 1 : 0;
}

int toLower( int c )
{
   return ( isUpper( c ) == 1 ) ? c + 32 : c;
}

int toUpper( int c )
{
   return ( isLower( c ) == 1 ) ? c - 32 : c;
}
```

```
Enter a character: U
According to isDigit U is not a digit character
According to isAlpha U is a letter character
According to isAlNum U is a letter/digit character
According to isLower U is not a lowercase character
According to isUpper U is a uppercase character
According to isSpace U is not a space character
According to isPunct U is not a punctuation character
According to isPrint U is a print character
According to isGraph U is a graph character
According to toLower U is a converted lowercase character
According to toUpper U is a converted uppercase character
```

16.33 Write your own versions of the functions in Fig. 16.34 for manipulating blocks of memory.

16.34 *(Project: A Spelling Checker)* Many popular word-processing software packages have built-in spell checkers. We used spell-checking capabilities in preparing this book and discovered that no matter how careful we thought we were in writing a chapter, the software was always able to find a few more spelling errors than we were able to catch manually.

In this project, you are asked to develop your own spell-checker utility. We make suggestions to help get you started. You should then consider adding more capabilities. You may find it helpful to use a computerized dictionary as a source of words.

Why do we type so many words with incorrect spellings? In some cases, it is because we simply do not know the correct spelling, so we make a "best guess." In some cases, it is because we transpose two letters (e.g., "defualt" instead of "default"). Sometimes we double type a letter accidentally (e.g., "hanndy" instead of "handy"). Sometimes we type a nearby key instead of the one we intended (e.g., "biryhday" instead of "birthday"). And so on.

Design and implement a spell-checker program. Your program maintains an array **`wordList`** of character strings. You can either enter these strings or obtain them from a computerized dictionary.

Your program asks a user to enter a word. The program then looks up that word in the **`wordList`** array. If the word is present in the array, your program should print "**`Word is spelled correctly.`**"

If the word is not present in the array, your program should print " **`Word is not spelled correctly.`**" Then your program should try to locate other words in **`wordList`** that might be the word the user intended to type. For example, you can try all possible single transpositions of adjacent letters to discover that the word "default" is a direct match to a word in **`wordList`**.

Of course, this implies that your program will check all other single transpositions, such as “edfault,” “dfeault,” “deafult,” “defalut” and “defautl.” When you find a new word that matches one in **`wordList`**, print that word in a message such as “ **`Did you mean "default?"`**.”

Implement other tests such as replacing each double letter with a single letter and any other tests you can develop to improve the value of your spell checker.

17

The Preprocessor Solutions

Solutions

17.4 Write a program that defines a macro with one argument to compute the volume of a sphere. The program should compute the volume for spheres of radius 1 to 10, and print the results in tabular format. The formula for the volume of a sphere is

$$(4.0 / 3) * \pi * r^3$$

where π is **3.14159**.

```
// Exercise 17.4 Solution
#include <iostream>

using std::cout;
using std::ios;


#include <iomanip>

using std::setw;
using std::setprecision;

#define PI 3.14159

// define macro for sphere volume
#define SPHEREVOLUME( r ) ( 4.0 / 3.0 * PI * ( r ) * ( r ) * ( r ) )

int main()
{
   // print header
   cout << setw( 10 ) << "Radius" << setw( 10 ) << "Volume\n";

   cout.setf( ios::fixed | ios::showpoint );
   for ( int i = 1; i <= 10; ++i )
      cout << setw( 10 ) << i << setw( 10 ) << setprecision( 3 )
           << SPHEREVOLUME( i ) << '\n';

   return 0;
}
```

```
   Radius   Volume
        1    4.189
        2   33.510
        3  113.097
        4  268.082
        5  523.598
        6  904.778
        7 1436.754
        8 2144.659
        9 3053.625
       10 4188.787
```

17.5 Write a program that produces the following output:

```
The sum of x and y is 13
```

The program should define macro **SUM** with two arguments, **x** and **y**, and use **SUM** to produce the output.

```
// Exercise 17.5 Solution
#include <iostream>

using std::cout;
using std::endl;

#define SUM( x, y ) ( ( x ) + ( y ) )

int main()
{
   cout << "The sum of 6 and 7 is " << SUM( 6, 7 ) << endl;
   return 0;
}
```

```
The sum of 6 and 7 is 13
```

17.6 Write a program that uses macro **MINIMUM2** to determine the smallest of two numeric values. Input the values from the keyboard.

```
// Exercise 17.6 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;

#define MINIMUM2( X, Y ) ( ( X ) < ( Y ) ? ( X ) : ( Y ) )
```

```

int main()
{
   int a, b;
   double c, d;

   cout << "Enter two integers: ";
   cin >> a >> b;
   cout << "The minimum of " << a << " and " << b << " is " << MINIMUM2( a, b )
        << "\n\n";

   cout << "Enter two doubles: ";
   cin >> c >> d;

   cout.setf( ios::fixed | ios::showpoint );
   cout << "The minimum of " << setprecision( 2 ) << c << " and " << d
        << " is " << MINIMUM2( c, d ) << '\n';

   return 0;
}
```

```
Enter two integers: 8 22
The minimum of 8 and 22 is 8

Enter two doubles: 73.46 22.22
The minimum of 73.46 and 22.22 is 22.22
```

17.7 Write a program that uses macro **MINIMUM3** to determine the smallest of three numeric values. Macro**MINIMUM3** should use macro **MINIMUM2** defined in Exercise 17.6 to determine the smallest number. Input the values from the keyboard.

```
// Exercise 17.7 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;

#define MINIMUM2( X, Y ) ( ( X ) < ( Y ) ? ( X ) : ( Y ) )
#define MINIMUM3( U, V, W ) ( MINIMUM2( W, MINIMUM2( U, V ) ) )

int main()
{
   int a, b, c;
   double d, e, f;

   cout << "Enter three integers: ";
   cin >> a >> b >> c;
   cout << "The minimum of " << a << ", " << b << ", and " << c
        << " is " << MINIMUM3( a, b, c ) << "\n\nEnter three doubles: ";

   cin >> d >> e >> f;
   cout.setf( ios::fixed | ios::showpoint );
   cout << "The minimum of " << setprecision( 2 ) << d << ", "
```

```
          << e << ", and " << f << " is " << MINIMUM3( d, e, f ) << '\n';

   return 0;
}
```

```
Enter three integers: 44 2 55
The minimum of 44, 2, and 55 is 2

Enter three doubles: 9.5 7.3 3.22
The minimum of 9.50, 7.30, and 3.22 is 3.22
```

17.8 Write a program that uses macro **PRINT** to print a string value.

```
// Exercise 17.8 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#define PRINT( s ) cout << ( s )
#define SIZE 20

int main()
{
   char text[ SIZE ];

   PRINT( "Enter a string: " );
   cin >> text;

   PRINT( "The string entered was: " );
   PRINT( text );
   PRINT( endl );

   return 0;
}
```

```
Enter a string: HELLO
The string entered was: HELLO
```

17.9 Write a program that uses macro **PRINTARRAY** to print an array of integers. The macro should receive the array and the number of elements in the array as arguments.

```
// Exercise 17.9 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;

#define PRINTARRAY( A, N )  for ( int i = 0; i < ( N ); ++i ) \
                                cout << setw( 3 ) << A[ i ]
```

```

#define SIZE 10

int main()
{
   int b[ SIZE ] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };

   cout << "The array values are:\n";
   PRINTARRAY( b, SIZE );
   cout << endl;
   return 0;
}
```

```
The array values are:
 2  4  6  8 10 12 14 16 18 20
```

17.10 Write a program that uses macro **SUMARRAY** to sum the values in a numeric array. The macro should receive the array and the number of elements in the array as arguments.

```
// Exercise 17.10 Solution
#include <iostream>

using std::cout;
using std::endl;

#define SUMMARRAY( A, S )  for ( int c = 0; c < S; ++c )   \
                              sum += A[ c ];
#define SIZE 10

int main()
{
   int array[ SIZE ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }, sum = 0;

   SUMMARRAY( array, SIZE );
   cout << "Sum is " << sum << endl;
   return 0;
}
```

```
Sum is 55
```

17.11 Rewrite the solutions to Exercises 17.4 to 17.10 as **inline** functions.

```
// Exercise 17.11 Solution
// NOTE: Exercises 17.9 and 17.10 cannot be
// expanded inline because they require the
// use of for repitition structures
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;

```

```
#define PI 3.14159

inline double sphereVolume( int r )
   { return 4.0 / 3.0 * PI * ( r ) * ( r ) * ( r ); }
inline int sum( int x, int y ) { return x + y; }
inline int minimum2( int x, int y ) { return x < y ? x : y; }
inline int minimum3( int x, int y, int z )
   { return minimum2( z, minimum2( x, y ) ); }
inline void print( char const * const cPtr ) { cout << cPtr; }

int main()
{
   // print header
   cout << "Function sphereVolume as an inline function:\n"
        << setw( 10 ) << "Radius" << setw( 10 ) << "Volume\n";

   cout.setf( ios::fixed | ios::showpoint );
   for ( int i = 1; i <= 10; ++i )
      cout << setw( 10 ) << i << setw( 10 ) << setprecision( 3 )
           << sphereVolume( i ) << '\n';

   int x = 6, y = 7;
   cout << "\nFunction sum as an inline function:\n"
        << "The sum of " << x << " and " << y << " is "
        << sum( 6, 7 ) << '\n';

   cout << "\nFunction minimum2 as an inline function:\n"
        << "The minimum of " << x << " and " << y << " is "
        << minimum2( x, y ) << '\n';

   int z = 4;
   cout << "\nFunction minimum3 as an inline function:\n"
        << "The minimum of " << x << ", " << y << " and " << z << " is "
        << minimum3( x, y, z ) << '\n';

   char s[] = "string...";
   cout << "\nFunction print as an inline function:\n"
        << "The output of print is: ";
   print( s );
   cout << endl;

   return 0;
}
```

```
 Function sphereVolume as an inline function:
    Radius   Volume
         1     4.189
         2    33.510
         3   113.097
         4   268.082
         5   523.598
         6   904.778
         7  1436.754
         8  2144.659
         9  3053.625
        10  4188.787

Function sum as an inline function:
The sum of 6 and 7 is 13

Function minimum2 as an inline function:
The minimum of 6 and 7 is 6

Function minimum3 as an inline function:
The minimum of 6, 7 and 4 is 4

Function print as an inline function:
The output of print is: string...
```

17.12 For each of the following macros, identify the possible problems (if any) when the preprocessor expands the macros:

a) **`#define SQR( x ) x * x`**

ANS: When **`x`** is an expression, such as **`z - y`**.

b) **`#define SQR( x ) ( x * x )`**

ANS: When **`x`** is an expression, such as **`z - y`**.

c) **`#define SQR( x ) ( x ) * ( x )`**

ANS: When **`SQR( x )`** is used in an expression such as **`12 / SQR( 2 )`**.

d) **`#define SQR( x ) ( ( x ) * ( x ) )`**

ANS: No problems.

18

C Legacy Code Topics Solutions

Solutions

18.2 Write a program that calculates the product of a series of integers that are passed to function **product** using a variable-length argument list. Test your function with several calls, each with a different number of arguments.

```
// Exercise 18.2 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <cstdarg>

int sum( int, ... );

int main()
{
   int a = 1, b = 2, c = 3, d = 4, e = 5;

   cout << "a = " << a << ", b = " << b << ", c = " << c << ", d = " << d
        << ", e = " << e << "\n\n";

   cout << "The sum of a and b is: " << sum( 2, a, b )
        << "\nThe sum of a, b, and c is: " << sum( 3, a, b, c )
        << "\nThe sum of a, b, c, and d is: " << sum( 4, a, b, c, d )
        << "\nThe sum of a, b, c, d, and e is: " << sum( 5, a, b, c, d, e )
        << endl;

   return 0;
}

int sum( int i, ... )
{
   int total = 0;
   va_list ap;

   va_start( ap, i );

   // calculate total
   for ( int j = 1; j <= i; ++j )
      total += va_arg( ap, int );

   va_end( ap );
   return total;
}
```

```
a = 1, b = 2, c = 3, d = 4, e = 5

The sum of a and b is: 3
The sum of a, b, and c is: 6
The sum of a, b, c, and d is: 10
The sum of a, b, c, d, and e is: 15
```

18.3 Write a program that prints the command-line arguments of the program.

```
// Exercise 18.3 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;

using std::ios;

int main( int argc, char *argv[] )
{
   cout << "The command line arguments are:\n";

   for ( int i = 0; i < argc; ++i )
      cout << argv[ i ] << ' ';

   return 0;
}
```

```
c:\>p18_03.exe arg1 arg2 arg3
The command line arguments are:
p18_03.exe arg1 arg2 arg3
```

18.4 Write a program that sorts an integer array into ascending order or descending order. The program should use command-line arguments to pass either argument `-a` for ascending order or `-d` for descending order. (Note: This is the standard format for passing options to a program in UNIX.)

```
// Exercise 18.4 Solution
#include <iostream>

using std::cout;
using std::cin;

#include <iomanip>
using std::setw;

```

```
const int SIZE = 100;

void swap( int * const, int * const );

int main( int argc, char *argv[] )
{
   int a[ SIZE ];
   bool order;

   if ( argc != 2 )
      cout << "Usage: p18_4 -option\n";
   else {
      cout << "Enter up to " << SIZE << " integers (EOF to end input): ";

      for ( int count = 0; !( cin.eof() ) && count < SIZE; ++count )
         cin >> a[ count ];

      // count is incremented before for loop continuation fails
      --count;
      order = ( argv[ 1 ][ 1 ] == 'd' ) ? true : false;

      for ( int i = 1; i < count; ++i )
         for ( int j = 0; j < count - 1; ++j )
            if ( order ) {
               if ( a[ i ] > a[ j ] )
                  swap( &a[ i ], &a[ j ] );
            }
            else if ( a[ i ] < a[ j ] )
               swap( &a[ j ], &a[ i ] );

      cout << "\n\nThe sorted array is:\n";

      for ( int j = 0; j < count; ++j )
         cout << setw( 3 ) << a[ j ];

      cout << '\n';
   }

   return 0;
}

void swap( int * const xPtr, int * const yPtr )
{
   int temp;

   temp = *xPtr;
   *xPtr = *yPtr;
   *yPtr = temp;
}
```

```
c:\>p18_04.exe -d
Enter up to 100 integers (EOF to end input): 77 2 -8 9 44 8 76 41 99
The sorted array is:
99 77 76 44 41  9  8  2 -8
```

18.5 Read the manuals for your system to determine what signals are supported by the signal-handling library (**csignal**). Write a program with signal handlers for the signals **SIGABRT** and **SIGINT**. The program should test the trapping of these signals by calling function **abort** to generate a signal of type **SIGABRT** and by pressing *Ctrl c* to generate a signal of type **SIGINT**.

18.6 Write a program that dynamically allocates an array of integers. The size of the array should be input from the keyboard. The elements of the array should be assigned values input from the keyboard. Print the values of the array. Next, reallocate the memory for the array to half of the current number of elements. Print the values remaining in the array to confirm that they match the first half of the values in the original array.

```
// Exercise 18.6 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>
using std::setw;
using std::setprecision;
using std::setiosflags;

#include <cstdlib>

int main()
{
   int count, *array;

   cout << "This program dynamically allocates an array of integers."
        << "\nEnter the number of elements in the array: ";
   cin >> count;

   // allocate memory
   array = static_cast< int * >( calloc( count, sizeof( int ) ) );

   for ( int i = 0; i < count; ++i ) {
      cout << "Enter an integer: ";
      cin >> array[ i ];
   }

   cout << "\nThe elements of the array are:\n";

   for ( int j = 0; j < count; ++j )
      cout << setw( 3 ) << array[ j ];

   // reallocate to half the original size
   realloc( array, count / 2 * sizeof( int ) );
   cout << "\n\nThe elements of the array after reallocation are:\n";

   for ( int k = 0; k < count / 2; ++k )
      cout << setw( 3 ) << array[ k ];

   cout << '\n';

   return 0;
}
```

```
This program dynamically allocates an array of integers.
Enter the number of elements in the array: 5
Enter an integer: 1
Enter an integer: 2
Enter an integer: 3
Enter an integer: 4
Enter an integer: 5

The elements of the array are:
  1  2  3  4  5

The elements of the array after reallocation are:
  1  2
```

18.7 Write a program that takes two file names as command-line arguments, reads the characters from the first file one at a time and writes the characters in reverse order to the second file.

```
// Exercise 18.7 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;
using std::cerr;
using std::istream;
using std::ostream;

#include <fstream>

using std::ofstream;
using std::ifstream;

void reverseFile( istream&, ostream& );

int main( int argc, char *argv[] )
{
   ifstream inFile( argv[ 1 ], ios::in );
   ofstream outFile( argv[ 2 ], ios::out );

   if ( argc != 3 )
      cout << "Usage: copy infile outfile\n";
   else
      if ( inFile )
         if ( outFile )
            reverseFile( inFile, outFile );
         else
            cerr << "File \"" << argv[ 2 ] << "\" could not be opened\n";
      else
         cerr << "File \"" << argv[ 1 ] << "\" could not be opened\n";

   return 0;
}

void reverseFile( istream &in, ostream &out )
{
   int c;

   if ( ( c = in.get() ) != EOF )
      reverseFile( in, out );
   else
      return;  // do not write EOF character

   out.put( static_cast< char > ( c ) );
}
```

```
c:\>p18_07.exe test.txt copy.txt
```

18.8 Write a program that uses `goto` statements to simulate a nested looping structure that prints a square of asterisks as follows:

```
*****
*   *
*   *
*   *
*****
```

The program should use only the following three output statements:

```
cout << '*';
cout << ' ';
cout << endl;
```

```
// Exercise 18.8 Solution
#include <iostream>

using std::cout;
using std::cin;



int main()
{
   int size, row = 0, col;

   cout << "Enter the side length of the square: ";
   cin >> size;

   start:               // label
      ++row;
      cout << '\n';

      if ( row > size )
         goto end;

      col = 1;

      innerLoop:        // label
         if ( col > size )
            goto start;

         cout << ( row == 1 || row == size || col == 1 ||
                   col == size ? '*' : ' ' );

         ++col;
         goto innerLoop;

   end:                 // label

   return 0;
}
```

```
Enter the side length of the square: 5

*****
*   *
*   *
*   *
*****
```

18.9 Provide the definition for **union Data** containing **char c**, **short s**, **long l**, **float f** and **double d**.

ANS:

```
union Data {
   char c;
   short s;
   long l;
   float f;
   double d;
};
```

18.10 Create **union Integer** with members **char c**, **short s**, **int i** and **long l**. Write a program that inputs values of type **char**, **short**, **int** and **long** and stores the values in **union** variables of type **union Integer**. Each **union** variable should be printed as a **char**, a **short**, an **int** and a **long**. Do the values always print correctly?

```
// Exercise 18.10 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

union Integer {
   char c;
   short s;
   int i;
   long l;
};

void printUnion( Integer );

int main()
{
   Integer value;

   cout << "Enter a character: ";
   value.c = static_cast< char >( cin.get() );
   printUnion( value );

   cout << "Enter a short: ";
   cin >> value.s;
   printUnion( value );

   cout << "Enter an int: ";
   cin >> value.i;
   printUnion( value );

   cout << "Enter a long: ";
   cin >> value.l;
   printUnion( value );

   return 0;
}

void printUnion( Integer x )
{
    cout << "Current values in union Integer are:\n"
         << "char c  = " << x.c
         << "\nshort s = " << x.s
         << "\nint i   = " << x.i
         << "\nlong l  = " << x.l << "\n\n";
}
```

```
Enter a character: w
Current values in union Integer are:
char c  = w
short s = -13193
int i   = -858993545
long l  = -858993545

Enter a short: 5
Current values in union Integer are:
char c  = ?
short s = 5
int i   = -859045883
long l  = -859045883

Enter an int: 9999
Current values in union Integer are:
char c  = ¤
short s = 9999
int i   = 9999
long l  = 9999

Enter a long: 1000000
Current values in union Integer are:
char c  = @
short s = 16960
int i   = 1000000
long l  = 1000000
```

18.11 Create **union FloatingPoint** with members **float f**, **double d** and **long double l**. Write a program that inputs value of type **float**, **double** and **long double** and stores the values in **union** variables of type **union FloatingPoint**. Each **union** variable should be printed as a **float**, a **double** and a **long double**. Do the values always print correctly?

```
// Exercise 18.11 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

union FloatingPoint {
   float f;
   double d;
   long double l;
};

void printUnion( FloatingPoint );

int main()
{
   FloatingPoint value;

   cout << "Enter a float: ";
   cin >> value.f;
   printUnion( value );

   cout << "Enter a double: ";
   cin >> value.d;
   printUnion( value );

   cout << "Enter a long double: ";
   cin >> value.l;
   printUnion( value );

   return 0;
}

void printUnion( FloatingPoint x )
{
    cout << "Current values in union Integer are:\n"
         << "float f  = " << x.f
         << "\ndouble d = " << x.d
         << "\nlong double l  = " << x.l << "\n\n";
}
```

```
Enter a float: 4.567
Current values in union Integer are:
float f  = 4.567
double d = -9.25596e+061
long double l  = -9.25596e+061

Enter a double: 9998888.765
Current values in union Integer are:
float f  = 3.24255e-024
double d = 9.99889e+006
long double l  = 9.99889e+006

Enter a long double: 5e306
Current values in union Integer are:
float f  = 0.0210515
double d = 5e+306
long double l  = 5e+306
```

18.12 Given the **union**

```
union A {
   double y;
   char *z;
};
```

which of the following are correct statements for initializing the **union**?

a) **A p = B; // B is of same type as A**

ANS: Correct.

b) `A q = x;  // x is a double`
ANS: Incorrect.
c) `A r = 3.14159;`
ANS: Incorrect.
d) `A s = { 79.63 };`
ANS: Correct.
e) `A t = { "Hi There!" };`
ANS: Incorrect.
f) `A u = { 3.14159, "Pi" };`
ANS: Incorrect.

19

Class **`string`** and String Stream Processing Solutions

Solutions

19.4 Fill in the blanks in each of the following:

a) Functions ________, ________ and ________ convert **`string`**s to C-style strings.
ANS: **`data`**, **`c_str`**, **`copy`**

b) Function ________ is used for assignment.
ANS: **`assign`**

c) ________ is the return type of function **`rbegin`**.
ANS: **`string::reverse_iterator`**

d) Function ________ is used to retrieve a substring.
ANS: **`substr`**

19.5 State which of the following statements are true and which are false. If a statement is false, explain why.

a) **`string`**s are null terminated.
ANS: False. **`string`**s are not necessarily null terminated.

b) Function **`max_size`** returns the maximum size for a **`string`**.
ANS: True.

c) Function **`at`** is capable of throwing an **`out_of_range`** exception.
ANS: True.

d) Function **`begin`** returns an **`iterator`**.
ANS: True (**`string::iterator`** is more precise).

e) **`string`**s are passed by reference by default.
ANS: False. By default, **`string`**s are passed by value.

19.6 Find any error(s) in each of the following and explain how to correct it (them):

a)
```
std::cout << s.data() << std::endl;  // s is "hello"
```
ANS: The array returned by **`data`** is not null terminated.

b)
```
erase( s.rfind( "x" ), 1 );  // s is "xenon"
```
ANS: Function **`erase`** is a **`string`** class member function (i.e., **`erase`** must be called by an object of type **`string`**).

c)
```
string& foo( void )
{
   string s( "Hello" );
   ...   // other statements of function
   return;
}
```

ANS: A value is not being returned from the function (i.e., the `return` statement should be `return s;`). The `return` type should be `string` not `string&`.

19.7 (*Simple Encryption*) Some information on the Internet may be encrypted with a simple algorithm known as “rot13”—which rotates each character by 13 positions in the alphabet. Thus, `'a'` corresponds to `'n'`, and `'x'` corresponds to `'k'`. rot13 is an example of *symmetric key encryption*. With symmetric key encryption, both the encrypter and decrypter use the same key.

a) Write a program that encrypts a message using rot13.
b) Write a program that decrypts the scrambled message using 13 as the key.
c) After writing the programs of part (a) and part (b) briefly answer the following question: If you did not know the key for part (b), how difficult do you think it would be to break the code using any resources available? What if you had access to substantial computing power (e.g., Cray supercomputers)? In Exercise 19.27 we ask you to write a program to accomplish this.

```
// Execise 19.07 Part A Solution
// When solving Part B of this exercise, you
// might find it more convenient to only use
// uppercase letters for you input.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;
using std::getline;

int main()
{
   string m;
   int key = 13;  // Our key for encryption

   cout << "Enter a string:";
   getline( cin, m );

   string::iterator mi = m.begin();

   while ( mi != m.end() ) {
      *mi += key;
      ++mi;
   }

   cout << "\nEncypted string is:" << m << endl;

   return 0;
}
```

```
Enter a string:JAMES BOND IS 007
Encypted string is:WNZR`-O\[Q-V`-==D
```

```
// Execise 19.07 Part B Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

```

```
#include <string>
using std::string;
using std::getline;

int main()
{
   string m;
   int key = 13;  // Our key for decryption

   cout << "Enter encrypted string:";
   getline( cin, m );

   string::iterator mi = m.begin();

   while ( mi != m.end() ) {
      *mi -= key;
      ++mi;
   }

   cout << "\nDecypted string is:" << m << endl;

   return 0;
}
```

```
Enter encrypted string:WNZR`-O\[Q-V`-==D
Decypted string is:JAMES BOND IS 007
```

19.8 Write a program using iterators that demonstrates the use of functions **rbegin** and **rend**.

```
// Exercise 19.8 Solution
// Program demonstrates rend and rbegin.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

int main()
{
   string s( "abcdefghijklmnopqrstuvwxyz" );
   string::reverse_iterator re = s.rend(),
                            rb = s.rbegin();

   cout << "Using rend() string is: ";
   while ( re >= s.rbegin() ) {
      cout << *re;
      --re;
   }

   cout << "\nUsing rbegin() string is: ";
   while ( rb != s.rend() ) {
      cout << *rb;
      ++rb;
   }

   cout << endl;

   return 0;
}
```

```
Using rend() string is:  abcdefghijklmnopqrstuvwxyz
Using rbegin() string is: zyxwvutsrqponmlkjihgfedcba
```

19.9 Write your own versions of functions **data** and **c_str**.

19.10 Write a program that reads in several **string**s and prints only those ending in “**r**” or “**ay**”. Only lowercase letters should be considered.

```
// Exercise 19.10 Solution
// Program determines if string ends in 'r'
// or "ay".
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

int main()
{
   string s[ 5 ];

   for ( int i = 0; i < 5; ++i ) {
      cout << "Enter a word: ";
      cin >> s[ i ];
   }

   for ( int j = 0; j < 5; ++j ) {
      if ( ( ( s[ j ].rfind( "ay" ) == s[ j ].length() - 2 ) )
          || ( s[ j ].rfind( "r" ) == s[ j ].length() - 1 ) )
         cout << s[ j ] << endl;
   }

   return 0;
}
```

```
Enter a word: bicycle
Enter a word: car
Enter a word: tree
Enter a word: canary
Enter a word: iron
car
```

19.11 Write a program that demonstrates passing a **string** both by reference and by value.

```
// Exercise 19.11 Solution
// Program passes a string by value and
// passes a string by reference.
#include <iostream>
```

```

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

void byValue( string );
void byReference( string& );

int main()
{
   string s = "Standard C++ draft standard";

   cout << "Original string: " << s;

   byValue( s );
   cout << "\nAfter calling byValue: " << s;

   byReference( s );
   cout << "\nAfter calling byReference: " << s << endl;

   return 0;
}

void byValue( string s )
{
   s.erase( 0, 4 );
}

void byReference( string& sRef )
{
   sRef.erase( 0, 9 );
}
```

```
Original string: Standard C++ draft standard
After calling byValue: Standard C++ draft standard
After calling byReference: C++ draft standard
```

19.12 Write a program that separately inputs a first name and a last name and then concatenates the two into a new **`string`**.

```
// Exercise 19.12 Solution
// Program reads a first name and
// last name and concatenates the two.
#include <iostream>
using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

int main()
{
   string first, last;

   cout << "Enter first name: ";
   cin >> first;

   cout << "Enter last name: ";
   cin >> last;

   first.append( " " ).append( last );
   cout << "The full name is: " << first << endl;

   return 0;
}
```

```
Enter first name: Hans
Enter last name: Gruber
The full name is: Hans Gruber
```

19.13 Write a program that plays the game of hangman. The program should pick a word (which is either coded directly into the program or read from a text file) and display the following:

```
Guess the word:    XXXXXX
```

Each **X** represents a letter. If the user guesses correctly, the program should display:

```
Congratulations!!! You guessed my word. Play again? yes/no
```

The appropriate response **yes** or **no** should be input. If the user guesses incorrectly, display the appropriate body part.

After seven incorrect guesses, the user should be hanged. The display should look like

```
  O
 /|\
  |
 / \
```

After each guess you want to display all their guesses.

```
// Exercise 19.13 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

#include <cctype>

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;
```

```

#include <cstdlib>

int main()
{
   string response;  // "yes"/"no" input from user
   int w = 0;        // index for current word
   const int WORDS = 4;   // total number of words

   do {
      const char body[] = " o/|\\|/\\";  // body parts
      string words[ WORDS ] = { "MACAW", "SADDLE",
                                "TOASTER", "XENOCIDE" };
      string xword( words[ w ].length(), '?' ); // masked display
      string::iterator i, ix = xword.begin();
      char letters[ 26 ] = { '\0' };  // letters guessed
      int n = 0, xcount = xword.length();
      bool found = false, solved = false;
      int offset = 0, bodyCount = 0;
      bool hung = false;

      // clear window; system is a stdlib function
      system( "cls" );  // "cls" for DOS; "clear" for unix

      do {
         cout << "\n\nGuess a letter (case does"
              << " not matter):  " << xword << "\n?";
         char temp;
         cin >> temp;  // letter guessed

         if ( !isalpha( temp ) ) {  // validate for letters
            cout << "\nLETTERS ONLY PLEASE\n";
            continue; // next iteration of do/while
         }

         letters[ n ] = toupper( temp ); // convert to uppercase

         system( "cls" );  // "cls" for DOS; "clear" for unix

         // seach word for letters
         i = words[ w ].begin(); // initialize iterator to beginning
         found = false;          // assume letter is not found in word
         offset = 0;             // initial position set to 0

         // replace letter in mask string in all the necessary
         // places. decrement count of characters masked such
         // that we know when word is solved.
         while ( i != words[ w ].end() ) {

            if ( *i == letters[ n ] ) {
               *( ix + offset ) = *i;
               found = true;

               if ( --xcount == 0 )
                  solved = true;
            }

            ++i;
            ++offset;
         }

         if ( !found )   // if the letter was not found
```

```
               ++bodyCount; // increment our count of incorrect guesses.

            // graphically draw the pieces of the body
            // based upon the number of incorrect answers.
            bool newline = false;
            for ( int q = 1; q <= bodyCount; ++q ) {
               if ( q == 1 || q == 5 || q == 7 ) {
                  newline = true;
                  cout << body[ 0 ];   // output space
               }
               else if ( q == 4 )
                  newline = true;
               else
                  newline = false;

               cout << body[ q ];

               if ( newline )
                  cout << '\n';
            }

            // test to see if guesses were exceeded.
            if ( bodyCount == 7 ) {
               cout << "\n\n...GAME OVER...\n";
               hung = true;
               break;
            }

            // display all guesses. note we did not provide
            // the code that would politely refuse duplicates.
            cout << "\n\n\nYour guesses:\n";
            for ( int k = 0; k <= n; k++ )
               cout << setw( 2 ) << letters[ k ];

            ++n;
         } while ( !solved );

         cout << "\n\nWord: " << words[ w ] << "\n\n";

         if ( !hung )
            cout << "\nCongratulations!!! You guessed "
                 << "my word.\n";

         // if we are out of words, then time to exit loop
         if ( ++w >= WORDS )
            break;

         cout << "Play again (yes/no)? ";
         cin >> response;

      } while ( !response.compare( "yes" ) );

      cout << "\nThank you for playing hangman." << endl;

      return 0;
   }
```

```
Your guesses:
 M

Guess a letter (case does not matter):  M????
?a

Your guesses:
 M A
...
 o
/|\
 |
/


Your guesses:
 M A E I O U N K C W

Word: MACAW


Congratulations!!! You guessed my word.
Play again (yes/no)? n

Thank you for playing hangman.
```

19.14 Write a program that inputs a **string** and prints the **string** backwards. Convert all uppercase characters to lowercase and all lowercase characters to uppercase.

```cpp
// Exercise 19.14 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

#include <cctype>

int main()
{
   string s;

   cout << "Enter a string: ";
   getline( cin, s, '\n' );

   string::reverse_iterator r = s.rbegin();

   while ( r != s.rend() ) {
      *r = ( isupper( *r ) ? tolower( *r ): toupper( *r ) );
      cout << *( r++ );
   }

   cout << endl;

   return 0;
}
```

```
Enter a string: The sinking of HMS Titanic
CINATIt smh FO GNIKNIS EHt
```

19.15 Write a program that uses the comparison capabilities introduced in this chapter to alphabetize a series of animal names. Only uppercase letters should be used for the comparisons.

```
// Exercise 19.15 Solution
// NOTE: The problem description should have asked
// the programmer to use a quicksort.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

void output( const string const *, const int );
void quickSort( string [], int, int );

int main()
{
   string animals[] = { "Macaw", "Lion", "Tiger",
                        "Bear", "Toucan", "Zebra",
                        "Puma", "Cat", "Yak", "Boar",
                        "Fox", "Ferret", "Crocodile",
                        "Alligator", "Elk", "Ox",
                        "Horse", "Eagle", "Hawk" };

   cout << "before:";
   output( animals, 19 );

   quickSort( animals, 0, 19 );

   cout << "\nafter:";
   output( animals, 19 );

   return 0;
}

void output( const string * const ani, const int length )
{
   for ( int j = 0; j < length; ++j )
      cout << ( j % 10 ? ' ': '\n' ) << ani[ j ];

   cout << endl;
}

void quickSort( string a[], int first, int last )
{
   int partition( string [], int, int );
   int currentLocation;

   if ( first >= last )
      return;

   currentLocation = partition( a, first, last );
   quickSort( a, first, currentLocation - 1 );
   quickSort( a, currentLocation + 1, last );
}

int partition( string b[], int left, int right )
{
   int pos = left;

   while ( true ) {

      while ( b[ pos ] <= b[ right ] && pos != right )
         --right;

      if ( pos == right )
         return pos;

      if ( b[ pos ] > b[ right ] ) {
         b[ pos ].swap( b[ right ] );
         pos = right;
      }

      while ( b[ left ] <= b[ pos ] && pos != left )
         ++left;

      if ( pos == left )
         return pos;

      if ( b[ left ] > b[ pos ] ) {
         b[ pos ].swap( b[ left ] );
         pos = left;
      }
   }
}
```

```
before:
Macaw Lion Tiger Bear Toucan Zebra Puma Cat Yak Boar
Fox Ferret Crocodile Alligator Elk Ox Horse Eagle Hawk

after:
Alligator Bear Boar Cat Crocodile Eagle Elk Ferret Fox Hawk
Horse Lion Macaw Ox Puma Tiger Toucan Yak Zebra
```

19.16 Write a program that creates a cryptogram out of a **string**. A cryptogram is a message or word where each letter is replaced with another letter. For example the **string**

The birds name was squawk

might be scrambled to form

xms kbypo zhqs fho obrhfu

Note that spaces are not scrambled. In this particular case, **'T'** was replaced with **'x'**, each **'a'** was replaced with **'h'**, etc. Uppercase letters and lowercase letters should be treated the same. Use techniques similar to those in Exercise 19.7.

```
// Exercise 19.16 Solution
// Program creates a cryptogram from a string.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;
using std::getline;

#include <cstdlib>
#include <ctime>
#include <cctype>

void convertToLower( string::iterator,
                     string::iterator );

int main()
{
   string s, alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   string::iterator is, is2, is3;

   srand( time( 0 ) );
   cout << "Enter a string: ";
   getline( cin, s, '\n' ); // allow white space to be read

   is = s.begin();
   convertToLower( is, s.end() );
   string s2( s ); // instantiate s2

   is3 = s2.begin();

   do {
      is2 = is3;  // position location on string s2

      // do not change spaces
      if ( *is == ' ' ) {
         is++;
         continue;
      }

      int x = rand() % alpha.length(); // pick letter
      char c = alpha.at( x );  // get letter
      alpha.erase( x, 1 );     // remove picked letter

      // iterate along s2 doing replacement
      while ( is2 != s2.end() ) {
         if ( *is2 == *is )
            *is2 = c;

         ++is2;
      }

      ++is3;  // position to next element
      ++is;   // position to next element
   } while ( is != s.end() );

   // change s2 to lowercase
   is3 = s2.begin();
   convertToLower( is3, s2.end() );

   // output strings
   cout << "Original string:      " << s
        << "\nCryptogram of string: " << s2 << endl;

   return 0;
}

void convertToLower( string::iterator i,
                     string::iterator e )
{
   while ( i != e ) {
      *i = tolower( *i );
      ++i;
   }
}
```

```
Enter a string: a cold hard rain fell
Original string:      a cold hard rain fell
Cryptogram of string: k yivu rkou okdj cmvv
```

19.17 Modify the previous exercise to allow a user to solve the cryptogram by inputting two characters. The first character specifies the letter in the cryptogram and the second letter specifies the user's guess. For example, if the user inputs `r g`, then the user is guessing that the letter `r` is really a `g`.

19.18 Write a program that inputs a sentence and counts the number of palindromes in the sentence. A palindrome is a word that reads the same backwards and forwards. For example, `"tree"` is not a palindrome but `"noon"` is.

19.19 Write a program that counts the total number of vowels in a sentence. Output the frequency of each vowel.

19.20 Write a program that inserts the characters `"******"` in the exact middle of a `string`.

19.21 Write a program that erases the sequences `"by"` and `"BY"` from a `string`.

```
// Exercise 19.21 Solution
// Program erases "by" or "BY" from strings.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

void deleteBy( string&, string );

int main()
{
   string s;

   cout << "Enter a word:";
   cin >> s;

   deleteBy( s, "by" );
   deleteBy( s, "BY" );

   cout << s << endl;

   return 0;
}


void deleteBy( string& sRef, string z )
{
   int x = sRef.find( z );
   while ( x <= sRef.length() ) {
      sRef.erase( x, 2 );
      x = sRef.find( z );
   }
}
```

```
Enter a word:DERBY
DER
```

19.22 Write a program that inputs a line of text, replaces all punctuation marks with spaces and then uses the C-string library function **strtok** to tokenize the **string** into individual words.

19.23 Write a program that inputs a line of text and prints the text backwards. Use iterators in your solution.

```
// Exercise 19.23 Solution
// Program prints a string backwards.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;
using std::getline;

int main()
{
   string s;

   cout << "Enter a string: ";
   getline( cin, s, '\n' );
   string::reverse_iterator rb = s.rbegin();

   while ( rb != s.rend() ) {
      cout << *rb;
      ++rb;
   }

   cout << endl;

   return 0;
}
```

```
Enter a string: PRINT THIS BACKWARDS
SDRAWKCAB SIHT TNIRP
```

19.24 Write a recursive version of Exercise 19.23.

```
// Exercise 19.23 Solution
// Program recursively prints a string backwards.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;
using std::getline;

void printBackwards( const string::reverse_iterator,
                     string::reverse_iterator );

int main()
{
   string s;

   cout << "Enter a string: ";
   getline( cin, s );
   string::reverse_iterator r = s.rend();

   printBackwards( s.rbegin(), r - 1 );
   cout << endl;

   return 0;
}

void printBackwards( const string::reverse_iterator s,
                     string::reverse_iterator rb )
{
   if ( rb == s - 1 )
      return;

   printBackwards( s, rb - 1 );
   cout << *rb;
}
```

```
Enter a string: automobile
elibomotua
```

19.25 Write a program that demonstrates the use of the **erase** functions that take **iterator** arguments.

19.26 Write a program that generates from the `string "abcdefghijklmnopqrstuvwxyz{"` the following:

```
             a
            bcb
           cdedc
          defgfed
         efghihgfe
        fghijkjihgf
       ghijklmlkjihg
      hijklmnonmlkjih
     ijklmnopqponmlkji
    jklmnopqrsrqponmlkj
   klmnopqrstutsrqponmlk
  lmnopqrstuvwvutsrqponml
 mnopqrstuvwxyxwvutsrqponm
nopqrstuvwxyz{zyxwvutsrqpon
```

```cpp
// Exercise 19.26 Solution
// Program prints a pyramid from a string.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

int main()
{
   string alpha = "abcdefghijklmnopqrstuvwxyz{";
   string::const_iterator x = alpha.begin(), x2;

   for ( int p = 1; p <= 14; ++p ) {
      int w, count = 0;  // set to 0 each iteration

      // output spaces
      for ( int k = 13; k >= p; --k )
         cout << ' ';

      x2 = x;  // set starting point

      // output first half of characters
      for ( int c = 1; c <= p; ++c ) {
         cout << *x2;
         x2++;      // move forwards one letter
         count++;   // keep count of iterations
      }

      // output back half of characters
      for ( w = 1, x2 -= 2; w < count; ++w ) {
         cout << *x2;
         --x2;   // move backwards one letter
      }

      x++;   // next letter
      cout << '\n';
   }

   return 0;
}
```

```
              a
             bcb
            cdedc
           defgfed
          efghihgfe
         fghijkjihgf
        ghijklmlkjihg
       hijklmnonmlkjih
      ijklmnopqponmlkji
     jklmnopqrsrqponmlkj
    klmnopqrstutsrqponmlk
   lmnopqrstuvwvutsrqponml
  mnopqrstuvwxyxwvutsrqponm
 nopqrstuvwxyz{zyxwvutsrqpon
```

19.27 In Exercise 19.7 we asked you to write a simple encryption algorithm. Write a program that will attempt to decrypt a "rot13" message using simple frequency substitution (assume you do not know the key). The most frequent letters in the encrypted phrase should be substituted with the most commonly used English letters (a, e, i, o, u, s, t, r, etc.). Write the possibilities to a f ile. What made the code breaking easy? How can the encryption mechanism be improved?

19.28 Write a version of the bubble sort routine that sorts `string`s. Use function `swap` in your solution.

```
// Exercise 19.28 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
 using std::string;

void output( const string const *, const int );
void bubbleSort( string [], const int );

int main()
{
   string animals[] = { "Macaw", "Lion", "Tiger",
                        "Bear", "Toucan", "Zebra",
                        "Puma", "Cat", "Yak", "Boar",
                        "Fox", "Ferret", "Crocodile",
                        "Alligator", "Elk", "Ox",
                        "Horse", "Eagle", "Hawk" };

   cout << "before:";
   output( animals, 19 );

   bubbleSort( animals, 19 );

   cout << "\nafter:";
   output( animals, 19 );

   return 0;
}
```

```

void output( const string * const ani, const int length )
{
   for ( int j = 0; j < length; ++j )
      cout << ( j % 10 ? ' ': '\n' ) << ani[ j ];

   cout << endl;
}

void bubbleSort( string mals[], const int length )
{
  for ( int pass = 1; pass < length; ++pass )
     for ( int comp = 0; comp < length - pass; ++comp )
        if ( mals[ comp ] > mals[ comp + 1 ] )
           mals[ comp ].swap( mals[ comp + 1 ] );
}
```

```
before:
Macaw Lion Tiger Bear Toucan Zebra Puma Cat Yak Boar
Fox Ferret Crocodile Alligator Elk Ox Horse Eagle Hawk

after:
Alligator Bear Boar Cat Crocodile Eagle Elk Ferret Fox Hawk
Horse Lion Macaw Ox Puma Tiger Toucan Yak Zebra
```

20

Standard Template Library (STL) Solutions

Solutions

20.14 Write a function template **palindrome** that takes as a parameter a **const vector** and returns **true** or **false** depending upon whether the **vector** does or does not read the same forwards as backwards (e.g., a **vector** containing 1, 2, 3, 2, 1 is a palindrome and a **vector** containing 1, 2, 3, 4 is not).

```
// Exercise 20.14 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::ios;

#include <vector>

template < class X >
bool palindrome( const std::vector< X > &vec )
{
   std::vector< X >::const_reverse_iterator r = vec.rbegin();
   std::vector< X >::const_iterator i = vec.begin();

   while ( r != vec.rend() && i != vec.end() )  {
      if ( *r != *i )
         return false;

      ++r;
      ++i;
   }

   return true;
}

template < class Y >
void printVector( const std::vector< Y > &vec )
{
   std::vector< Y >::const_iterator i;

   for ( i = vec.begin(); i != vec.end(); ++i )
      cout << *i << ' ';
}

int main()
{
   std::vector< int > iv;
   std::vector< char > ic;
   int x = 0;

   for ( int i = 75; i >= 65; --i ) {
      iv.push_back( i );
      ic.push_back( static_cast< char > ( i + x ) );

      if ( i <= 70 )
         x += 2;
   }

   printVector( iv );
   cout << ( palindrome( iv ) ? " is " : " is not " )
        << "a palindrome\n";

   printVector( ic );
   cout << ( palindrome( ic ) ? " is " : " is not " )
        << "a palindrome" << endl;

   return 0;
}
```

```
75 74 73 72 71 70 69 68 67 66 65  is not a palindrome
K J I H G F G H I J K  is a palindrome
```

20.15 Modify the program of Fig. 20.29, the Sieve of Eratosthenes, so that if the number the user inputs into the program is not prime, the program displays the prime factors of the number. Remember that a prime number's factors are only 1 and the prime number itself. Every number that is not prime has a unique prime factorization. For example, consider the number 54. The factors of 54 are 2, 3, 3 and 3. When these values are multiplied together, the result is 54. For the number 54, the prime factors outpt should be 2 and 3.

```
// Exercise 20.15 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <bitset>

using std::bitset;

#include <cmath>
```

```

int main()
{
   const int size = 1024;
   int i, value, counter;
   bitset< size > sieve;

   sieve.flip();

   // perform Sieve of Eratosthenes
   int finalBit = sqrt( sieve.size() ) + 1;

   for ( i = 2; i < finalBit; ++i )
      if ( sieve.test( i ) )
         for ( int j = 2 * i; j < size; j += i )
            sieve.reset( j );

   cout << "The prime numbers in the range 2 to 1023 are:\n";

   for ( i = 2, counter = 0; i < size; ++i )
      if ( sieve.test( i ) ) {
         cout << setw( 5 ) << i;

         if ( ++counter % 12 == 0 )
            cout << '\n';
      }

   cout << endl;

   // get a value from the user to determine if it is prime
   cout << "\nEnter a value from 1 to 1023 (-1 to end): ";
   cin >> value;

   while ( value != -1 ) {
      if ( sieve[ value ] )
         cout << value << " is a prime number\n";
      else {
         cout << value << " is not a prime number\n"
              << "prime factor(s): ";

         bool print = true;

         for ( int f = 2; f < size; )
            if ( sieve.test( f ) && value % f == 0 ) {
               if ( print )
                  cout << f << ' '; // output factor

               value /= f;       // modify value

               if ( value <= 1 ) // time to stop
                  break;

               print = false;
            }
            else {
               ++f;  // move to next prime
               print = true;
            }

         cout << '\n';
      }

      cout << "\nEnter a value from 2 to 1023 (-1 to end): ";
      cin >> value;
   }

   return 0;
}
```

```
The prime numbers in the range 2 to 1023 are:
    2    3    5    7   11   13   17   19   23   29   31   37
   41   43   47   53   59   61   67   71   73   79   83   89
   97  101  103  107  109  113  127  131  137  139  149  151
  157  163  167  173  179  181  191  193  197  199  211  223
  227  229  233  239  241  251  257  263  269  271  277  281
  283  293  307  311  313  317  331  337  347  349  353  359
  367  373  379  383  389  397  401  409  419  421  431  433
  439  443  449  457  461  463  467  479  487  491  499  503
  509  521  523  541  547  557  563  569  571  577  587  593
  599  601  607  613  617  619  631  641  643  647  653  659
  661  673  677  683  691  701  709  719  727  733  739  743
  751  757  761  769  773  787  797  809  811  821  823  827
  829  839  853  857  859  863  877  881  883  887  907  911
  919  929  937  941  947  953  967  971  977  983  991  997
 1009 1013 1019 1021

Enter a value from 1 to 1023 (-1 to end): 8
8 is not a prime number
prime factor(s): 2

Enter a value from 2 to 1023 (-1 to end): 444
444 is not a prime number
prime factor(s): 2 3 37

Enter a value from 2 to 1023 (-1 to end): -1
```

20.16 Modify Exercise 20.15 so that if the number the user inputs into the program is not prime, the program displays the prime factors of the number and the number of times that prime factor appears in the unique prime factorization. For example, the output for the number 54 should be `The unique prime factorization of 54 is: 2 * 3 * 3 * 3`

```
// Exercise 20.16 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <iomanip>

using std::setw;
using std::setprecision;
using std::setiosflags;

#include <bitset>

using std::bitset;

#include <cmath>


```

```
int main()
{
   const int size = 1024;
   int i, value, counter;
   bitset< size > sieve;

   sieve.flip();

   // perform Sieve of Eratosthenes
   int finalBit = sqrt( sieve.size() ) + 1;

   for ( i = 2; i < finalBit; ++i )
      if ( sieve.test( i ) )
         for ( int j = 2 * i; j < size; j += i )
            sieve.reset( j );

   cout << "The prime numbers in the range 2 to 1023 are:\n";

   for ( i = 2, counter = 0; i < size; ++i )
      if ( sieve.test( i ) ) {
         cout << setw( 5 ) << i;

         if ( ++counter % 12 == 0 )
            cout << '\n';
      }

   cout << endl;

   // get a value from the user to determine if it is prime
   cout << "\nEnter a value from 1 to 1023 (-1 to end): ";
   cin >> value;

   while ( value != -1 ) {
      if ( sieve[ value ] )
         cout << value << " is a prime number\n";
      else {
         cout << value << " is not a prime number\n"
              << "prime factor(s): ";

         for ( int f = 2; f < size; )
            if ( sieve.test( f ) && value % f == 0 ) {
               cout << f << ' '; // output factor
               value /= f;       // modify value

               if ( value <= 1 ) // time to stop
                  break;
            }
            else
               ++f;  // move to next prime

         cout << '\n';
      }

      cout << "\nEnter a value from 2 to 1023 (-1 to end): ";
      cin >> value;
   }

   return 0;
}
```

```
The prime numbers in the range 2 to 1023 are:
    2    3    5    7   11   13   17   19   23   29   31   37
   41   43   47   53   59   61   67   71   73   79   83   89
   97  101  103  107  109  113  127  131  137  139  149  151
  157  163  167  173  179  181  191  193  197  199  211  223
  227  229  233  239  241  251  257  263  269  271  277  281
  283  293  307  311  313  317  331  337  347  349  353  359
  367  373  379  383  389  397  401  409  419  421  431  433
  439  443  449  457  461  463  467  479  487  491  499  503
  509  521  523  541  547  557  563  569  571  577  587  593
  599  601  607  613  617  619  631  641  643  647  653  659
  661  673  677  683  691  701  709  719  727  733  739  743
  751  757  761  769  773  787  797  809  811  821  823  827
  829  839  853  857  859  863  877  881  883  887  907  911
  919  929  937  941  947  953  967  971  977  983  991  997
 1009 1013 1019 1021

Enter a value from 1 to 1023 (-1 to end): 99
99 is not a prime number
prime factor(s): 3 3 11

Enter a value from 2 to 1023 (-1 to end): 888
888 is not a prime number
prime factor(s): 2 2 2 3 37

Enter a value from 2 to 1023 (-1 to end): -1
```

21

Standard C++ Language Additions Solutions

Solutions

21.3 Fill in the blanks for each of the following:

a) Operator _________ is used to determine an object's type at run-time.
ANS: `typeid` or `dynamic_cast`

b) Keyword _________ specifies that a `namespace` or `namespace` member is being used.
ANS: `using`

c) Operator _________ is the operator keyword for logical OR.
ANS: `or`

d) Storage specifier _________ allows a member of a `const` object to be modified.
ANS: `mutable`

21.4 State which of the following are true and which are false. If a statement is false, explain why.

a) The validity of a `static_cast` operation is checked at compile-time.
ANS: True.

b) The validity of a `dynamic_cast` operation is checked at run-time.
ANS: True.

c) The name `typeid` is a keyword.
ANS: True.

d) Keyword `explicit` may be applied to constructors, member functions and data members.
ANS: False. Keyword `explicit` can only be used with constructors.

21.5 What does each expression evaluate to? (*Note:* Some expressions may generate errors; if so, say what the cause of the error is.)

a) `cout << false;`
ANS: 0 is output.

b) `cout << ( bool b = 8 );`
ANS: Error. Most compilers will not allow a variable to be declared in this manner. If the compiler permits the expression, 1 is output.

c) `cout << ( a = true );   // a is of type int`
ANS: 1 is output.

d) `cout << ( *ptr + true && p );   // *ptr is 10 and p is 8.88`
ANS: 1 is output.

e)
```
// *ptr is 0 and m is false
bool k = ( *ptr * 2 || ( true + 24 ) );
```

ANS: `k` is assigned 1.

f) `bool s = true + false;`

ANS: `true` is assigned to `s`.

g) `cout << boolalpha << false << setw( 3 ) << true;`

ANS: `falsetrue` is output.

21.6 Write a `namespace Currency` which defines constant members `ONE`, `TWO`, `FIVE`, `TEN`, `TWENTY`, `FIFTY` and `HUNDRED`. Write two short programs that use `Currency`. One program should make all constants available and the other program should only make `FIVE` available.

```
// Exercise 21.6 Part A Solution
// Program makes namespace members accessible.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

namespace Currency {
   enum Money { ONE = 1, TWO, FIVE = 5, TEN = 10,
                TWENTY = 20, FIFTY = 50,
                HUNDRED = 100 };
}

int main()
{
   using namespace Currency;

   cout << "TWO's value is: " << TWO
        << "\nTEN's value is: " << TEN << endl;

   return 0;
}
```

```
TWO's value is: 2
TEN's value is: 10
```

```
// Exercise 21.6 Part B Solution
// Program makes one namespace member accessible.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

namespace Currency {
   enum Money { ONE = 1, TWO, FIVE = 5, TEN = 10,
                TWENTY = 20, FIFTY = 50,
                HUNDRED = 100 };
}

int main()
{
   using Currency::FIVE;

   cout << "FIVE's value is: " << FIVE << endl;

   return 0;
}
```

```
FIVE's value is: 5
```

21.7 Write a program that uses the **reinterpret_cast** operator to cast different pointer types to **int**. Do any conversions result in syntax errors?

```
// Exercise 21.7 Solution
// Program exercises reinterpret cast
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

#include <string>
using std::string;

int main()
{
   // declare variables
   int x;
   double d;
   float f;
   long l;
   short s;
   string z;
   char c;

   // declare and initialize pointers
   int *xPtr = &x;
   double *dPtr = &d;
   float *fPtr = &f;
   long *lPtr = &l;
   short *sPtr = &s;
   string *zPtr = &z;
   char *cPtr = &c;
   void *vPtr = &z;

   // test reinterpret_cast
   cout << "reinterpret_cast< int > ( xPtr ) = "
        << reinterpret_cast< int > ( xPtr )
        << "\nreinterpret_cast< int > ( dPtr ) = "
        << reinterpret_cast< int > ( dPtr )
        << "\nreinterpret_cast< int > ( fPtr ) = "
        << reinterpret_cast< int > ( fPtr ) ;

   cout << "\nreinterpret_cast< int > ( lPtr ) = "
        << reinterpret_cast< int > ( lPtr )
        << "\nreinterpret_cast< int > ( sPtr ) = "
        << reinterpret_cast< int > ( sPtr )
        << "\nreinterpret_cast< int > ( zPtr ) = "
        << reinterpret_cast< int > ( zPtr )
        << "\nreinterpret_cast< int > ( cPtr ) = "
        << reinterpret_cast< int > ( cPtr )
        << "\nreinterpret_cast< int > ( vPtr ) = "
        << reinterpret_cast< int > ( vPtr ) << endl;
   return 0;
}
```

```
reinterpret_cast< int > ( xPtr ) = 1245040
reinterpret_cast< int > ( dPtr ) = 1245032
reinterpret_cast< int > ( fPtr ) = 1245024
reinterpret_cast< int > ( lPtr ) = 1245020
reinterpret_cast< int > ( sPtr ) = 1245016
reinterpret_cast< int > ( zPtr ) = 1245000
reinterpret_cast< int > ( cPtr ) = 1244996
reinterpret_cast< int > ( vPtr ) = 1245000
```

21.8 Write a program that uses the `static_cast` operator to cast some fundamental data types to `int`. Does the compiler allow the casts to `int`?

```
// Exercise 21.8 Solution
// Program exercises static casting.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

int main()
{
   // declare variables
   int x = 8;
   double d = 22.22;
   float f = 33.33f;  // floating point representation
   long l = 888888;
   short s = 32000;
   unsigned u = 65000;
   char c = 'U';   // ascii of 85
   long double ld = 999999999.00;
   unsigned char uc = 250;
   unsigned long ul = 777777;
   unsigned short us = 55555;

   // test static_cast
   cout << "static_cast< int > ( x ) = "
        << static_cast< int > ( x )
        << "\nstatic_cast< int > ( d ) = "
        << static_cast< int > ( d )
        << "\nstatic_cast< int > ( f ) = "
        << static_cast< int > ( f );

   cout << "\nstatic_cast< int > ( l ) = "
        << static_cast< int > ( l )
        << "\nstatic_cast< int > ( s ) = "
        << static_cast< int > ( s )
        << "\nstatic_cast< int > ( u ) = "
        << static_cast< int > ( u );

   cout << "\nstatic_cast< int > ( c ) = "
        << static_cast< int > ( c )
        << "\nstatic_cast< int > ( ld ) = "
        << static_cast< int > ( ld )
```

```
          << "\nstatic_cast< int > ( uc ) = "
          << static_cast< int > ( uc );

   cout << "\nstatic_cast< int > ( ul ) = "
        << static_cast< int > ( ul )
        << "\nstatic_cast< int > ( us ) = "
        << static_cast< int > ( us ) << endl;

   return 0;
}
```

```
static_cast< int > ( x ) = 8
static_cast< int > ( d ) = 22
static_cast< int > ( f ) = 33
static_cast< int > ( l ) = 888888
static_cast< int > ( s ) = 32000
static_cast< int > ( u ) = 65000
static_cast< int > ( c ) = 85
static_cast< int > ( ld ) = 999999999
static_cast< int > ( uc ) = 250
static_cast< int > ( ul ) = 777777
static_cast< int > ( us ) = 55555
```

21.9 Write a program that demonstrates upcasting from a derived class to a base class. Use the **`static_cast`** operator to perform the upcast.

```
// Exercise 21.9 Solution
// Program upcasts with static_cast.
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

class Base {
public:
   void print() const { cout << "BASE"; }
};

class Derived : public Base {
public:
   void print() const { cout << "DERIVED"; }
};

int main()
{
   Base *bPtr;    // base class pointer
   Derived d, *dPtr;

   dPtr = &d;  // point to d

   // upcast from Derived * to Base *
   bPtr = static_cast< Base * > ( dPtr );
   bPtr -> print();  // invoke function print

   cout << endl;

   return 0;
}
```

```
BASE
```

21.10 Write a program that creates an **explicit** constructor that takes two arguments. Does the compiler permit this? Remove **explicit** and attempt an implicit conversion. Does the compiler permit this?

21.11 What is the benefit of an **explicit** constructor?
ANS: An **explicit** constructor prevents arguments from being implicitly converted.

21.12 Write a program that creates a class containing two constructors. One constructor should take a single **int** argument. The second constructor should take one **char *** argument. Write a driver program that constructs several different objects, each object having a different type passed into the constructor. Do not use **explicit**. What happens? Now use **explicit** only for the constructor that takes one **int**. What happens?

21.13 Given the following **namespace**s, answer whether or not each statement is *true* or *false*. Explain any *false* answers.

```
#include <string>
namespace Misc {
   using namespace std;
   enum Countries { POLAND, SWITZERLAND, GERMANY,
                    AUSTRIA, CZECH_REPUBLIC };
   int kilometers;
   string s;

   namespace Temp {
      short y = 77;
      Car car;   // assume definition exists
   }
}

namespace ABC {
   using namespace Misc::Temp;
   void *function( void *, int );
}
```

a) Variable **y** is accessible within **namespace ABC**.
ANS: True.
b) Object **s** is accessible within **namespace Temp**.
ANS: True.
c) Constant **POLAND** is not accessible within **namespace Temp**.
ANS: False.
d) Constant **GERMANY** is accessible within **namespace ABC**.
ANS: False.
e) Function **function** is accessible to **namespace Temp**.
ANS: False.
f) Namespace **ABC** is accessible to **Misc**.
ANS: False.
g) Object **car** is accessible to **Misc**.
ANS: True.

21.14 Compare and contrast **mutable** and **const_cast**. Give at least one example of when one might be preferred over the other. Note: This exercise does not require any code to be written.

ANS: Operator **const_cast** is used to cast away **const** or **volatile** qualifications. Users of a class usually will not be aware of **const_cast** operations, because this implementation is typically hidden. Storage class specifier **mutable** allows a variable to be modified even if the object is **const**. Users of the class are not likely to be aware of a **mutable** member. Members that are**mutable** usually correspond to some "secret" implementation**mutable** members are always modifiable, whereas **const_cast** operations are confined to the line where the cast is performed.

21.15 Write a program that uses `const_cast` to modify a `const` variable. (*Hint:* Use a pointer in your solution to point to the `const` identifier.)

```
// Exercise 21.15 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

int main()
{
   const char c = 'A';
   const char *ptr = &c;

   cout << "c is " << *ptr;

   *const_cast< char * > ( ptr ) = 'Z';

   cout << "\nc is " << *ptr << endl;

   return 0;
}
```

```
c is A
c is Z
```

21.16 What problem does `virtual` base classes solve?
ANS: `virtual` base classes solve the problem of "diamond inheritance" where a derived class recieves duplicate subobjects from its base classes. With `virtual` base classes, only one copy of the subobject is inherited into the derived class (at the bottom of the diamond).

21.17 Write a program that uses `virtual` base classes. The class at the top of the hierarchy should provide a constructor that takes at least one argument (i.e., do not provide a default constructor). What challenges does this present for the inheritance hierarchy?

```
// Exercise 21.17 Solution
#include <iostream>

using std::cout;
using std::endl;
using std::cin;
using std::ios;

class Base {
public:
   Base( int n ) { num = n; }
   void print() const { cout << num; }
private:
   int num;
};

class D1 : virtual public Base {
public:
   D1(): Base( 3 ) {}
};

```

```
class D2 : virtual public Base {
public:
   D2(): Base( 5 ) {}
};

class Multi : public D1, D2 {
public:
   Multi( int a ): Base( a ) {}
};

int main()
{
   Multi m( 9 );

   m.print();

   cout << endl;

   return 0;
}
```

```
9
```

21.18 Find the error(s) in each of the following. When possible, explain how to correct each error.

a)
```
namespace Name {
   int x, y;
   mutable int z;
};
```
ANS: A `mutable` member can only belong to a class.

b) `int integer = const_cast< int >( double );`

ANS: Operator `const_cast` cannot be used to convert a `float` to an `int`. Either `static_cast` or `reinterpret_cast` should be used. Note: The keyword `double` in parentheses should be a floating-point value.

c) `namespace PCM( 111, "hello" );   // construct namespace`

ANS: A `namespace` cannot be constructed, because it only defines a scope.

d) `explicit int x = 99;`

ANS: Keyword `explicit` can only be applied to a constructor definition.

Appendix

C++ Multimedia Cyber Classroom: Solutions Provided on CD

This appendix contains the complete list of solutions provided on the *C++ Multimedia Cyber Classroom* CD-ROM. This will help instructors avoid the exercises for which students have solutions if they purchase the Cyber Classroom product. Note that key exercises like the Simpletron Simulator (Chapter 5) and the compiler (Chapter 15) are not provided on the CD.

Chapter 1:	1.11, 1.13, 1.16, 1.19, 1.21, 1.24, 1.25, 1.30, 1.31, 1.34, 1.37
Chapter 2:	2.14, 2.16, 2.18, 2.20, 2.24, 2.26, 2.29, 2.31, 2.33, 2.37a/b, 2.40, 2.43, 2.45, 2.48, 2.49, 2.54, 2.58, 2.60, 2.63
Chapter 3:	3.12, 3.13, 3.16, 3.18, 3.20, 3.22, 3.27, 3.29, 3.33, 3.35, 3.38, 3.40, 3.44, 3.48, 3.53, 3.56, 3.58
Chapter 4:	4.10, 4.12, 4.15, 4.20, 4.23, 4.29, 4.30, 4.33, 4.36, 4.38
Chapter 5:	5.9, 5.12, 5.22, 5.25, 5.26, 5.27, 5.30, 5.31, 5.33, 5.37, 5.42a/b/c, 5.46
Chapter 6:	6.5, 6.7, 6.8, 6.12, 6.16
Chapter 7:	7.7, 7.8
Chapter 8:	8.12, 8.15, 8.19
Chapter 9:	9.10, 9.12
Chapter 10:	10.6, 10.7, 10.14
Chapter 11:	11.7, 11.9, 11.10, 11.12, 11.15, 11.18
Chapter 12:	12.3, 12.7, 12.9, 12.14, 12.15, 12.19, 12.23
Chapter 13:	13.21, 13.27, 13.29, 13.31, 13.34, 13.42, 13.43
Chapter 14:	14.7, 14.12, 14.14
Chapter 15:	15.6, 15.8, 15.10, 15.11, 15.17, 15.20, 15.24
Chapter 16:	16.9, 16.12, 16.21, 16.22, 16.26, 16.30
Chapter 17:	17.4, 17.6, 17.8 ,17.10
Chapter 18:	18.3, 18.8, 18.10
Chapter 19:	19.4, 19.5, 19.8, 19.11, 19.12, 19.21, 19.23
Chapter 20:	20.15
Chapter 21:	21.3, 21.4, 21.7, 21.8, 21.13, 21.15