Instructor's Manual

for

C How to Program, 4/e

Deitel & Deitel

Contents

1

Introduction to Computers, the Internet and the World Wide Web: Solutions

SOLUTIONS

1.3 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) a word processor program
ANS: software.

1.4 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.

1.5 Translator programs such as assemblers and compilers convert programs from one language (referred to as the *source* language) to another language (referred to as the *object* language). Determine which of the following statements are true and which are false:

a) A compiler translates high-level language programs into object language.
ANS: True.
b) An assembler translates source language programs into machine language programs.
ANS: True.
c) A compiler converts source language programs into object language programs.
ANS: False.
d) High-level languages are generally machine-dependent.
ANS: False.
e) A machine language program requires translation before the program can be run on a computer.
ANS: False.

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

a) Devices from which users access timesharing computer systems are usually called __________.
ANS: terminals.
b) A computer program that converts assembly language programs to machine language programs is called __________.
ANS: an assembler.
c) The logical unit of the computer that receives information from outside the computer for use by the computer is called __________.
ANS: The input unit.
d) The process of instructing the computer to solve specific problems is called __________.
ANS: computer programming.
e) What type of computer language uses English-like abbreviations for machine language instructions? __________.
ANS: a high-level language.
f) 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: The output unit.
g) The general name for a program that converts programs written in a certain computer language into machine language is __________.
ANS: compiler.
h) Which logical unit of the computer retains information? __________.
ANS: memory unit and secondary storage unit.
i) Which logical unit of the computer performs calculations? __________.
ANS: arithmetic and logical unit.
j) Which logical unit of the computer makes logical decisions? __________.
ANS: arithmetic and logical unit
k) The commonly used abbreviation for the computer's control unit is __________.
ANS: CPU.
l) The level of computer language most convenient to the programmer for writing programs quickly and easily is __________.
ANS: high-level language.
m) The only language that a computer can directly understand is called that computer's __________.
ANS: machine language.
n) Which logical unit of the computer coordinates the activities of all the other logical units? __________.
ANS: central processing unit.

1.7 State whether each of the following is *true* or *false*. If *false*, explain your answer.

a) Machine languages are generally machine dependent.
ANS: True. Machine languages are closely related to the hardware of a particular machine.
b) Timesharing truly runs several users simultaneously on a computer.
ANS: False. Time sharing systems split CPU time amongst several users so that the users appear to be operating simultaneously
c) Like other high-level languages, C is generally considered to be machine independent.
ANS: True. C programs can be written on most machines, and with some care, C programs can be written on one machine and run on many machines with few changes or no changes.

1.8 Discuss the meaning of each of the following names:

a) `stdin`
ANS: This refers to the standard input device. The standard input device is normally connected to the keyboard
b) `stdout`
ANS: This refers to the standard output device. The standard output device is normally connected to the computer screen.
c) `stderr`
ANS: This refers to the standard error device. Error messages are normally sent to this device which is typically connected to the computer screen.

1.9 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 used for object-oriented programming because it is a superset of the C programming language and C is widely used.

1.10 Which programming language is best described by each of the following?

a) Developed by IBM for scientific and engineering applications.
ANS: FORTRAN

b) Developed specifically for business applications.
ANS: COBOL

c) Developed for teaching structured programming.
ANS: Pascal

d) Named after the world's first computer programmer.
ANS: Ada

e) Developed to familiarize novices with programming techniques.
ANS: BASIC

f) Specifically developed to help programmers migrate to .NET.
ANS: C#

g) Known as the development language of UNIX.
ANS: C

h) Formed primarily by adding object-oriented programming to C.
ANS: C++

i) Succeeded initially because of its ability to create Web pages with dynamic content.
ANS: Java

2

Introduction to C Programming: Solutions

SOLUTIONS:

2.7 Identify and correct the errors in each of the following statements (*Note:* there may be more than one error per statement):

a) `scanf( "d", value );`

ANS: `scanf( "%d", &value );`

b) `printf( "The product of %d and %d is %d"\n, x, y );`

ANS: `printf( "The product of %d and %d is %d\n", x, y, z );`

c) `firstNumber + secondNumber = sumOfNumbers`

ANS: `sumOfNumbers = firstNumber + secondNumber;`

d)
```
if ( number => largest )
   largest == number;
```

ANS:
```
if ( number >= largerst )
   largest = number;
```

e) `*/ Program to determine the largest of three integers /*`

ANS: `/* Program to determine the largest of three integers */`

f) `Scanf( "%d", anInteger );`

ANS: `scanf( "%d", &anInteger );`

g) `printf( "Remainder of %d divided by %d is\n", x, y, x % y );`

ANS: `printf( "Remainder of %f divided by %d is %d\n", x, y, x % y );`

h)
```
if ( x = y );
   printf( %d is equal to %d\n", x, y );
```

ANS:
```
if ( x == y )
   printf( "%d is equal to %d\n", x, y );
```

i) `print( "The sum is %d\n," x + y );`

ANS: `printf( "The sum is %d\n", x + y );`

j) `Printf( "The value you entered is: %d\n, &value );`

ANS: `printf( "The value you entered is: %d\n", value );`

2.8 Fill in the blanks in each of the following:

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

ANS: comments.

b) The function used to display information on the screen is __________.

ANS: `printf`.

c) A C statement that makes a decision is __________.
ANS: `if`.
d) Calculations are normally performed by __________ statements.
ANS: assignment.
e) The __________ function inputs values from the keyboard.
ANS: `scanf`.

2.9 Write a single C statement or line that accomplishes each of the following:
a) Print the message "`Enter two numbers`."
ANS: `printf( "Enter two numbers\n" );`
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 place these values in integer variables `a`, `b` and `c`.
ANS: `scanf( "%d%d%d", &a, &b, &c );`

2.10 State which of the following are *true* and which are *false*. If *false*, explain your answer.
a) C operators are evaluated from left to right.
ANS: False. Some operators are evaluated left to right and others are evaluated from right to left depending on their associativity (see Appendix C).
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.
c) The statement `printf("a = 5;");` is a typical example of an assignment statement.
ANS: False. The statement prints `a = 5;` on the screen.
d) A valid arithmetic expression containing no parentheses is evaluated from left to right.
ANS: False. Multiplication, division, and modulus are all evaluated first from left to right, then addition and subtraction are evaluated from left to right.
e) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22`, `2h`.
ANS: False. Those beginning with a number are invalid.

2.11 Fill in the blanks in each of the following:
a) What arithmetic operations are on the same level of precedence as multiplication? __________.
ANS: division, modulus.
b) When parentheses are nested, which set of parentheses is evaluated first in an arithmetic expression? __________.
ANS: The innermost pair of parenthesis.
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.

2.12 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) `printf( "%d", x );`
ANS: `2`
b) `printf( "%d", x + x );`
ANS: `4`
c) `printf( "x=" );`
ANS: `x=`
d) `printf( "x=%d", x );`
ANS: `x=2`
e) `printf( "%d = %d", x + y, y + x );`
ANS: `5 = 5`
f) `z = x + y;`
ANS: Nothing. Value of `x + y` is assigned to `z`.
g) `scanf( "%d%d", &x, &y );`
ANS: Nothing. Two integer values are read into the location of `x` and the location of `y`.
h) `/* printf( "x + y = %d", x + y ); */`
ANS: Nothing. This is a comment.
i) `printf( "\n" );`

ANS: A newline character is printed, and the cursor is positioned at the beginning of the next line on the screen.

2.13 Which, if any, of the following C statements contain variables involved in destructive read-in?

a) `scanf( "%d%d%d%d%d", &b, &c, &d, &e, &f );`
b) `p = i + j + k + 7;`
c) `printf( "Destructive read-in" );`
d) `printf( "a = 5" );`

ANS: (a).

2.14 Given the 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: (a), (d), and (e).

2.15 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: `*` is first, `/` is second, `+` is third, and `-` is fourth. Value of `x` is `15`.

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

ANS: `%` is first, `*` is second, `/` is third, `+` is fourth, `-` is fifth. Value of `x` is `3`.

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

ANS: `5 6 4 2 3 1.` Value of `x` is `338`.

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

ANS:

```c
/* Exercise 2.16 Solution */
#include <stdio.h>

int main()
{
   int x; /* define first number */
   int y; /* define second number */

   printf( "Enter two numbers: "); /* prompt user */
   scanf( "%d%d", &x, &y );        /* read values from keyboard */

   /* output results */
   printf( "The sum is %d\n", x + y );
   printf( "The product is %d\n", x * y );
   printf( "The difference is %d\n", x - y );
   printf( "The quotient is %d\n", x / y );
   printf( "The modulus is %d\n", x % y );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter two numbers: 20 5
The sum is 25
The product is 100
The difference is 15
The quotient is 4
The modulus is 0
```

2.17 Write a program that prints the numbers 1 to 4 on the same line. Write the program using the following methods.

a) Using one `printf` statement with no conversion specifiers.
b) Using one `printf` statement with four conversion specifiers.
c) Using four `printf` statements.

ANS:

```
/* Exercise 2.17 Solution */
#include <stdio.h>

int main()
{
   printf( "1 2 3 4\n\n" ); /* part a */

   printf( "%d %d %d %d\n\n", 1, 2, 3, 4 ); /* part b */

   printf( "1 " ); /* part c */
   printf( "2 " );
   printf( "3 " );
   printf( "4\n" );

   return 0; /* indicates successful termination */

} /* end main */
```

```
1 2 3 4

1 2 3 4

1 2 3 4
```

2.18 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`." Use only the single-selection form of the `if` statement you learned in this chapter.

ANS:

```
/* Exercise 2.18 Solution */
#include <stdio.h>

int main()
{
   int x; /* define first number */
   int y; /* define second number */

   printf( "Enter two numbers: " ); /* prompt */
   scanf( "%d%d", &x, &y );         /* read two integers */

   /* compare the two numbers */
   if ( x > y ) {
      printf( "%d is larger\n", x );
   } /* end if */

   if ( x < y ) {
      printf( "%d is larger\n", y );
   } /* end if */

   if ( x == y ) {
      printf( "These numbers are equal\n" );
   } /* end if */

```

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

} /* end main */
```

```
Enter two numbers: 5 20
20 is larger
```

```
Enter two numbers: 239 92
239 is larger
```

```
Enter two numbers: 17 17
These numbers are equal
```

2.19 Write a program that inputs three different integers from the keyboard, then prints the sum, the average, the product, the smallest and the largest of these numbers. Use only the single-selection form of the `if` statement you learned in this chapter. 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
```

ANS:

```
/* Exercise 2.19 Solution */
#include <stdio.h>

int main()
{
   int a;        /* define first integer */
   int b;        /* define second integer */
   int c;        /* define third integer */
   int smallest; /* smallest integer */
   int largest;  /* largest integer */

   printf( "Input three different integers: " ); /* prompt user */
   scanf( "%d%d%d", &a, &b, &c ); /* read three integers */

   /* output sum, average and product of the three integers */
   printf( "Sum is %d\n", a + b + c );
   printf( "Average is %d\n", ( a + b + c ) / 3 );
   printf( "Product is %d\n", a * b * c );

   smallest = a; /* assume first number is the smallest */

   if ( b < smallest ) { /* is b smaller? */
      smallest = b;
   } /* end if */

   if ( c < smallest ) { /* is c smaller? */
      smallest = c;
   } /* end if */
```

```

   printf( "Smallest is %d\n", smallest );

   largest = a; /* assume first number is the largest */

   if ( b > largest ) { /* is b larger? */
      largest = b;
   } /* end if */

   if ( c > largest ) { /* is c larger? */
      largest = c;
   } /* end if */

   printf( "Largest is %d\n", largest );

   return 0; /* indicate successful termination */

} /* end main */
```

2.20 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. Perform each of these calculations inside the `printf` statement(s) and use the conversion specifier `%f`. [*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.]

ANS:

```
/* Exercise 2.20 Solution */
#include <stdio.h>

int main()
{
   int radius; /* circle radius */

   printf( "Input the circle radius: " ); /* prompt user */
   scanf( "%d", &radius ); /* read integer radius */

   /* calculate and output diameter, circumference and area */
   printf( "\nThe diameter is %d\n", 2 * radius );
   printf( "The circumference is %f\n", 2 * 3.14159 * radius );
   printf( "The area is %f\n", 3.14159 * radius * radius );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Input the circle radius: 9

The diameter is 18
The circumference is 56.548620
The area is 254.468790
```

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

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

ANS:

```c
/* Exercise 2.21 Solution */
#include <stdio.h>

int main()
{
   printf( "*********       ***        *          *\n" );
   printf( "*       *     *     *     ***        * *\n" );
   printf( "*       *    *       *   *****      *   *\n" );
   printf( "*       *    *       *     *       *     *\n" );
   printf( "*       *    *       *     *      *       *\n" );
   printf( "*       *    *       *     *       *     *\n" );
   printf( "*       *    *       *     *        *   *\n" );
   printf( "*       *     *     *      *         * *\n" );
   printf( "*********       ***        *          *\n" );

   return 0; /* indicates successful termination */

} /* end main */
```

2.22 What does the following code print?

```c
printf( "*\n**\n***\n****\n*****\n" );
```

ANS:

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

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

ANS:

```c
/* Exercise 2.23 Solution */
#include <stdio.h>

int main()
{
   int largest;  /* largest integer */
   int smallest; /* smallest integer */
   int int1;     /* define int1 for user input */
   int int2;     /* define int2 for user input */
```

```
   int int3;     /* define int3 for user input */
   int temp;     /* temporary integer for swapping */

   printf( "Input 5 integers: " ); /* prompt user and read 5 ints */
   scanf( "%d%d%d%d%d", &largest, &smallest, &int1, &int2, &int3 );

   if ( smallest > largest ) { /* make comparisons */
      temp = largest;
      largest = smallest;
      smallest = temp;
   } /* end if */

   if ( int1 > largest ) {
      largest = int1;
   } /* end if */

   if ( int1 < smallest ) {
      smallest = int1;
   } /* end if */

   if ( int2 > largest ) {
      largest = int2;
   } /* end if */

   if ( int2 < smallest ) {
      smallest = int2;
   } /* end if */

   if ( int3 > largest ) {
      largest = int3;
   } /* end if */

   if ( int3 < smallest ) {
      smallest = int3;
   } /* end if */

   printf( "The largest value is %d\n", largest );
   printf( "The smallest value is %d\n", smallest );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Input 5 integers: 9 4 5 8 7
The largest value is 9
The smallest value is 4
```

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

ANS:

```
/* Exercise 2.24 Solution */
#include <stdio.h>

int main()
{
   int integer; /* integer input by user */

   printf( "Input an integer: " ); /* prompt */
```

```c
   scanf( "%d", &integer );        /* read integer */

   /* test if integer is even */
   if ( integer % 2 == 0 ) {
      printf( "%d is an even integer\n", integer );
   } /* end if */

   /* test if integer is odd */
   if ( integer % 2 != 0 ) {
      printf( "%d is an odd integer\n", integer );
   } /* end if */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Input an integer: 78
78 is an even integer
```

```
Input an integer: 79
79 is an odd integer
```

2.25 Print your initials in block letters down the page. Construct each block letter out of the letter it represents as shown below.

```
PPPPPPPPP
    P   P
    P   P
    P   P
     P P

  JJ
 J
J
 J
  JJJJJJJ

DDDDDDDDD
D       D
D       D
 D     D
  DDDDD
```

ANS:

```c
/* Exercise 2.25 Solution */
#include <stdio.h>

int main()
{
   printf( "PPPPPPPPP\n" );
   printf( "    P   P\n" );
   printf( "    P   P\n" );
   printf( "    P   P\n" );
   printf( "     P P\n" );
   printf( "\n" );
   printf( "  JJ\n" );
   printf( " J\n" );
   printf( "J\n" );
```

```
   printf( " J\n" );
   printf( "  JJJJJJ\n" );
   printf( "\n" );
   printf( "DDDDDDDDD\n" );
   printf( "D       D\n" );
   printf( "D       D\n" );
   printf( " D     D\n" );
   printf( "  DDDDD\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

2.26 Write a program that reads in two integers and determines and prints if the first is a multiple of the second. [*Hint*: Use the remainder operator.]

ANS:

```
/* Exercise 2.26 Solution */
#include <stdio.h>

int main()
{
   int integer1; /* first integer */
   int integer2; /* second integer */

   printf( "Input two integers: " );      /* prompt user */
   scanf( "%d%d", &integer1, &integer2 ); /* read two integers */

   /* use remainder operator */
   if ( integer1 % integer2 == 0 ) {
      printf( "%d is a multiple of %d ", integer1, integer2 );
      printf( "by a factor of %d\n", integer1 / integer2 );
   } /* end if */

   if ( integer1 % integer2 != 0 ) {
      printf( "%d is not a multiple of %d\n", integer1, integer2 );
   } /* end if */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Input two integers: 88 11
88 is a multiple of 11 by a factor of 8
```

```
Input two integers: 777 5
777 is not a multiple of 5
```

2.27 Display the following checkerboard pattern with eight `printf` statements and then display the same pattern with as few `printf` statements as possible.

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

ANS:

```c
/* Exercise 2.27 Solution */
#include <stdio.h>

int main()
{
   printf( "With eight printf() statements: \n" );

   printf( "* * * * * * * *\n" );
   printf( " * * * * * * * *\n" );
   printf( "* * * * * * * *\n" );
   printf( " * * * * * * * *\n" );
   printf( "* * * * * * * *\n" );
   printf( " * * * * * * * *\n" );
   printf( "* * * * * * * *\n" );
   printf( " * * * * * * * *\n" );

   printf( "\nNow with one printf() statement: \n" );

   printf( "* * * * * * * *\n * * * * * * * *\n"
          "* * * * * * * *\n * * * * * * * *\n"
          "* * * * * * * *\n * * * * * * * *\n"
          "* * * * * * * *\n * * * * * * * *\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

```
With eight printf() statements:
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *

Now with one printf() statement:
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

2.28 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 the 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.

2.29 Here's 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 the integer equivalent of uppercase `A` for example, by executing the statement

```
printf( "%d", 'A' );
```

Write a C program that prints the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. As a minimum, determine the integer equivalents of the following: `A B C a b c 0 1 2 $ * + /` and the blank character.

ANS:

```c
/* Exercise 2.29 Solution */
#include <stdio.h>

int main()
{
   char intEquivalent; /* letter, digit or character */

   printf( "Input a letter, digit, or character: " ); /* prompt */
   scanf( "%c", &intEquivalent ); /* read user input */

   printf( "%c's integer equivalent is %d\n", intEquivalent,
          intEquivalent );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Input a letter, digit, or character: %
%'s integer equivalent is 37
```

```
Input a letter, digit, or character: y
y's integer equivalent is 121
```

```
Input a letter, digit, or character: 0
0's integer equivalent is 48
```

2.30 Write a program that inputs one five-digit number, separates the number into its individual digits and prints the digits separated from one another by three spaces each. [*Hint*: Use combinations of integer division and the remainder operation.] For example, if the user types in `42139`, the program should print

```
4   2   1   3   9
```

ANS:

```c
/* Exercise 2.30 Solution */
#include <stdio.h>

int main()
{
   int number; /* number input by user */
   int temp1;  /* first temporary integer */
   int temp2;  /* second temporary integer */

   printf( "Enter a five-digit number: " ); /* prompt user */
   scanf( "%d", &number ); /* read integer */

   printf( "%d ", number / 10000 ); /* print left-most digit */
   temp2 = number % 10000;

   printf( "  %d ", temp2 / 1000 );
   temp1 = temp2 % 1000;

   printf( "  %d ", temp1 / 100 );
   temp2 = temp1 % 100;

   printf( "  %d ", temp2 / 10 );
   temp1 = temp2 % 10;

   printf( "  %d\n", temp1 ); /* print right-most digit */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a five-digit number: 23456
2   3   4   5   6
```

2.31 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
```

ANS:

```c
/* Exercise 2.31 Solution */
#include <stdio.h>

int main()
{
```

```
   int count = 0; /* initialize count to zero */

   /* calculate the square and cube for the numbers 0 to 10 */
   printf( "\nnumber\tsquare\tcube\n" );
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1; /* increment count by 1 */
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   count = count + 1;
   printf( "%d\t%d\t%d\n", count, count * count,
      count * count * count );

   return 0; /* indicate successful termination */

} /* end main */
```

3

Structured Program Development in C: Solutions

SOLUTIONS

3.11 Identify and correct the errors in each of the following [*Note*: There may be more than one error in each piece of code]:

a)
```
if ( age >= 65 );
   printf( "Age is greater than or equal to 65\n" );
else
   printf( "Age is less than 65\n" );
```

ANS:
```
if ( age >= 65 ) /* ; removed */
   printf( “Age is greater than or equal to 65\n” );
else
   printf( “Age is less than 65\n” );
```

b)
```
int x = 1, total;

while ( x <= 10 ) {
   total += x;
   ++x;
}
```

ANS:
```
int x = 1, total = 0;
while ( x <= 10 ) {
   total += x;
   ++x;
}
```

c)
```
While ( x <= 100 )
   total += x;
   ++x;
```

ANS:
```
while ( x <= 100 ) {
   total += x;
   ++x;
}
```

d)
```
while ( y > 0 ) {
   printf( "%d\n", y );
   ++y;
}
```

ANS:
```
while ( y > 0 ) {
   printf( "%d\n", y );
   --y;
}
```

3.12 Fill in the blanks in each of the following:

a) The solution to any problem involves performing a series of actions in a specific __________.
ANS: order.
b) A synonym for procedure is __________.
ANS: algorithm
c) A variable that accumulates the sum of several numbers is a __________.
ANS: total.
d) The process of setting certain variables to specific values at the beginning of a program is called __________.
ANS: initialization.
e) A special value used to indicate "end of data entry" is called a __________, a __________, a __________ or a __________ value.
ANS: sentinel value, dummy value, signal value, flag value.
f) A __________ is a graphical representation of an algorithm.
ANS: flowchart.
g) In a flowchart, the order in which the steps should be performed is indicated by __________symbols.
ANS: arrow (flowline).
h) The termination symbol indicates the __________ and __________ of every algorithm.
ANS: beginning, end.
i) Rectangle symbols correspond to calculations that are normally performed by __________ statements and input/output operations that are normally performed by calls to the__________ and __________ standard library functions.
ANS: assignment, `printf`, `scanf`.
j) The item written inside a decision symbol is called a __________.
ANS: condition.

3.13 What does the following program print?

```
#include <stdio.h>

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

   while ( x <= 10 ) {
      y = x * x;
      printf( "%d\n", y );
      total += y;
      ++x;
   }

   printf("Total is %d\n", total);

   return 0;
}
```

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

3.14 Write a single pseudocode statement that indicates each of the following:

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

ANS: *print "enter two numbers"*

b) Assign the sum of variables `x`, `y`, and `z` to variable `p`.

ANS: *p = x + y + z*

c) The following condition is to be tested in an `if...else` selection statement: The current value of variable `m` is greater than twice the current value of variable `v`.

ANS: *if m is greater than twice v*
do this ...
else
do this ...

d) Obtain values for variables `s`, `r`, and `t` from the keyboard.

ANS: *input `s`, input `r`, input `t`*

3.15 Formulate a pseudocode algorithm for each of the following:

a) Obtain two numbers from the keyboard, compute the sum of the numbers and display the result.

ANS:

Input the first number
Input the second number
Add the two numbers
Output the sum

b) Obtain two numbers from the keyboard, and determine and display which (if either) is the larger of the two numbers.

ANS:

Input the first number from the keyboard
Input the second number from the keyboard
If the first number is greater than the second number
print it
Else if the second number is greater than the first number
print it
Else
print a message stating that the numbers are equal

c) Obtain a series of positive numbers from the keyboard, and determine and display the sum of the numbers. Assume that the user types the sentinel value `-1` to indicate "end of data entry."

ANS:

Input a value from the keyboard
While the input value is not equal to `-1`
add the number to the running total
input the next number
Print the sum

3.16 State which of the following are *true* and which are *false*. If a statement is *false*, explain why.

a) Experience has shown that the most difficult part of solving a problem on a computer is producing a working C program.

ANS: False. The algorithm is the hardest of solving a problem.
b) A sentinel value must be a value that cannot be confused with a legitimate data value.
ANS: True.
c) Flowlines indicate the actions to be performed.
ANS: False. Flowlines indicate the order in which steps are performed.
d) Conditions written inside decision symbols always contain arithmetic operators (i.e., +, -, *, /, and %).
ANS: False. They normally contain conditional operators.
e) In top-down, stepwise refinement, each refinement is a complete representation of the algorithm.
ANS: True.

For Exercises 3.17 to 3.21, perform each of these steps:

1. Read the problem statement.
2. Formulate the algorithm using pseudocode and top-down, stepwise refinement.
3. Write a C program.
4. Test, debug, and execute the C program.

3.17 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 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. Here is a sample input/output dialog:.

```
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:

2) ***Top:***
Determine the average miles/gallon for each tank of gas, and the overall miles/gallon 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 the 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 the next tank

Set totalAverage to totalMiles divided by totalGallons.
print the average miles/gallon

3)

```c
/* Exercise 3.17 Solution */
#include <stdio.h>

int main()
{
   double gallons;            /* gallons used for current tank*/
   double miles;              /* miles driven for current tank*/
   double totalGallons = 0.0; /* total gallons used */
   double totalMiles = 0.0;   /* total miles driven */
   double totalAverage;       /* average miles/gallon */

   /* get gallons used for first tank */
   printf( "Enter the gallons used ( -1 to end): " );
   scanf( "%lf", &gallons );

   /* loop until sentinel value read from user */
   while ( gallons != -1.0 ) {
      totalGallons += gallons; /* add current tank gallons to total */

      printf( "Enter the miles driven: " ); /* get miles driven */
      scanf( "%lf", &miles );
      totalMiles += miles; /* add current tank miles to total */

      /* display miles per gallon for current tank */
      printf( "The Miles / Gallon for this tank was %f\n\n",
         miles / gallons );

      /* get next tank's gallons */
      printf( "Enter the gallons used ( -1 to end ): " );
      scanf( "%lf", &gallons );
   } /* end while */

   /* calculate average miles per gallon over all tanks */
   totalAverage = totalMiles / totalGallons;
   printf( "\nThe overall average Miles/Gallon was %f\n", totalAverage );

   return 0; /* indicate successful termination */

} /* end main */
```

3.18 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:

1. Account number
2. Balance at the beginning of the month
3. Total of all items charged by this customer this month
4. Total of all credits applied to this customer's account this month
5. 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." Here is a sample input/output dialog:

```
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:

2) ***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, calcu late 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.

3)

```
/* Exercise 3.18 Solution */
#include <stdio.h>

int main()
{
   int accountNumber; /* current account's number */
   double balance;    /* current account's starting balance */
   double charges;    /* current account's total charges */
   double credits;    /* current account's total credits */
   double limit;      /* current account's credit limit */

   /* get account number */
   printf( "\nEnter account number ( -1 to end): " );
   scanf( "%d", &accountNumber );

   /* loop until sentinel value read from user */
   while ( accountNumber != -1 ) {
      printf( "Enter beginning balance: " );
      scanf( "%lf", &balance );

      printf( "Enter total charges: " );
      scanf( "%lf", &charges );

      printf( "Enter total credits: " );
      scanf( "%lf", &credits );

      printf( "Enter credit limit: " );
      scanf( "%lf", &limit );

      balance += charges - credits; /* calculate balance */

      /* if balance is over limit, display account number
         with credit limit and balance to two digits of precision */
      if ( balance > limit ) {
         printf( "%s%d\n%s%.2f\n%s%.2f\n%s\n",
            "Account:      ", accountNumber, "Credit limit: ", limit,
            "Balance:      ", balance, "Credit Limit Exceeded." );
      } /* end if */

      /* prompt for next account */
      printf( "\nEnter account number ( -1 to end ): " );
      scanf( "%d", &accountNumber );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

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

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

Enter sales in dollars ( -1 to end ): 1234.56
Salary is: $311.11

Enter sales in dollars ( -1 to end ): 1088.89
Salary is: $298.00

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

ANS:

2) ***Top:***

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

First refinement:

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

3)

```
/* Exercise 3.19 Solution */
#include <stdio.h>

int main()
{
   double sales; /* gross weekly sales */
   double wage;  /* commissioned earnings */

   /* get first sales */
   printf( "Enter sales in dollars ( -1 to end): " );
   scanf( "%lf", &sales );

   /* loop until sentinel value read from user */
   while ( sales != -1.0 ) {
      wage = 200.0 + 0.09 * sales; /* calculate wage */

      /* display salary */
      printf( "Salary is: $%.2f\n\n", wage );

      /* prompt for next sales */
      printf( "Enter sales in dollars ( -1 to end ): " );
      scanf( "%lf", &sales );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.20 The simple interest on a loan is calculated by the formula

```
interest = principal * rate * days / 365;
```

The preceding formula assumes that `rate` is the annual interest rate, and therefore includes the division by 365 (days). Develop a program that will input `principal`, `rate` and `days` for several loans, and will calculate and display the simple interest for each loan, using the preceding formula. Here is a sample input/output dialog:

```
Enter loan principal ( -1 to end): 1000.00
Enter interest rate: .1
Enter term of the loan in days: 365
The interest charge is $100.00

Enter loan principal ( -1 to end ): 1000.00
Enter interest rate: .08375
Enter term of the loan in days: 224
The interest charge is $51.40

Enter loan principal ( -1 to end ): 10000.00
Enter interest rate: .09
Enter term of the loan in days: 1460
The interest charge is $3600.00

Enter loan principal ( -1 to end ): -1
```

ANS:

2) ***Top:***

For an arbitrary number of loans determine the simple interest for each loan.

First refinement:

Input the principal of the loan, the interest rate, and the term of the loan, calculate and print the simple interest for the loan, and process the next loan.

Second refinement:

input the first loan principal in dollars.
While the sentinel value (-1) has not been entered for the loan principal
  Input the interest rate
  Input the term of the loan in days
  Calculate the simple interest for the loan
  Print the simple interest for the loan
  Input the loan principal for the next loan

3)

```
/* Exercise 3.20 Solution */
#include <stdio.h>

int main()
{
   double principal; /* loan principal */
   double rate;      /* interest rate */
   double interest;  /* interest charge */
   int term;         /* length of loan in days */

   /* get loan principal */
   printf( "Enter loan principal ( -1 to end): " );
   scanf( "%lf", &principal );
```

```c

   /* loop until sentinel value is read from user */
   while ( principal != -1.0 ) {
      printf( "Enter interest rate: " ); /* get rate */
      scanf( "%lf", &rate );

      printf( "Enter term of the loan in days: " ); /* get term */
      scanf( "%d", &term );

      /* calculate interest charge */
      interest = principal * rate * term / 365.0;
      printf( "The interest charge is $%.2f\n\n", interest );

      /* get next loan principal */
      printf( "Enter loan principal ( -1 to end ): " );
      scanf( "%lf", &principal );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.21 Develop a 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. Here is a sample input/output dialog:

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

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

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

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

ANS:

2) ***Top:***

For an arbitrary number of employees, determine the gross pay for each employee.

First refinement:

Input the number of hours worked for the employee, enter the employee's hourly wage, calculate and print the employee's gross pay, and process the next employee.

Second refinement:

Input the first employee's number of hours worked.
While the sentinel value (-1) has not been entered for the hours worked
Input the employee's hourly wage
Calculate the employee's gross pay with overtime for hours over 40
Print the employee's gross pay
Input the number of hours worked for the next computer

3)

```
/* Exercise 3.21 Solution */
#include <stdio.h>

int main( void )
{
   double hours;  /* total hours worked */
   double rate;   /* hourly pay rate */
   double salary; /* gross pay */

   /* get first employee's hours worked */
   printf( "Enter number of hours worked ( -1 to end ): " );
   scanf( "%lf", &hours );

   /* loop until sentinel value read from user */
   while ( hours != -1.0 ) {

      /* get hourly rate */
      printf( "Enter hourly rate of the worker ( $00.00 ): " );
      scanf( "%lf", &rate );

      /* if employee worked less than 40 hours */
      if ( hours <= 40 ) {
         salary = hours * rate;
      } /* end if */
      else { /* compute "time-and-a-half" pay */
         salary = 40.0 * rate + ( hours - 40.0 ) * rate * 1.5;
      } /* end else */

      /* display gross pay */
      printf( "Salary is $%.2lf\n\n", salary );

      /* prompt for next employee's data */
      printf( "Enter number of hours worked ( -1 to end ): " );
      scanf( "%lf", &hours );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.22 Write a program that demonstrates the difference between predecrementing and postdecrementing using the decrement operator --.

ANS:

```
/* Exercise 3.22 Solution */
#include <stdio.h>

int main()
{
   int c; /* define c to use decrement operator */

   c = 5;
   printf( "%d\n", c );
   printf( "%d\n", --c ); /* predecrement */
   printf( "%d\n\n", c );

```

```
   c = 5;
   printf( "%d\n", c );
   printf( "%d\n", c-- ); /* postdecrement */
   printf( "%d\n\n", c );

   return 0; /* indicate successful termination */

} /* end main */
```

```
5
4
4

5
5
4
```

3.23 Write a program that utilizes looping to print the numbers from 1 to 10 side-by-side on the same line with 3 spaces between each number.

ANS:

```
/* Exercise 3.23 Solution */
#include <stdio.h>

int main()
{
   int i = 0; /* initialize i */

   /* loop while i is less than 11 */
   while ( ++i < 11 ) {
      printf( "%d   ", i );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

```
1   2   3   4   5   6   7   8   9   10
```

3.24 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 and then a 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

ANS:

Input the first number directly into the variable `largest`
Increment `counter` to 2

While `counter` is less than or equal to 10
input a new variable into the variable `number`
If `number` is greater than `largest`
replace `largest` with `number`
Increment `counter`
Print the value of the largest (`while` condition false when `counter` is 11)

```c
/* Exercise 3.24 Solution */
#include <stdio.h>

int main()
{
   int counter; /* counter for 10 repetitions */
   int number;  /* current number input */
   int largest; /* largest number found so far */

   /* get first number */
   printf( "Enter the first number: " );
   scanf( "%d", &largest );
   counter = 2;

   /* loop 9 more times */
   while ( counter <= 10 ) {
      printf( "Enter next number: " ); /* get next number */
      scanf( "%d", &number );

      /* if current number input is greater than largest number,
         update largest */
      if ( number > largest ) {
         largest = number;
      } /* end if */

      counter++;
   } /* end while */

   printf( "Largest is %d\n", largest ); /* display largest number */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the first number: 7
Enter next number: 37
Enter next number: 78
Enter next number: 2
Enter next number: 437
Enter next number: 72
Enter next number: 1
Enter next number: 4
Enter next number: 36
Enter next number: 100
Largest is 437
```

3.25 Write a program that utilizes looping 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
6	60	600	6000
7	70	700	7000
8	80	800	8000
9	90	900	9000
10	100	1000	10000

The tab character, `\t`, may be used in the `printf` statement to separate the columns with tabs.

ANS:

```c
/* Exercise 3.25 Solution */
#include <stdio.h>

int main()
{
   int n = 0; /* counter */

   /* display table headers */
   printf( "\tN\t\t10 * N\t\t100 * N\t\t1000 * N\n\n" );

   /* loop 10 times */
   while ( ++n <= 10 ) {

      /* calculate and display table values */
      printf( "\t%-2d\t\t%-5d\t\t%-5d\t\t%-6d\n",
              n, 10 * n, 100 * n, 1000 * n );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.26 Write a program that utilizes looping to produce the following table of values:

A	A+2	A+4	A+6
3	5	7	9
6	8	10	12
9	11	13	15
12	14	16	18
15	17	19	21

ANS:

```
/* Exercise 3.26 Solution */
#include <stdio.h>

int main()
{
   int a = 3; /* counter */

   /* display table headers */
   printf( "A\tA+2\tA+4\tA+6\n\n" );

   /* loop 5 times */
   while ( a <= 15 ) {

      /* calculate and display table values */
      printf( "%d\t%d\t%d\t%d\n", a, a + 2, a + 4, a + 6 );
      a += 3;
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.27 Using an approach similar to Exercise 3.24, find the *two* largest values of the 10 numbers. [*Note*: You may input each number only once.]

ANS:

```
/* Exercise 3.27 Solution */
#include <stdio.h>

int main()

{
   int counter;           /* counter for 10 repetitions */
   int number;            /* current number input */
   int largest;           /* largest number found */
   int secondLargest = 0; /* second largest number found */

   printf( "Enter the first number: " ); /* get first number */
   scanf( "%d", &largest );
   counter = 2;

   /* loop 9 more times */
   while ( counter <= 10 ) {
      printf( "Enter next number: " ); /* prompt for next number */
      scanf( "%d", &number );

      /* if current number is greater than largest */
      if ( number > largest ) {

         /* update second largest with previous largest */
         secondLargest = largest;

         /* update largest with current number */
         largest = number;
      } /* end if */
      else {

```

```
         /* if number is between secondLargest and largest */
         if ( number > secondLargest ) {
            secondLargest = number;
         } /* end if */

      } /* end else */

      ++counter;
   } /* end while */

   /* display largest two numbers */
   printf( "Largest is %d\n", largest );
   printf( "Second largest is %d\n", secondLargest );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the first number: 100
Enter next number: 102
Enter next number: 83
Enter next number: 3883
Enter next number: 328
Enter next number: 28
Enter next number: 839
Enter next number: 2398
Enter next number: 182
Enter next number: 0
Largest is 3883
Second largest is 2398
```

3.28 Modify the program in Figure 3.10 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.

ANS:

```
/* Exercise 3.28 Solution */
#include <stdio.h>

int main()
{
   int passes = 0;   /* number of passes */
   int failures = 0; /* number of failures */
   int student = 1;  /* student counter */
   int result;       /* one exam result */

   /* process 10 students using counter-controlled loop */
   while ( student <= 10 ) {

      /* prompt user for input and obtain value from user */
      printf( "Enter result ( 1=pass, 2=fail ): " );
      scanf( "%d", &result );

      /* loop until valid input */
      while ( result != 1 && result != 2 ) {
         printf( "Invalid result\nEnter result ( 1=pass, 2=fail ): " );
         scanf( "%d", &result );
      } /* end inner while */

```

```
      /* if result 1, increment passes */
      if ( result == 1 ) {
         ++passes;
      } /* end if */
      else { /* if result is not 1, increment failures */
         ++failures;
      } /* end else */

      ++student;
   } /* end while */

   printf( "Passed %d\nFailed %d\n", passes, failures );

   /* if more than eight students passed, print "raise tuition" */
   if ( passes >= 8 ) {
      printf( "Raise tuition\n" );
   } /* end if */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter result ( 1=pass, 2=fail ): 1
Enter result ( 1=pass, 2=fail ): 2
Enter result ( 1=pass, 2=fail ): 3
Invalid result
Enter result ( 1=pass, 2=fail ): 4
Invalid result
Enter result ( 1=pass, 2=fail ): 2
Enter result ( 1=pass, 2=fail ): 2
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 ): 1
Enter result ( 1=pass, 2=fail ): 1
Passed 6
Failed 4
```

3.29 What does the following program print?

```
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int count = 1; /* initialize count */

   while ( count <= 10 ) { /* loop 10 times */

      /* output line of text */
      printf( "%s\n", count % 2 ? "****" : "++++++++" );
      count++; /* increment count */
   } /* end while */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

ANS:

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

3.30 What does the following program print?

```c
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int row = 10; /* initialize row */
   int column;   /* define column */

   while ( row >= 1 ) { /* loop until row < 1 */
      column = 1;       /* set column to 1 as iteration begins */

      while ( column <= 10 ) {               /* loop 10 times */
         printf( "%s", row % 2 ? "<": ">" ); /* output */
         column++;                           /* increment column */
      } /* end inner while */

      row--;           /* decrement row */
      printf( "\n" ); /* begin new output line */
   } /* end outer while */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

ANS:

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

3.31 *(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 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 )
printf( "*****\n" );
else
printf( "#####\n" );
printf( "$$$$$\n" );
```
ANS:
`x = 9, y = 11`

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

`x = 11, y = 9`

```
$$$$$
```

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

```
*****
```

`x = 11, y = 9`

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

3.32 *(Another Dangling Else Problem)* Modify the following code to produce the output shown. Use proper indentation techniques. You might not make any changes other than inserting braces. The compiler ignores the indentation in a 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 )
printf( "@@@@@\n" );
else
printf( "#####\n" );
printf( "$$$$$\n" );
printf( "&&&&&\n" );
```

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

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

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

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

```
@@@@@
```

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

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

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

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

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

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

ANS:

```
if ( y == 8 ) {
   if ( x == 5 )
      printf( “@@@@@\n” );
}
else {
   printf( “#####\n” );
   printf( “$$$$$\n” );
   printf( “&&&&&\n” );
}
```

3.33 Write a program that reads in the side of a square and then prints that square out of asterisks. Your program should work for squares of all side sizes between 1 and 20. For example, if your program reads a size of 4, it should print

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

ANS:

```
/* Exercise 3.33 Solution */
#include<stdio.h>

int main()
{
   int side;     /* side counter */
   int temp;     /* temporary integer */
   int asterisk; /* asterisk counter */

   printf( "Enter the square side: " ); /* get size of square */
   scanf( "%d", &side );

   temp = side;

   /* loop through rows of square */
   while ( side-- > 0 ) {
      asterisk = temp;

      /* loop through columns of square */
      while ( asterisk-- > 0 ) {
         printf( "*" );
      } /* end inner while */

      putchar( '\n' );
   } /* end outer while */

  return 0; /* indicate successful termination */

} /* end main */
```

3.34 Modify the program you wrote in Exercise 3.33 so that it prints a hollow square. For example, if your program reads a size of 5, it should print

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

ANS:

```
/* Exercise 3.34 Solution */
#include<stdio.h>

int main()
{
   int side;        /* side counter */
   int rowPosition; /* row counter */
   int size;        /* length of side */

   printf( "Enter the square side: " ); /* prompt for side length */
   scanf( "%d", &side );

   size = side; /* set size counter to length of side */

   /* loop side number of times */
   while ( side > 0 ) {
      rowPosition = size; /* set row counter to length of size */

      /* loop rowPosition number of times */
      while ( rowPosition > 0 ) {

         /* if side or row counter is 1 or size print an '*' */
         if ( size == side ) {
            printf( "*" );
         } /* end if */
         else if ( side == 1 ) {
            printf( "*" );
         } /* end else if */
         else if ( rowPosition == 1 ) {
            printf( "*" );
         } /* end else if */
         else if ( rowPosition == size ) {
            printf( "*" );
         } /* end else if */
         else { /* otherwise, print a space */
            printf( " " );
         } /* end else */

         --rowPosition; /* decrement row counter */
      } /* end inner while */

      printf( "\n" ); /* new line for next row */
      --side; /* decrement side counter */
   } /* end outer while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.35 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 are palindromes: 12321, 55555, 45554 and 11611. Write a program that reads in a five-digit integer and determines whether or not it is a palindrome. [*Hint*: Use the division and remainder operators to separate the number into its individual digits.]

ANS:

```
/* Exercise 3.35 Solution */
#include<stdio.h>

int main()
{
   int number;      /* input number */
   int temp1;       /* first temporary integer */
   int temp2;       /* second temporary integer */
   int firstDigit;  /* first digit of input */
   int secondDigit; /* second digit of input */
   int fourthDigit; /* fourth digit of input */
   int fifthDigit;  /* fifth digit of input */

   printf( "Enter a five-digit number: " ); /* get number */
   scanf( "%d", &number );

   temp1 = number;

   /* determine first digit by integer division by 10000 */
   firstDigit = temp1 / 10000;
   temp2 = temp1 % 10000;

   /* determine second digit by integer division by 1000 */
   secondDigit = temp2 / 1000;
   temp1 = temp2 % 1000;

   temp2 = temp1 % 100;

   /* determine fourth digit by integer division by 10 */
   fourthDigit = temp2 / 10;
   temp1 = temp2 % 10;

   fifthDigit = temp1;

   /* if first and fifth digits are equal */
   if ( firstDigit == fifthDigit ) {

      /* if second and fourth digits are equal */
      if ( secondDigit == fourthDigit ) {

         /* number is a palindrome */
         printf( "%d is a palindrome\n", number );
      } /* end if */
      else { /* number is not a palindrome */
         printf( "%d is not a palindrome\n", number );
      } /* end else */

   } /* end if */
   else { /* number is not a palindrome */
      printf( "%d is not a palindrome\n", number );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a five-digit number: 18181
18181 is a palindrome
```

```
Enter a five-digit number: 16738
16738 is not a palindrome
```

3.36 Input an integer containing only 0s and 1s (i.e., a “binary” integer) and print its decimal equivalent. [*Hint*: Use the remainder 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 in which 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.]

ANS:

```
/* Exercise 3.36 Solution */
#include<stdio.h>

int main()
{
   int binary;         /* current value of binary number */
   int number;         /* input binary number */
   int decimal = 0;    /* current value of decimal number */
   int highBit = 16;   /* value of highest bit */
   int factor = 10000; /* factor of 10 to pick off digits */

   /* prompt for binary input */
   printf( "Enter a binary number ( 5 digits maximum ): " );
   scanf( "%d", &binary );

   number = binary; /* save in number for final display */

   /* loop 5 times using powers of 2 */
   while ( highBit >= 1 ) {

      /* update decimal value with decimal value corresponding
         to current highest binary bit */
      decimal += binary / factor * highBit;

      /* reduce high bit by factor of 2, i.e.,
         move one bit to the right */
      highBit /= 2;

      /* reduce binary number to eliminate current highest bit */
      binary %= factor;

      /* reduce factor by power of 10, i.e.,
         move one bit to the right */
      factor /= 10;
   } /* end while */

   /* display decimal value */
   printf( "The decimal equivalent of %d is %d\n", number, decimal );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a binary number ( 5 digits maximum ): 10111
The decimal equivalent of 10111 is 23
```

```
Enter a binary number ( 5 digits maximum ): 1101
The decimal equivalent of 1101 is 13
```

3.37 How can you determine how fast your own computer really operates? Write a program with a `while` loop that counts from 1 to 300,000,000 by 1s. Every time the count reaches a multiple of 100,000,000 print that number on the screen. Use your watch to time how long each million repetitions of the loop takes.

ANS:

```c
/* Exercise 3.37 Solution */
#include<stdio.h>

int main()
{
   long int count = 1; /* counter */

   /* loop to 300,000,000 */
   while( count <= 300000000 ) {

      if ( count % 100000000 == 0 ) {
         printf( "Multiple is %d\n", count / 100000000 );
      } /* end if */

      ++count; /* increment count */
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Multiple is 1
Multiple is 2
Multiple is 3
```

3.38 Write a program that prints 100 asterisks, one at a time. After every tenth asterisk, your program should print a newline character. (Hint: Count from 1 to 100. Use the remainder operator to recognize each time the counter reaches a multiple of 10.)

ANS:

```c
/* Exercise 3.38 Solution */
#include <stdio.h>

int main()
{
  int count = 0; /* counter */

  /* loop to 100 */
  while( ++count <= 100 )

     /* print a new line after every 10th asterisk */
     count % 10 == 0 ? printf( "*\n" ) : printf( "*" );

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

```c

} /* end main */
```

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

3.39 Write a program that reads an integer and determines and prints how many digits in the integer are 7s.
ANS:

```c
/* Exercise 3.39 Solution */
#include <stdio.h>

int main()
{
   int number;         /* user input */
   int numCopy;        /* copy of number */
   int factor = 10000; /* set factor to pick off digits */
   int digit;          /* individual digit of number */
   int sevens = 0;     /* sevens counter */

   printf( "Enter a 5-digit number: " ); /* get number from user */
   scanf( "%d", &number );

   numCopy = number;

   /* loop through each of the 5 digits */
   while ( factor >= 1 ) {
      digit = numCopy / factor; /* pick off next digit */

      if ( digit == 7 ) { /* if digit equals 7, increment sevens */
         ++sevens;
      } /* end if */

      numCopy %= factor;
      factor /= 10;
   } /* end while */

   /* output number of sevens */
   printf( "The number %ld has %d seven(s) in it\n", number, sevens );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a 5-digit number: 17737
The number 17737 has 3 seven(s) in it
```

```
Enter a 5-digit number: 11727
The number 11727 has 2 seven(s) in it
```

3.40 Write a program that displays the following checkerboard pattern

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

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

```
printf( "* " );
printf( " " );
printf( "\n" );
```

ANS:

```c
/* Exercise 3.40 Solution */
#include <stdio.h>

int main()
{
   int side = 8; /* side counter */
   int row;      /* row counter */
   int mod;      /* remainder */

   /* loop 8 times */
   while ( side >= 1 ) {
      row = 8; /* reset row counter */
      mod = side % 2;

      /* loop 8 times */
      while ( row >= 1 ) {

         /* if odd row, begin with a space */
         if ( mod != 0 ) {
            printf( " " );
            mod = 0;
         } /* end if */

         printf( "* " );
         --row;
      } /* end while */

      printf( "\n" ); /* go to next line */
      --side;
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

3.41 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?

ANS: Program execution terminates when largest integer is exceeded (i.e., the loop continuation test fails when the maximum value for an integer is exceeded. On a 4-byte system, the largest integer value is 2147483647 and anything above that is represented by a negative number, which fails the loop continuation test).

```c
/* Exercise 3.41 Solution */
#include <stdio.h>

int main()
{
   int multiple = 1; /* counter */

   /* infinite loop */
   while ( multiple > 0 ) {

      /* calculate the next power of two */
      multiple *= 2;
      printf( "%d\n", multiple );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
```

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

ANS:

```
/* Exercise 3.42 Solution */
#include<stdio.h>

int main()
{
   float radius;       /* input radius */
   float pi = 3.14159; /* value for pi */

   printf( "Enter the radius: "); /* get radius value */
   scanf( "%f", &radius );

   /* compute and display diameter */
   printf( "The diameter is %.2f\n", radius * 2 );

   /* compute and display circumference */
   printf( "The circumference is %.2f\n", 2 * pi * radius );

   /* compute and display area */
   printf( "The area is %.2f\n", pi * radius * radius );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the radius: 4.7
The diameter is 9.40
The circumference is 29.53
The area is 69.40
```

3.43 What is wrong with the following statement? Rewrite the statement to accomplish what the programmer was probably trying to do.

```
printf( "%d", ++( x + y ) );
```

ANS: `printf( “%d”, 1 + x + y );`

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

ANS:

```
/* Exercise 3.44 Solution */
#include <stdio.h>

int main()
{
   double a; /* first number */
   double b; /* second number */
   double c; /* third number */

   /* input 3 numbers */
   printf( "Enter three doubleing point numbers: " );
   scanf( "%lf%lf%lf", &a, &b, &c);

   /* use Pythagorean Theorem */
   if ( c * c == a * a + b * b ) {
      printf( "The three numbers could be sides of a triangle\n" );
   } /* end if */
```

```
   else {
      printf( "The three numbers probably");
      printf( " are not the sides of a triangle\n" );
   } /* end if */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three doubleing point numbers: 5.7 3.6 2.2
The three numbers probably are not the sides of a triangle
```

```
Enter three doubleing point numbers: 3.0 4.0 5.0
The three numbers could be sides of a triangle
```

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

```
/* Exercise 3.45 Solution */
#include <stdio.h>

int main()
{
   int a; /* first number */
   int b; /* second number */
   int c; /* third number */

   /* input three numbers */
   printf( "Enter three integers: ");
   scanf( "%d%d%d", &a, &b, &c );

   /* use Pythagorean Theorem */
   if ( c * c == a * a + b * b ) {
      printf( "The three integers are the sides of");
      printf( " a right triangle\n" );
   } /* end if */
   else {
      printf( "The three integers are not the sides");
      printf( " of a right triangle\n" );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

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

```
Enter three integers: 9 4 1
The three integers are not the sides of a right triangle
```

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

ANS:

```c
/* Exercise 3.46 Part A solution */
#include <stdio.h>

int main()
{
   int first;  /* first digit replacement */
   int second; /* second digit replacement */
   int third;  /* third digit replacement */
   int fourth; /* fourth digit replacement */
   int digit;  /* input number */
   int temp1;  /* temporarily hold digit */
   int temp2;  /* temporarily hold digit */
   int encryptedNumber; /* resulting encrypted number */

   /* prompt for input */
   printf( "Enter a four digit number to be encrypted: " );
   scanf( "%d", &digit );

   temp1 = digit;

   /* retrieve each digit and replace with
      (sum of digit and 7) mod 10 */
   first = ( temp1 / 1000 + 7 ) % 10;
   temp2 = temp1 % 1000;

   second = ( temp2 / 100 + 7 ) % 10;
   temp1 = temp2 % 100;

   third = ( temp1 / 10 + 7 ) % 10;
   temp2 = temp1 % 10;

   fourth = ( temp2 + 7 ) % 10;

   /* swap first and third */
   temp1 = first;
   first = third * 1000; /* multiply by 1000 for 1st digit component */
   third = temp1 * 10; /* multiply by 10 for 3rd digit component */

   /* swap second and fourth */
   temp1 = second;
   second = fourth * 100; /* multiply by 100 for 2nd digit component */
   fourth = temp1 * 1;

   /* add components to obtain encrypted number */
   encryptedNumber = first + second + third + fourth;

   /* display encrypted number */
   printf( "Encrypted number is %d\n", encryptedNumber );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a four digit number to be encrypted: 5678
Encrypted number is 4523
```

```c
/* Exercise 3.46 Part B Solution */
#include <stdio.h>

int main()
{
   int first;     /* first decrypted digit */
   int second;    /* second decrypted digit */
   int third;     /* third decrypted digit */
   int fourth;    /* fourth decrypted digit */
   int decrypted; /* decrypted number */
   int temp1;     /* temporarily hold digit */
   int temp2;     /* temporarily hold digit */
   int encryptedNumber; /* input number */

   /* prompt for input */
   printf( "Enter a four digit encrypted number: " );
   scanf( "%d", &encryptedNumber );

   temp1 = encryptedNumber;

   /* retrieve each digit and decrypt by
      (sum of digit and 3) mod 10 */
   first = ( temp1 / 1000 );
   temp2 = temp1 % 1000;

   second = ( temp2 / 100 );
   temp1 = temp2 % 100;

   third = ( temp1 / 10 );
   temp2 = temp1 % 10;

   fourth = temp2;

   temp1 = ( first + 3 ) % 10;
   first = ( third + 3 ) % 10;
   third = temp1;

   temp1 = ( second + 3 ) % 10;
   second = ( fourth + 3 ) % 10;
   fourth = temp1;

   /* add components to obtain decrypted number */
   decrypted = ( first * 1000 ) + ( second * 100 ) +
               ( third * 10 ) + fourth;

   /* display decrypted number */
   printf( "Decrypted number is %d\n", decrypted );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a four digit encrypted number: 4523
Decrypted number is 5678
```

1.1 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.

ANS:

```c
/* Exercise 3.47 Part A Solution */
#include <stdio.h>

int main()
{
   int n;                    /* current multiplication factor */
   int number = -1;          /* input number */
   unsigned factorial = 1; /* resulting factorial */

   /* loop until valid input */
   do {
      printf( "Enter a positive integer: " );
      scanf( "%d", &number );
   } while ( number < 0 ); /* end do...while */

   n = number;

   /* compute factorial */
   while( n >= 0 ) {

      if ( n == 0 ) {
         factorial *= 1;
      } /* end if */
      else {
         factorial *= n;
      } /* end else */

      --n;
   } /* end while */

   /* display factorial */
   printf( "%d! is %u\n", number, factorial );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a positive integer: 5
5! is 120
```

```
Enter a positive integer: 9
9! is 362880
```

```
Enter a positive integer: -8
Enter a positive integer: 0
0! is 1
```

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

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

ANS:

```
/* Exercise 3.47 Part B Solution */
#include <stdio.h>

int main()
{
   int n = 0;          /* loop counter for accuracy */
   int fact = 1;       /* current n factorial */
   int accuracy = 10; /* degree of accuracy */
   double e = 0;       /* current estimated value of e */

   /* loop until degree of accuracy */
   while( n <= accuracy ) {

      if ( n == 0 ) {
         fact *= 1;
      } /* end if */
      else {
         fact *= n;
      } /* end else */

      e += 1.0 / fact;
      ++n;
   } /* end while */

   printf( "e is %f\n", e ); /* display estimated value */

   return 0; /* indicate successful termination */

} /* end main */
```

```
e is 2.718282
```

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

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

ANS:

```
/* Exercise 3.47 Part C Solution */
#include <stdio.h>

int main()
{
   int n = 0;          /* counter */
   int accuracy = 15; /* degree of accuracy */
   int x = 3;          /* exponent */
   int times = 0;      /* counter */
   int count;          /* copy of n */
   double e = 1.0;     /* e raised to the x power */
   double exp = 0.0;  /* x raised to the n power */
   double fact = 1.0; /* n factorial */

```

```
   /* loop while less than degree of accuracy */
   while( n <= accuracy ) {
      count = n;

      /* update n! */
      if ( n == 0 ) {
         fact *= 1.0;
      } /* end if */
      else {
         fact *= n;
      } /* end else */

      while ( times < count ) {

         /* calculate x raised to the n power */
         if ( times == 0 ) {
            exp = 1.0;
            exp *= x;
         } /* end if */
         else {
            exp *= x;
         } /* end else */

         ++times;
      } /* end while */

      e += exp / fact; /* update e raised to the x power */
      ++n;
   } /* end while */

   /* display result */
   printf( "e raised to the %d power is %f\n", x, e );

   return 0; /* indicate successful termination */

} /* end main */
```

```
e raised to the 3 power is 20.085534
```

4

C Program Control: Solutions

SOLUTIONS

4.5 Find the error in each of the following (Note: there may be more than one error):

a)
```
For ( x = 100, x >= 1, x++ )
   printf( "%d\n", x );
```

ANS: `F` in `for` should be lowercase. The infinite loop can be corrected by switching the `1` and the `100` and changing the relational operator to <=. Semicolons are needed between the `for` conditions, not comma operators.

```
for ( x = 1; x <= 100; x++ )
   printf( "%d\n", x);
```

b) The following code should print whether a given integer is odd or even:

```
switch ( value % 2 ) {
   case 0:
     printf( "Even integer\n" );
   case 1:
     printf( "Odd integer\n" );
}
```

ANS: A `break` is needed for `case 0`, otherwise both statements will be printed out.

```
switch ( value % 2 ) {
    case 0:
        printf( "Even integer\n" );
        break;
    case 1:
        printf( "Odd integer\n" );
```

c) The following code should input an integer and a character and print them. Assume the user types as input `100 A`.

```
scanf( "%d", &intVal );
charVal = getchar();
printf( "Integer: %d\nCharacter: %c\n", intVal, charVal );
```

ANS: `charVal` will read the return character when the user types in `intVal` and hits return. To correct this, `scanf` should be used to read in `charVal`.

```
scanf( "%d", &intVal );
scanf( "\n%c", &charVal );
printf( "Integer: %d\nCharacter: %c\n", intVal, charVal );
```

d) `for ( x = .000001; x <= .0001; x += .000001 )`
   `printf( "%.7f\n", x );`

ANS: Floating point numbers should never be used in loops due to imprecision. This imprecision often causes infinite loops to occur. To correct this, an integer variable should be used in the `for` loop.

e) The following code should output the odd integers from 999 to 1:

```
for ( x = 999; x >= 1; x += 2 )
   printf( "%d\n", x );
```

ANS: loop should be decrementing not incrementing.

```
for ( x = 999; x >= 1; x -= 2 )
   printf( "%d\n", x );
```

f) The following code should output the even integers from 2 to 100:

```
counter = 2;

Do {
   if ( counter % 2 == 0 )
      printf( "%d\n", counter );

   counter += 2;
} While ( counter < 100 );
```

ANS: `D` in `Do` should be lowercase. `W` in `While` should be lowercase. The range of `2` to `100` needs to be printed, so the relational operator `<` should be changed to `<=`, to include `100`. The if test is not necessary here, because `counter` is being incremented by `2`, and will always be even within the body of the `do…while`.

g) The following code should sum the integers from 100 to 150 (assume `total` is initialized to 0):

```
for ( x = 100; x <= 150; x++ );
   total += x;
```

ANS: semicolon at the end of the for statement should be removed, such that `total += x;` is in the loop's body.

```
for ( x = 100; x <= 150; x++ ) /* ; removed */
   total += x;
```

4.6 State which values of the control variable `x` are printed by each of the following `for` statements:

a) `for ( x = 2; x <= 13; x += 2 )`
   `printf( "%d\n", x );`

ANS: `2, 4, 6, 8, 10, 12`

b) `for ( x = 5; x <= 22; x += 7 )`
   `printf( "%d\n", x );`

ANS: `5, 12, 19`

c) `for ( x = 3; x <= 15; x += 3 )`
   `printf( "%d\n", x );`

ANS: `3, 6, 9, 12, 15`

d) `for ( x = 1; x <= 5; x += 7 )`
   `printf( "%d\n", x );`

ANS: `1`

e) `for ( x = 12; x >= 2; x -= 3 )`
   `printf( "%d\n", x );`

ANS: `12, 9, 6, 3`

4.7 Write `for` statements that print the following sequences of values:

a) 1, 2, 3, 4, 5, 6, 7

ANS:

```
for ( i = 1; i <= 7; i++ )
   printf( "%d ", i );
```

b) 3, 8, 13, 18, 23

ANS:

```
/* increments of 5 */
```

```
for ( i = 3; i <= 23; i += 5 )
   printf( “%d ", i );
```

c) 20, 14, 8, 2, -4, -10

ANS:

```
/* decrements of 6 */
for ( i = 20; i >= -10; i -= 6 )
   printf( “%d ", i );
```

d) 19, 27, 35, 43, 51

ANS:

```
/* increments of 8 */
for ( i = 19; i <= 51; i += 8 )
   printf( “%d “, i );
```

4.8 What does the following program do?

```
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int x;
   int y;
   int i;
   int j;

   /* prompt user for input */
   printf( "Enter two integers in the range 1-20: " );
   scanf( "%d%d", &x, &y ); /* read values for x and y */

   for ( i = 1; i <= y; i++ ) { /* count from 1 to y */

      for ( j = 1; j <= x; j++ ) { /* count from 1 to x */
         printf( "@" ); /* output @ */
      } /* end inner for */

      printf( "\n" ); /* begin new line */
   } /* end outer for */

   return 0; /* indicate program ended successfully */

} /* end function main */
```

ANS:

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

4.9 Write a program that sums a sequence of integers. Assume that the first integer read with `scanf` specifies the number of values remaining to be entered. Your program should read only one value each time `scanf` is executed. A typical input sequence might be

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

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

ANS:

```
/* Exercise 4.9 Solution */
#include <stdio.h>

int main( void )
{
   int sum = 0; /* current sum */
   int number;  /* number of values */
   int value;   /* current value */
   int i;       /* counter */

   /* display prompt */
   printf( "Enter the number of values"
           " to be processed: " );
   scanf( "%d", &number ); /* input number of values */

   /* loop number times */
   for ( i = 1; i <= number; i++ ) {
      printf( "Enter a value: " );
      scanf( "%d", &value );
      sum += value; /* add to sum */
   } /* end for */

   /* display sum */
   printf( "Sum of the %d values is %d\n", number, sum );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the number of values to be processed: 5
Enter a value: 10
Enter a value: 15
Enter a value: 20
Enter a value: 25
Enter a value: 30
Sum of the 5 values is 100
```

4.10 Write a program that calculates and prints the average of several integers. Assume the last value read with `scanf` 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.

ANS:

```
/* Exercise 4.10 Solution */
#include <stdio.h>

int main( void )
{
   int value;     /* current value */
   int count = 0; /* number of values */
   int total = 0; /* sum of integers */

   /* display prompt */
   printf( "Enter an integer ( 9999 to end ): " );
   scanf( "%d", &value );

```

```
   /* loop while sentinel value not read from user */
   while ( value != 9999 ) {
      total += value; /* update total */
      ++count;

      /* get next value */
      printf( "Enter next integer ( 9999 to end ): " );
      scanf( "%d", &value );
   } /* end while */

   /* show average if more than 0 values entered */
   if ( count != 0 ) {
      printf( "\nThe average is:  %.2f\n", ( double ) total / count );
   } /* end if */
   else {
      printf( "\nNo values were entered.\n" );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter an integer ( 9999 to end ): 1
Enter next integer ( 9999 to end ): 2
Enter next integer ( 9999 to end ): 3
Enter next integer ( 9999 to end ): 4
Enter next integer ( 9999 to end ): 5
Enter next integer ( 9999 to end ): 6
Enter next integer ( 9999 to end ): 9999

The average is:  3.50
```

4.11 Write a program that finds the smallest of several integers. Assume that the first value read specifies the number of values remaining.

ANS:

```
/* Exercise 4.11 Solution */
#include <stdio.h>

int main( void )
{
   int number;   /* number of integers */
   int value;    /* value input by user */
   int smallest; /* smallest number */
   int i;        /* counter */

   /* prompt user for number of integers */
   printf( "Enter the number of integers to be processed: " );
   scanf( "%d", &number );

   /* prompt user for an integer */
   printf( "Enter an integer: " );
   scanf( "%d", &smallest );

   /* loop until user has entered all integers */
   for ( i = 2; i <= number; i++ ) {
      printf( "Enter next integer: " ); /* get next integer */
      scanf( "%d", &value );

```

```
      /* if value is smaller than smallest */
      if ( value < smallest ) {
         smallest = value;
      } /* end if */

   } /* end for */

   printf( "\nThe smallest integer is: %d\n", smallest );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the number of integers to be processed: 5
Enter an integer: 372
Enter next integer: 920
Enter next integer: 73
Enter next integer: 8
Enter next integer: 3433

The smallest integer is: 8
```

4.12 Write a program that calculates and prints the sum of the even integers from 2 to 30.
ANS:

```
/* Exercise 4.12 Solution */
#include <stdio.h>

int main( void )
{
   int i;       /* counter */
   int sum = 0; /* current sum of integers */

   /* loop through even integers up to 30 */
   for ( i = 2; i <= 30; i += 2 ) {
      sum += i; /* add i to sum */
   } /* end for */

   printf( "Sum of the even integers from 2 to 30 is: %d\n", sum );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Sum of the even integers from 2 to 30 is: 240
```

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

```
/* Exercise 4.13 Solution */
#include <stdio.h>

int main( void )
{
   long i;           /* counter */
   long product = 1; /* current product */

```

```
   /* loop through odd integers up to 15 */
   for ( i = 3; i <= 15; i += 2 ) {
      product *= i; /* update product */
   } /* end for */

   printf( "Product of the odd integers from 1 to 15 is: %ld\n", product );

   return 0; /* indicate successful termination */

} /* end main */
```

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

4.14 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?

ANS:

```
/* Exercise 4.14 Solution */
#include <stdio.h>

int main( void )
{
   int i;         /* outer counter */
   int j;         /* inner counter */
   int factorial; /* current factorial value */

   printf( "X\tFactorial of X\n" ); /* display table headers */

   /* compute the factorial for 1 to 5 */
   for ( i = 1; i <= 5; i++ ) {
      factorial = 1;

      /* calculate factorial of current number */
      for ( j = 1; j <= i; j++ ) {
         factorial *= j;
      } /* end inner for */

      printf( "%d\t%d\n", i, factorial );
   } /* end outer for */

   return 0; /* indicate successful termination */

} /* end main */
```

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

4.15 Modify the compound interest program of Section 4.6 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.

ANS:

```
/* Exercise 4.15 Solution */
#include <stdio.h>
#include <math.h>

int main( void )
{
   int year;                   /* year counter */
   int rate;                   /* interest rate */
   double amount;              /* amount on deposit */
   double principal = 1000.0; /* starting principal */

   /* loop through interest rates 5% to 10% */
   for ( rate = 5; rate <= 10; rate++ ) {

      /* display table headers */
      printf( "Interest Rate: %f\n", rate / 100.0 );
      printf( "%s%21s\n", "Year","Amount on deposit" );

      /* calculate amount on deposit for each of ten years */
      for ( year = 1; year <= 10; year++ ) {

         /* calculate new amount for specified year */
         amount = principal * pow( 1 + ( rate / 100.0 ), year );

         /* output one table row */
         printf( "%4d%21.2f\n", year, amount );
      } /* end for */

      printf( "\n" );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Interest Rate: 0.050000
Year     Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.63
   4              1215.51
   5              1276.28
   6              1340.10
   7              1407.10
   8              1477.46
   9              1551.33
  10              1628.89

...

Interest Rate: 0.080000
Year     Amount on deposit
   1              1080.00
   2              1166.40
   3              1259.71
   4              1360.49
   5              1469.33
   6              1586.87
   7              1713.82
   8              1850.93
   9              1999.00
  10              2158.92

...

Interest Rate: 0.100000
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
```

4.16 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 `printf` statement of the form `printf( "*" );` (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.]

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

ANS:

```
/* Exercise 4.16 Solution */
#include <stdio.h>

int main( void )
{
   int row;    /* row counter */
```

```
   int col;   /* column counter */
   int space; /* spaces counter */

   /* Pattern A, loop 10 times for rows */
   for ( row = 1; row <= 10; row++ ) {

      /* print row asterisks */
      for ( col = 1; col <= row; col++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );

   /* Pattern B, loop 10 times for rows
      row counts down to correspond to number of asterisks */
   for ( row = 10; row >= 1; row-- ) {

      /* print row asterisks */
      for ( col = 1; col <= row; col++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );

   /* Pattern C, loop 10 times for rows
      row counts down to correspond to number of asterisks */
   for ( row = 10; row >= 1; row-- ) {

      /* print (10 - row) number of preceding spaces */
      for ( space = 1; space <= 10 - row; space++ ) {
         printf( " " );
      } /* end for */

      /* print row asterisks */
      for ( col = 1; col <= row; col++ ) {
        printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );

   /* Pattern D, loop 10 times for rows */
   for ( row = 1; row <= 10; row++ ) {

      /* print (10 - row) number of preceding spaces */
      for ( space = 1; space <= 10 - row; space++ ) {
         printf( " " );
      } /* end for */

      /* print row asterisks */
      for ( col = 1; col <= row; col++ ) {
         printf( "*" );
      } /* end for */
```

```

      printf( "\n" );
   } /* end for */

   printf( "\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

4.17 Collecting money becomes increasingly difficult during periods of recession, so companies may tighten their credit limits to prevent their accounts receivable (money owed to them) from becoming too large. In response to a prolonged recession, one company has cut its customer's credit limits in half. Thus, if a particular customer had a credit limit of $2000, this customer’s credit limit is now $1000. If a customer had a credit limit of $5000, this customer's credit limit is now $2500. Write a program that analyzes the credit status of three customers of this company. For each customer you are given:

a) The customer’s account number
b) The customer’s credit limit before the recession
c) The customer’s current balance (i.e., the amount the customer owes the company).

Your program should calculate and print the new credit limit for each customer and should determine (and print) which customers have current balances that exceed their new credit limits.

ANS:

```
/* Exercise 4.17 Solution */
#include <stdio.h>

int main( void )
{
   int account;      /* current account number */
   int i;            /* counter */
   double limit;     /* current credit limit */
   double balance;   /* current balance */
   double newLimit; /* new credit limit */

   /* loop three times */
   for ( i = 1; i <= 3; i++ ) {

      /* get account number, credit limit and balance */
      printf( "\nEnter account, limit, balance: " );
      scanf( "%d%lf%lf", &account, &limit, &balance );

      newLimit = limit / 2.0; /* calculate new limit */
      printf( "New credit limit for account %d is %.2f\n", account, newLimit );

      /* if balance is greater than new credit limit */
      if ( balance > newLimit ) {
         printf( "Limit exceeded for account %d\n", account );
      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter account, limit, balance: 100 4000.00 2136.87
New credit limit for account 100 is 2000.00
Limit exceeded for account 100

Enter account, limit, balance: 200 10500.00 4927.39
New credit limit for account 200 is 5250.00

Enter account, limit, balance: 300 1000.00 750.00
New credit limit for account 300 is 500.00
Limit exceeded for account 300
```

4.18 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 *******.

ANS:

```
/* Exercise 4.18 Solution */
#include <stdio.h>

int main( void )
{
   int i;      /* outer counter */
   int j;      /* inner counter */
   int number; /* current number */

   printf( "Enter 5 numbers between 1 and 30: " );

   /* loop 5 times */
   for ( i = 1; i <= 5; i++ ) {
      scanf( "%d", &number );

      /* print asterisks corresponding to current input */
      for ( j = 1; j <= number; j++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter 5 numbers between 1 and 30: 28 5 13 24 7
****************************
*****
*************
************************
*******
```

4.19 A mail order house sells five different products whose retail prices are shown in the following table:

Product number	Retail price
1	$ 2.98
2	$ 4.50
3	$ 9.98
4	$ 4.49
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.

ANS:

```
/* Exercise 4.19 Solution */
#include <stdio.h>

int main( void )
{
   int product;        /* current product number */
   int quantity;       /* quantity of current product sold */
   double total = 0.0; /* current total retail value */

   /* prompt for input */
   printf( "Enter pairs of item numbers and quantities.\n");
   printf( "Enter -1 for the item number to end input.\n" );
   scanf( "%d", &product );

   /* loop while sentinel value not read from user */
   while ( product != -1 ) {
      scanf( "%d", &quantity );

      /* determine product number and corresponding retail price */
      switch ( product ) {

         case 1:
            total += quantity * 2.98; /* update total */
            break;

         case 2:
            total += quantity * 4.50; /* update total */
            break;

         case 3:
            total += quantity * 9.98; /* update total */
            break;

         case 4:
            total += quantity * 4.49; /* update total */
            break;

         case 5:
            total += quantity * 6.87; /* update total */
            break;
```

```
         default:
            printf( "Invalid product code:  %d\n", product );
            printf( "            Quantity:  %d\n", quantity );
      } /* end switch */

      scanf( "%d", &product ); /* get next input */
   } /* end while */

   /* display total retail value */
   printf( "The total retail value was:  %.2f\n", total );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter pairs of item numbers and quantities.
Enter -1 for the item number to end input.
1 1
2 1
3 1
4 1
5 1
6 1
Invalid product code:  6
            Quantity:  1
1 1
-1
The total retail value was:  31.80
```

4.20 Complete the following truth tables by filling in each blank with 0 or 1

ANS: .

Condition1	Condition2	Condition1 && Condition2
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Condition1	Condition2	Condition1 || Condition2
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

Condition1	!Condition1
0	1
nonzero	0

4.21 Rewrite the program of Fig. 4.2 so that the initialization of the variable `counter` is done in the definition instead of the `for` statement.

ANS:

```
/* Exercise 4.21 Solution */
#include <stdio.h>

int main( void )
{
   int counter = 1; /* initialize counter */

   /* leave first statement empty */
   for (  ; counter <= 10; counter++ ) {
      printf( "%d\n", counter );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
1
2
3
4
5
6
7
8
9
10
```

4.22 Modify the program of Fig. 4.7 so that it calculates the average grade for the class.

ANS:

```
/* Exercise 4.22 Solution */
#include <stdio.h>

int main( void )
{
   int grade;           /* current grade */
   int aCount = 0;      /* total a grades */
   int bCount = 0;      /* total b grades */
   int cCount = 0;      /* total c grades */
   int dCount = 0;      /* total d grades */
   int fCount = 0;      /* total f grades */
   double averageGrade; /* average grade for class */

   /* prompt user for grades */
   printf( "Enter the letter grades.\n");
   printf( "Enter the EOF character to end input.\n" );

   /* loop while not end of file */
   while ( ( grade = getchar() ) != EOF ) {

      /* determine which grade was input */
      switch ( grade ) {

         case 'A':    /* grade was uppercase A */
         case 'a':    /* grade was lowercase a */
            ++aCount; /* update grade A counter */
            break;    /* exit switch */
```

```c

         case 'B':    /* grade was uppercase B */
         case 'b':    /* grade was lowercase b */
            ++bCount; /* update grade B counter */
            break;    /* exit switch */

         case 'C':    /* grade was uppercase C */
         case 'c':    /* grade was lowercase c */
            ++cCount; /* update grade C counter */
            break;    /* exit switch */

         case 'D':    /* grade was uppercase C */
         case 'd':    /* grade was lowercase c */
            ++dCount; /* update grade C counter */
            break;    /* exit switch */

         case 'F':    /* grade was uppercase C */
         case 'f':    /* grade was lowercase c */
            ++fCount; /* update grade C counter */
            break;    /* exit switch */

         case '\n':   /* ignore newlines, */
         case '\t':   /* tabs, */
         case ' ':    /* and spaces in input */
            break;    /* exit switch */

         default:     /* catch all other characters */
            printf( "Incorrect letter grade entered." );
            printf( " Enter a new grade.\n" );
            break; /* optional, will exit switch anyway */
      } /* end switch */

   } /* end while */

   /* output totals for each grade */
   printf( "\nThe totals for each letter grade are:\n" );
   printf( "A: %d\n", aCount );
   printf( "B: %d\n", bCount );
   printf( "C: %d\n", cCount );
   printf( "D: %d\n", dCount );
   printf( "F: %d\n", fCount );

   /* calculate average grade */
   averageGrade =
      ( 4 * aCount + 3 * bCount + 2 * cCount + dCount  ) /
      ( aCount + bCount + cCount + dCount + fCount );

   /* output appropriate message for average grade */
   if ( averageGrade > 3.4 ) {
      printf( "Average grade is A\n" );
   } /* end if */
   else if ( averageGrade > 2.4 ) {
      printf( "Average grade is B\n" );
   } /* end else if */
   else if ( averageGrade > 1.4 ) {
      printf( "Average grade is C\n" );
   } /* end else if */
   else if ( averageGrade > 0.4 ) {
      printf( "Average grade is D\n" );
   } /* end else if */
```

```c
    else {
       printf( "Average grade is F\n" );
    } /* end else */

    return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the letter grades.
Enter the EOF character to end input.
A B B B C D F
C C C D D D C B
B A A B C C C
^Z

The totals for each letter grade are:
A: 3
B: 6
C: 8
D: 4
F: 1
Average grade is C
```

4.23 Modify the program of Fig. 4.6 so that 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 remainder operations, respectively. Insert a period.]

ANS:

```c
/* Exercise 4.23 Solution */
#include <stdio.h>
#include <math.h>

int main( void )
{
   int year;                /* year counter */
   int amount;              /* amount on deposit, in pennies */
   int dollars;             /* dollar portion of amount */
   int cents;               /* cents portion of amount */
   int principal = 100000; /* starting principal, in pennies ($1000) */
   double rate = .05;       /* interest rate */

   /* display headers for table */
   printf( "%s%21s\n", "Year", "Amount on deposit" );

   /* loop 10 times */
   for ( year = 1; year <= 10; year++ ) {

      /* determine new amount (in pennies) */
      amount = principal * pow( 1.0 + rate, year );

      /* determine cents portion of amount (last two digits) */
      cents = amount % 100;

      /* determine dollars portion of amount */
      /* integer division truncates decimal places */
      dollars = amount / 100;

      /* display year, dollar portion followed by period */
      printf( "%4d%18d.", year, dollars );

```

```
      /* display cents portion */
      /* if cents portion only 1 digit, insert 0 */
      if ( cents < 10 ) {
         printf("0%d\n", cents);
      } /* end if */
      else {
         printf("%d\n", cents);
      } /* end else */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Year     Amount on deposit
   1              1050.00
   2              1102.50
   3              1157.62
   4              1215.50
   5              1276.28
   6              1340.09
   7              1407.10
   8              1477.45
   9              1551.32
  10              1628.89
```

4.24 Assume `i = 1`, `j = 2`, `k = 3` and `m = 2`. What does each of the following statements print?

a) `printf( "%d", i == 1 );`
ANS: 1
b) `printf( "%d", j == 3 );`
ANS: 0
c) `printf( "%d", i >= 1 && j < 4 );`
ANS: 1
d) `printf( "%d", m < = 99 && k < m );`
ANS: 0
e) `printf( "%d", j >= i || k == m );`
ANS: 1
f) `printf( "%d", k + m < j || 3 - j >= k );`
ANS: 0
g) `printf( "%d", !m );`
ANS: 0
h) `printf( "%d", !( j - m ) );`
ANS: 1
i) `printf( "%d", !( k > m ) );`
ANS: 0
j) `printf( "%d", !( j > k ) );`
ANS: 1

4.25 Print a table of decimal, binary, octal, and hexadecimal equivalents. If you are not familiar with these number systems, read Appendix E first if you would like to attempt this exercise.
ANS: see Exercise 4.34 Solution

4.26 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 one 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?

ANS: 3.14 occurs at an accuracy of 627, 3.141 occurs at an accuracy of 2458, 3.1415 occurs at an accuracy around 147,000, and 3.14159 occurs at an accuracy around 319,000.

```
/* Exercise 4.26 Solution */
#include<stdio.h>

int main( void )
{
   long double pi = 0.0;     /* approximated value for pi */
   long double num = 4.0;    /* numerator */
   long double denom = 1.0; /* denominator of current term */
   long int loop;            /* loop counter */
   long int accuracy;        /* number of terms */

   accuracy = 400000; /* set decimal accuracy */

   /* display table headers */
   printf( "Accuracy set at: %ld\n", accuracy );
   printf( "term\t\t  pi\n" );

   /* loop through each term */
   for ( loop = 1; loop <= accuracy; loop++ ) {

      /* if odd-numbered term, add current term */
      if ( loop % 2 != 0 ) {
         pi += num / denom;
      } /* end if */
      else { /* if even-numbered term, subtract current term */
         pi -= num / denom;
      } /* end else */

      /* display number of terms and approximated
         value for pi with 8 digits of precision */
      printf( "%ld\t\t%Lf\n", loop, pi );

      denom += 2.0; /* update denominator */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Accuracy set at: 400000
term              pi
1                4.000000
2                2.666667
3                3.466667
4                2.895238
5                3.339683
6                2.976046

...

995              3.142598
996              3.140589
997              3.142596
998              3.140591
999              3.142594

...

399998           3.141590
399999           3.141595
400000           3.141590
```

4.27 *(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 the hypotenuse all no larger than 500. Use a triple-nested `for` loop that simply tries all possibilities. This is an example of "brute force" computing. It is not aesthetically pleasing to many people. But there are many reasons why these techniques are important. First, with computing power increasing at such a phenomenal pace, solutions that would have taken years or even centuries of computer time to produce with the technology of just a few years ago can now be produced in hours, minutes or even seconds. Recent microprocessor chips can process a billion instructions per second! Second, as you will learn in more advanced computer science courses, there are large numbers of interesting problems for which there is no known algorithmic approach other than sheer brute force. We investigate many kinds of problem-solving methodologies in this book. We will consider many brute force approaches to various interesting problems.

ANS:

```c
/* Exercise 4.27 Solution */
#include<stdio.h>

int main( void )
{
   int count = 0;          /* number of triples found */
   long int side1;         /* side1 value counter */
   long int side2;         /* side2 value counter */
   long int hypotenuse;    /* hypotenuse value counter */
   long int hyptSquared;   /* hypotenuse squared */
   long int sidesSquared; /* sum of squares of sides */

   /* side1 values range from 1 to 500 */
   for ( side1 = 1; side1 <= 500; side1++ ) {

      /* side2 values range from current side1 to 500 */
      for ( side2 = 1; side2 <= 500; side2++ ) {

         /* hypotenuse values range from current side2 to 500 */
         for ( hypotenuse = 1; hypotenuse <= 500; hypotenuse++ ) {

            /* calculate square of hypotenuse value */
            hyptSquared = hypotenuse * hypotenuse;
```

```

            /* calculate sum of squares of sides */
            sidesSquared = side1 * side1 + side2 * side2;

            /* if hypotenuse squared = side1 squared + side2 squared,
               Pythagorean triple */
            if ( hyptSquared == sidesSquared ) {

               /* display triple */
               printf( "%ld %ld %ld\n", side1, side2, hypotenuse );
               ++count; /* update count */
            } /* end if */

         } /* end for */

      } /* end for */

   } /* end for */

   /* display total number of triples found */
   printf( "A total of %d triples were found.\n", count );

   return 0; /* indicate successful termination */

} /* end main */
```

```
3 4 5
4 3 5
5 12 13
6 8 10
7 24 25
8 6 10

...

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.
```

4.28 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 a $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 based on that employee's paycode.

ANS:

```
/* Exercise 4.28 Solution */
#include<stdio.h>

int main( void )
{
   int payCode;       /* current employee's pay code */
   int managers = 0; /* total number of managers */
   int hWorkers = 0; /* total number of hourly workers */
   int cWorkers = 0; /* total number of commission workers */
   int pWorkers = 0; /* total number of pieceworkers */
   int pieces;        /* current pieceworker's number of pieces */
   double mSalary;    /* manager's salary */
   double hSalary;    /* hourly worker's salary */
   double cSalary;    /* commission worker's salary */
   double pSalary;    /* pieceworker's salary */
   double hours;      /* total hours worked */
   double otPay;      /* overtime pay */
   double otHours;    /* overtime hours */
   double pay;        /* current employee's weekly pay */

   /* prompt for first employee input */
   printf( "Enter paycode ( -1 to end): " );
   scanf( "%d", &payCode );

   /* loop while sentinel value not read from user */
   while ( payCode != -1 ) {

      /* switch to appropriate computation according to pay code */
      switch ( payCode ) {

         /* pay code 1 corresponds to manager */
         case 1:

            /*prompt for weekly salary */
            printf( "Manager selected.\n" );
            printf( "Enter weekly salary: " );
            scanf( "%lf", &mSalary );

            /* manager's pay is weekly salary */
            printf( "The manager's pay is $%.2f\n", mSalary );

            ++managers; /* update total number of managers */
            break; /* exit switch */

         /* pay code 2 corresponds to hourly worker */
         case 2:

            /* prompt for hourly salary */
            printf( "Hourly worker selected.\n" );
```

```
            printf( "Enter the hourly salary: " );
            scanf( "%lf", &hSalary );

            /* prompt for number of hours worked */
            printf( "Enter the total hours worked: " );
            scanf( "%lf", &hours );

            /* pay fixed for up to 40 hours, 1.5 for hours over 40 */
            if ( hours > 40.0 ) {

               /* calculate OT hours and total pay */
               otHours = hours - 40.0;
               otPay = hSalary * 1.5 * otHours + hSalary * 40.0;

               printf( "Worker has worked %.1f overtime hours.\n", otHours );
               printf( "Workers pay is $%.2f\n", otPay );
            } /* end if */
            else { /* no overtime */
               pay = hSalary * hours;
               printf( "Worker's pay is $%.2f\n", pay );
            } /* end else */

            ++hWorkers; /* update total number of hourly workers */
            break; /* exit switch */

         /* pay code 3 corresponds to commission worker */
         case 3:

            /* prompt for gross weekly sales */
            printf( "Commission worker selected.\n" );
            printf( "Enter gross weekly sales: " );
            scanf( "%lf", &cSalary );

            /* pay $250 plus 5.7% of gross weekly sales */
            pay = 250.0 + 0.057 * cSalary;
            printf( "Commission Worker's pay is $%.2f\n", pay );

            ++cWorkers; /* update total number of commission workers */
            break; /* exit switch */

         /* pay code 4 corresponds to pieceworker */
         case 4:

            /* prompt for number of pieces */
            printf( "Piece worker selected.\nEnter number of pieces: " );
            scanf( "%d", &pieces );

            /* prompt for wage per piece */
            printf( "Enter wage per piece: " );
            scanf( "%lf", &pSalary );

            pay = pieces * pSalary; /* compute pay */
            printf( "Piece Worker's pay is $%.2f\n", pay );

            ++pWorkers; /* update total number of pieceworkers */
            break; /* exit switch */

         /* default case */
         default :
            printf( "Invalid pay code.\n" );
            break;
      } /* end switch */
```

```c

      /* prompt for next employee input */
      printf( "\nEnter paycode ( -1 to end ): " );
      scanf( "%d", &payCode );
   } /* end while */

   /* display total counts for each type of employee */
   printf( "\n" );
   printf( "Total number of managers paid          : %d\n", managers );
   printf( "Total number of hourly workers paid    : %d\n", hWorkers );
   printf( "Total number of commission workers paid: %d\n", cWorkers );
   printf( "Total number of piece workers paid     : %d\n", pWorkers );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter paycode ( -1 to end): 4
Piece worker selected.
Enter number of pieces: 200
Enter wage per piece: 20
Piece Worker's pay is $4000.00

Enter paycode ( -1 to end ): -1

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

```
Enter paycode ( -1 to end): 1
Manager selected.
Enter weekly salary: 2500
The manager's pay is $2500.00

Enter paycode ( -1 to end ): 2
Hourly worker selected.
Enter the hourly salary: 10.50
Enter the total hours worked: 75
Worker has worked 35.0 overtime hours.
Workers pay is $971.25

Enter paycode ( -1 to end ): 3
Commission worker selected.
Enter gross weekly sales: 9000
Commission Worker's pay is $763.00

Enter paycode ( -1 to end ): 4
Piece worker selected.
Enter number of pieces: 200
Enter wage per piece: 20
Piece Worker's pay is $4000.00

Enter paycode ( -1 to end ): -1

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

4.29 *(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, and 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 ) )`

ANS:

```c
/* Exercise 4.29 Solution */
#include<stdio.h>

int main( void )
{
   int x = 10; /* define current variable value */
   int y = 1;  /* define current variable value */
   int a = 3;  /* define current variable value */
   int b = 3;  /* define current variable value */
   int g = 5;  /* define current variable value */
   int Y = 1;  /* define current variable value */
   int i = 2;  /* define current variable value */
   int j = 9;  /* define current variable value */

   /* display variable values */
   printf( "current variable values are: \n" );
   printf( "x = %d, y = %d, a = %d,", x, y, a );
   printf( " b = %d\n", b );
   printf( "g = %d, Y = %d, i = %d,", g, Y, i );
   printf( " j = %d\n\n", j );

   /* part a */
   if ( ( !( x < 5 ) && !( y >= 7 ) ) == ( !( ( x < 5 ) || ( y >= 7 )
      ) ) ) {
       printf( "!( x < 5 ) && !( y >= 7 ) is equivalent to"
              " !( ( x < 5 ) || ( y >= 7 ) )\n" );
   } /* end if */
   else {
       printf( "!( x < 5 ) && !( y >= 7 ) is not equivalent to"
              " !( ( x < 5 ) || ( y >= 7 ) )\n" );
   } /* end else */

   /* part b */
   if ( ( !( a == b ) || !( g != 5 ) ) == ( !( ( a == b ) && ( g != 5 )
      ) ) ) {
       printf( "!( a == b ) || !( g != 5 ) is equivalent to"
              " !( ( a == b ) && ( g != 5 ) )\n" );
   } /* end if */
   else {
       printf( "!( a == b ) || !( g != 5 ) is not equivalent to"
              " !( ( a == b ) && ( g != 5 ) )\n" );
   } /* end else */

   /* part c */
   if ( !( ( x <= 8 ) && ( Y > 4 ) ) == ( !( x <= 8 ) || !( Y > 4 )
      ) ) {
      printf( "!( ( x <= 8 ) && ( Y > 4 ) ) is equivalent to"
             " ( !( x <= 8 ) || !( Y > 4 ) )\n" );
   } /* end if */
```

```c
    else {
       printf( "!( ( x <= 8 ) && ( Y > 4 ) ) is not equivalent to"
               " ( !( x <= 8 ) || !( Y > 4 ) )\n" );
    } /* end else */

    /* part d */
    if ( !( ( i > 4 ) || ( j <= 6 ) ) == ( !( i > 4 ) && !( j <= 6 )
       ) ) {
        printf( "!( ( i > 4 ) || ( j <= 6 ) ) is equivalent to"
                " ( !( i > 4 ) && !( j <= 6 ) )\n" );
    } /* end if */
    else {
        printf( "!( ( i > 4 ) || ( j <= 6 ) ) is not equivalent to"
                " ( !( i > 4 ) && !( j <= 6 ) )\n" );
    } /* end else */

    return 0; /* indicate successful termination */

 } /* end main */
```

```
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 ) )
```

4.30 Rewrite the program of Fig. 4.7 by replacing the `switch` statement with a nested `if…else` statement; be careful to deal with the `default` case properly. Then rewrite this new version by replacing the nested `if…else` statement with a series of `if` statements; here, too, be careful to deal with the `default` case properly (this is more difficult than in the nested `if…else` version). This exercise demonstrates that `switch` is a convenience and that any `switch` statement can be written with only single-selection statements.

```c
/* Exercise 4.30 Part A Solution */
#include <stdio.h>

int main( void )
{
   int grade;       /* current grade */
   int aCount = 0; /* total A grades */
   int bCount = 0; /* total B grades */
   int cCount = 0; /* total C grades */
   int dCount = 0; /* total D grades */
   int fCount = 0; /* total F grades */

   /* prompt user for grades */
   printf( "Enter the letter grades." );
   printf( " Enter the EOF character to end input:\n" );

   /* while EOF not entered by user */
   while ( ( grade = getchar() ) != EOF ) {

      /* Update count for appropriate grade */
      if ( grade == 'A' || grade == 'a' ) {
         ++aCount;
      } /* end if */
```

```
      else if ( grade == 'B' || grade == 'b' ) {
         ++bCount;
      } /* end else if */
      else if ( grade == 'C' || grade == 'c' ) {
         ++cCount;
      } /* end else if */
      else if ( grade == 'D' || grade == 'd' ) {
         ++dCount;
      } /* end else if */
      else if ( grade == 'F' || grade == 'f' ) {
         ++fCount;
      } /* end else if */
      else if ( grade == '\n' || grade == ' ' ) {
         ;      /* empty body */
      } /* end else if */
      else {
         printf( "Incorrect letter grade entered." );
         printf( " Enter a new grade.\n" );
      } /* end else */

   } /* end while */

   /* display totals for each grade */
   printf( "\nTotals for each letter grade were:\n" );
   printf( "A: %d\n", aCount );
   printf( "B: %d\n", bCount );
   printf( "C: %d\n", cCount );
   printf( "D: %d\n", dCount );
   printf( "F: %d\n", fCount );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the letter grades. Enter the EOF character to end input:
A
c
b
d
e
Incorrect letter grade entered. Enter a new grade.
f
^Z

Totals for each letter grade were:
A: 1
B: 1
C: 1
D: 1
F: 1
```

```
/* Exercise 4.30 Part B Solution */
#include <stdio.h>

int main( void )
{
   int grade;       /* current grade */
   int aCount = 0; /* total A grades */
   int bCount = 0; /* total B grades */
   int cCount = 0; /* total C grades */
   int dCount = 0; /* total D grades */
   int fCount = 0; /* total F grades */
```

```c

   /* prompt user for grades */
   printf( "Enter the letter grades." );
   printf( " Enter the EOF character to end input:\n" );

   /* while EOF not entered by user */
   while ( ( grade = getchar() ) != EOF ) {

      /* update count for appropriate grade */
      if ( grade == 'A' || grade == 'a' ) {
         ++aCount;
      } /* end if */

      if ( grade == 'B' || grade == 'b' ) {
         ++bCount;
      } /* end if */

      if ( grade == 'C' || grade == 'c' ) {
         ++cCount;
      } /* end if */

      if ( grade == 'D' || grade == 'd' ) {
         ++dCount;
      } /* end if */

      if ( grade == 'F' || grade == 'f' ) {
         ++fCount;
      } /* end if */

      if ( grade == '\n' || grade == ' ' ) {
         ;  /* empty body */
      } /* end if */

      /* default */
      if ( grade != 'a' && grade != 'A' &&
           grade != 'B' && grade != 'b' &&
           grade != 'c' && grade != 'C' &&
           grade != 'd' && grade != 'd' &&
           grade != 'f' && grade != 'F' &&
           grade != '\n'&& grade != ' ' ) {

         printf( "Incorrect letter grade entered." );
         printf( " Enter a new grade.\n" );
      } /* end if */

   } /* end while */

   /* display totals for each grade */
   printf( "\nTotals for each letter grade were:\n" );
   printf( "A: %d\n", aCount );
   printf( "B: %d\n", bCount );
   printf( "C: %d\n", cCount );
   printf( "D: %d\n", dCount );
   printf( "F: %d\n", fCount );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the letter grades. Enter the EOF character to end input:
A
b
c
s
Incorrect letter grade entered. Enter a new grade.
d
f
^Z

Totals for each letter grade were:
A: 1
B: 1
C: 1
D: 1
F: 1
```

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

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

ANS:

```c
/* Exercise 4.31 Solution */
#include <stdio.h>

int main( void )
{
   int line;     /* line counter */
   int space;    /* space counter */
   int asterisk; /* asterisk counter */

   /* top half */
   for ( line = 1; line <= 9; line += 2 ) {

      /* print preceding spaces */
      for ( space = ( 9 - line ) / 2; space > 0; space-- ) {
         printf( " " );
      } /* end for */

      /* print asterisks */
      for ( asterisk = 1; asterisk <= line; asterisk++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   /* bottom half */
   for ( line = 7; line >= 0; line -= 2 ) {

      /* print preceding spaces */
      for ( space = ( 9 - line ) / 2; space > 0; space-- ) {
         printf( " " );
      } /* end for */
```

```

      /* print asterisks */
      for ( asterisk = 1; asterisk <= line; asterisk++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

4.32 Modify the program you wrote in Exercise 4.31 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.

ANS:

```
/* Exercise 4.32 Solution */
#include<stdio.h>

int main( void )
{
   int line;     /* line counter */
   int space;    /* space counter */
   int asterisk; /* asterisk counter */
   int size;     /* number of rows in diamond */

   /* prompt for diamond size */
   printf( "Enter an odd number for the diamond size ( 1-19 ):\n" );
   scanf( "%d", &size );

   /* top half */
   for ( line = 1; line <= size -  2; line += 2 ) {

      /* print preceding spaces */
      for ( space = ( size - line ) / 2; space > 0; space-- ) {
         printf( " " );
      } /* end for */

      /* print asterisks */
      for ( asterisk = 1; asterisk <= line; asterisk++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

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

      /* print preceding spaces */
      for ( space = ( size - line ) / 2; space > 0; space-- ) {
         printf( " " );
      } /* end for */

      /* print asterisks */
      for ( asterisk = 1; asterisk <= line; asterisk++ ) {
         printf( "*" );
      } /* end for */
```

```

      printf( "\n" );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter an odd number for the diamond size ( 1-19 ):
13
      *
     ***
    *****
   *******
  *********
 ***********
*************
 ***********
  *********
   *******
    *****
     ***
      *
```

4.33 Write a program that prints a table of all the Roman numeral equivalents of the decimal numbers in the range 1 to 100.
ANS:

```
/* Exercise 4.33 Solution */
#include<stdio.h>

int main( void )
{
   int loop; /* loop counter */
   int div;  /* tens digit */
   int mod;  /* ones digit */

   /* display table headers */
   printf( " Roman\nNumeral\t\tDecimal\n" );

   /* loop 100 times */
   for ( loop = 1; loop <= 100; loop++ ) {
      div = loop / 10; /* separate tens digit */
      mod = loop % 10; /* separate ones digit */

      /* switch structure for tens digit */
      switch ( div ) {

         /* print appropriate Roman numeral for tens digit */
         case 0:
            break;

         case 1:
            printf( "X" );
            break; /* exit switch */

         case 2:
            printf( "XX" );
            break; /* exit switch */

         case 3:
            printf( "XXX" );
            break; /* exit switch */
```

```

         case 4:
            printf( "XL" );
            break; /* exit switch */

         case 5:
            printf( "L" );
            break; /* exit switch */

         case 6:
            printf( "LX" );
            break; /* exit switch */

         case 7:
             printf( "LXX" );
             break; /* exit switch */

         case 8:
             printf( "LXXX" );
             break; /* exit switch */

         case 9:
             printf( "XC" );
             break; /* exit switch */

         case 10:
             printf( "C" );
             break; /* exit switch */

         default:
             break; /* exit switch */
      } /* end switch */

      /* switch structure for ones digit */
      switch( mod ) {

         /* print appropriate Roman numeral for ones digit */
         case 0:
            printf( "\t\t%4d\n", div * 10 );
            break; /* exit switch */

         case 1:
            printf( "I\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 2:
            printf( "II\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 3:
            printf( "III\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 4:
            printf( "IV\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 5:
            printf( "V\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

```

```
         case 6:
            printf( "VI\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 7:
            printf( "VII\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 8:
            printf( "VIII\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 9:
            printf( "IX\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         case 10:
            printf( "X\t\t%4d\n", div * 10 + mod );
            break; /* exit switch */

         default:
            break; /* exit switch */
      } /* end switch */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
 Roman
Numeral         Decimal
I                  1
II                 2
III                3
IV                 4
V                  5
VI                 6
VII                7
VIII               8
IX                 9
X                 10

...

LXXXIX            89
XC                90
XCI               91
XCII              92
XCIII             93
XCIV              94
XCV               95
XCVI              96
XCVII             97
XCVIII            98
XCIX              99
C                100
```

4.34 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 E before you attempt this exercise.

ANS:

```
/* Exercise 4.34 Solution */
#include <stdio.h>

int main( void )
{
   int loop;   /* loop counter */
   int number; /* current number */
   int temp1;  /* temporary integer */

   /* print table headers */
   printf( "Decimal\t\tBinary\t\tOctal\t\tHexadecimal\n" );

   /* loop through values 1 to 256 */
   for ( loop = 1; loop <= 256; loop++ ) {
      printf( "%d\t\t", loop );
      number = loop;

      /* binary numbers */
      printf( "%c", number == 256 ? '1' : '0' );

      printf( "%c", number < 256 && number >= 128 ? '1' : '0' );
      number %= 128;

      printf( "%c", number < 128 && number >= 64 ? '1' : '0' );
      number %= 64;

      printf( "%c", number < 64 && number >= 32 ? '1' : '0' );
      number %= 32;

      printf( "%c", number < 32 && number >= 16 ? '1' : '0' );
      number %= 16;

      printf( "%c", number < 16 && number >= 8 ? '1' : '0' );
      number %= 8;

      printf( "%c", number < 8 && number >= 4 ? '1' : '0' );
      number %= 4;

      printf( "%c", number < 4 && number >= 2 ? '1' : '0' );
      number %= 2;

      printf( "%c\t", number == 1 ? '1' : '0' );

      /* octal numbers */
      number = loop;

      printf( "%d", number < 512 && number >= 64 ? number / 64 : 0 );
      number %= 64;

      printf( "%d", number < 64 && number >= 8 ? number / 8 : 0 );
      number %= 8;

      printf( "%d\t\t", number == 0 ? 0 : number );

      /* hexadecimal numbers */
      number = loop;
      temp1 = 16;

```

```c
      if ( number < 4096 && number >= 256 ) {
         printf( "%d", number / 256 );
      } /* end if */

      if ( number < 256 && number >= 16 ) {
         temp1 = number / 16;
         number %= 16;
      } /* end if */
      else {
         printf( "0" );
      } /* end else */

      /* convert to letter if temp1 is above 9 */
      if ( temp1 <= 9 ) {
         printf( "%d", temp1 );
      } /* end if */
      else if ( temp1 >= 10 && temp1 <= 15 ) {
         printf( "%c", 'A' + ( temp1 - 10 ) );
      } /* end else if */

      /* convert to letter if number is above 9 */
      if ( number <= 9 ) {
         printf( "%d", number );
      } /* end if */
      else if ( number >= 10 && number <= 15 ) {
         printf( "%c", 'A' + ( number - 10 ) );
      } /* end else if */

      printf( "\n" );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Decimal         Binary          Octal           Hexadecimal
1               000000001       001             01
2               000000010       002             02
3               000000011       003             03
4               000000100       004             04
5               000000101       005             05
6               000000110       006             06
7               000000111       007             07
8               000001000       010             08
9               000001001       011             09
10              000001010       012             0A

...

250             011111010       372             FA
251             011111011       373             FB
252             011111100       374             FC
253             011111101       375             FD
254             011111110       376             FE
255             011111111       377             FF
256             100000000       400             10F
```

4.35 Describe the process you would use to replace a `do...while` loop with an equivalent `while` loop. What problem occurs when you try to replace a `while` loop with an equivalent `do...while` loop? Suppose you have been told that you must remove a `while` loop and replace it with a `do...while`. What additional control statement would you need to use and how would you use it to ensure that the resulting program behaves exactly as the original?

ANS: The body of a `do…while` loop becomes the body of a while loop, and the contents of the body are repeated before the `while` loop. In a `do…while` loop, the body is executed at least once, whereas execution of the body in a `while` loop depends on the continuation condition.

Replacing a `while` loop with a `do…while` loop requires an if selection statement. The `do…while` loop would be the body of the `if` statement and the condition would be the same as the loop continuation condition in the `do…while`.

4.36 Write a program that inputs the year in the range 1994 through 1999 and uses `for`-loop repetition to produce a condensed, neatly printed calendar. Watch out for leap years.

ANS:

```c
/* Exercise 4.36 Solution */
/* This is a simple calender solution, that does */
/* not account for the shifting of dates from    */
/* year to year.                                 */

#include<stdio.h>

int main( void )
{
   int year;        /* current year */
   int leapYear;    /* leap year, 1 = yes, 0 = no */
   int days;        /* total days in current month */
   int month;       /* current month */
   int space;       /* space counter */
   int dayPosition; /* starting day position of year */
   int dayNum;      /* counter for days of the month */

   /* loop until input is valid */
   do {
      printf( "Enter a calendar year between 1994 and 1999: " );
      scanf( "%d", &year );
   } while ( year < 1994 || year > 1999 ); /* end do...while */

   /* determine starting day position */
   switch ( year ) {

      case 1994:
         dayPosition = 7;
         break; /* exit switch */

      case 1995:
         dayPosition = 1;
         break; /* exit switch */

      case 1996:
         dayPosition = 2;
         break; /* exit switch */

      case 1997:
         dayPosition = 4;
         break; /* exit switch */

      case 1998:
         dayPosition = 5;
         break; /* exit switch */

```

```
         case 1999:
            dayPosition = 6;
            break; /* exit switch */
      } /* end switch */

      /* check for leap years */
      if ( year % 400 == 0 ) {
         leapYear = 1;
      } /* end if */
      else if ( year % 4 == 0 && year % 100 != 0 ) {
         leapYear = 1;
      } /* end else if */
      else {
         leapYear = 0;
      } /* end else */

      /* loop through months and print calendar */
      for ( month = 1; month <= 12; month++ ) {

         /* begin with the month */
         switch ( month ) {

            case 1:
               printf( "\n\nJanuary %d\n", year );
               days = 31;
               break; /* exit switch */

            case 2:
               printf( "\n\nFebruary %d\n", year );
               days = leapYear == 1 ? 29 : 28;
               break; /* exit switch */

            case 3:
               printf( "\n\nMarch %d\n", year );
               days = 31;
               break; /* exit switch */

            case 4:
               printf( "\n\nApril %d\n", year );
               days = 30;
               break; /* exit switch */

            case 5:
               printf( "\n\nMay %d\n", year );
               days = 31;
               break; /* exit switch */

            case 6:
               printf( "\n\nJune %d\n", year );
               days = 30;
               break; /* exit switch */

            case 7:
               printf( "\n\nJuly %d\n", year );
               days = 31;
               break; /* exit switch */

            case 8:
               printf( "\n\nAugust %d\n", year );
               days = 31;
               break; /* exit switch */
```

```c

         case 9:
            printf( "\n\nSeptember %d\n", year );
            days = 30;
            break; /* exit switch */

         case 10:
            printf( "\n\nOctober %d\n", year );
            days = 31;
            break; /* exit switch */

         case 11:
            printf( "\n\nNovember %d\n", year );
            days = 30;
            break; /* exit switch */

         case 12:
            printf( "\n\nDecember %d\n", year );
            days = 31;
            break; /* exit switch */
      } /* end switch */

      printf( " S  M  T  W  R  F  S\n" ); /* print heads */

      /* move to proper space to begin printing month */
      for ( space = 1; space < dayPosition; space++ ) {
         printf( "   " );
      } /* end for */

      /* print days of the month */
      for ( dayNum = 1; dayNum <= days; dayNum++ ) {
         printf( "%2d ", dayNum );

         /* if end of the week, start a new line */
         if ( dayPosition % 7 == 0 ) {
            printf( "\n" );
            dayPosition = 1; /* reset dayPosition */
         } /* end if */
         else {
            ++dayPosition;
         } /* end else */

      } /* end for */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a calendar year between 1994 and 1999: 1999


January 1999
 S  M  T  W  R  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

February 1999
 S  M  T  W  R  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28

March 1999
 S  M  T  W  R  F  S
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
.
.
.
```

4.37 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. 4.11.

ANS:

```c
/* Exercise 4.37 Solution */
#include <stdio.h>

int main( void )
{
   int x;             /* loop counter */
   int breakOut = 1; /* breakout condition */

   /* test for breakout condition */
   for ( x = 1; x <= 10 && breakOut == 1; x++ ) {

      /* break out of loop after x = 4 */
      if ( x == 4 ) {
         breakOut = -1;
      } /* end if */

      printf( "%d ", x );
   } /* end for */

   printf( "\nBroke out of loop at x = %d\n", x );

   return 0; /* indicate successful termination */

} /* end main */
```

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

4.38 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++ )
         printf( "*" );
      printf( "\n" );
   }
   printf( "\n" );
}
```

ANS:

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

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

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

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

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

4.39 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. 4.12.

ANS:

```
/* Exercise 4.39 Solution */
#include <stdio.h>

int main( void )
{
   int x; /* loop counter */

   /* loop 10 times */
   for ( x = 1; x <= 10; x++ ) {

      /* if x == 5, skip to next interation */
      if ( x == 5 ) {
         ++x;
      } /* end if */

      printf( "%d ", x );
   } /* end for */
```

```
   printf( "\nUsed ++x to skip printing the value 5\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

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

5

C Functions: Solutions

SOLUTIONS

5.8 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`

5.9 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 program should use the function `calculateCharges` to determine the charge for each customer. Your outputs should appear in the following format:

```
Enter the hours parked for 3 cars: 1.5 4.0 24.0
  Car          Hours         Charge
    1            1.5           2.00
    2            4.0           2.50
    3           24.0          10.00
TOTAL           29.5          14.50
```

ANS:

```c
/* Exercise 5.9 Solution */
#include <stdio.h>
#include <math.h>

double calculateCharges( double hours ); /* function prototype */

int main()
{
   double h;                  /* number of hours for current car */
   double currentCharge;      /* parking charge for current car */
   double totalCharges = 0.0; /* total charges */
   double totalHours = 0.0;   /* total number of hours */
   int i;                     /* loop counter */
   int first = 1;             /* flag for printing table headers */

   printf( "Enter the hours parked for 3 cars: " );

   /* loop 3 times for 3 cars */
   for ( i = 1; i <= 3; i++ ) {
      scanf( "%lf", &h );
      totalHours += h; /* add current hours to total hours */

      /* if first time through loop, display headers */
      if ( first ) {
         printf( "%5s%15s%15s\n", "Car", "Hours", "Charge" );

         /* set flag to false to prevent from printing again */
         first = 0;
      } /* end if */

      /* calculate current car's charge and update total */
      totalCharges += ( currentCharge =  calculateCharges( h ) );

      /* display row data for current car */
      printf( "%5d%15.1f%15.2f\n", i, h, currentCharge );
   } /* end for */

   /* display row data for totals */
   printf( "%5s%15.1f%15.2f\n", "TOTAL", totalHours, totalCharges );

   return 0; /* indicate successful termination */

} /* end main */

/* calculateCharges returns charge according to number of hours */
double calculateCharges( double hours )
{
   double charge; /* calculated charge */

   /* $2 for up to 3 hours */
   if ( hours < 3.0 ) {
      charge = 2.0;
   } /* end if */

   /* $.50 for each hour or part thereof in excess of 3 hours */
   else if ( hours < 19.0 ) {
      charge = 2.0 + .5 * ceil( hours - 3.0 );
   } /* end else if */
   else { /* maximum charge $10 */
```

```
      charge = 10.0;
   } /* end else */

   return charge; /* return calculated charge */

} /* end function calculateCharges */
```

5.10 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.

ANS:

```
/* Exercise 5.10 Solution */
#include <stdio.h>
#include <math.h>

void calculateFloor( void ); /* function prototype */

int main()
{
   calculateFloor(); /* call function calculateFloor */

   return 0; /* indicate successful termination */

} /* end main */

/* calculateFloor rounds 5 inputs */
void calculateFloor( void )
{
   double x; /* current input */
   double y; /* current input rounded */
   int loop; /* loop counter */

   /* loop for 5 inputs */
   for ( loop = 1; loop <= 5; loop++ ) {
      printf( "Enter a floating point value: " );
      scanf( "%lf", &x );

      /* y holds rounded input */
      y = floor( x + .5 );
      printf( "%f rounded is %.1f\n\n", x, y );
   } /* end for */

} /* end function calculateFloor */
```

```
Enter a floating point value: 1.5
1.500000 rounded is 2.0

Enter a floating point value: 5.55
5.550000 rounded is 6.0

Enter a floating point value: 73.2341231432
73.234123 rounded is 73.0

Enter a floating point value: 9.0
9.000000 rounded is 9.0

Enter a floating point value: 4
4.000000 rounded is 4.0
```

5.11 Function `floor` may 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 (i.e., 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) `roundToHundreths( 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.

ANS:

```
/* Exercise 5.11 Solution */
#include <stdio.h>
#include <math.h>

double roundToInteger( double n );     /* function prototype */
double roundToTenths( double n );      /* function prototype */
double roundToHundredths( double n );  /* function prototype */
double roundToThousandths( double n ); /* function prototype */

int main()
{
   int i;          /* loop counter */
   int count;      /* number of values to process */
   double number; /* current input */

   printf( "How many numbers do you want to process? " );
   scanf( "%d", &count );

   /* loop for inputs */
   for ( i = 0; i < count; i++ ) {
      printf( "Enter number: " );
      scanf( "%lf", &number );

      /* display number rounded to nearest integer */
      printf( "%f rounded to an integer is %f\n",
            number, roundToInteger( number ) );
```

```
      /* display number rounded to nearest tenth */
      printf( "%f rounded to the nearest tenth is %f\n",
            number, roundToTenths( number ) );

      /* display number rounded to nearest hundredth */
      printf( "%f rounded to the nearest hundredth is %f\n",
            number, roundToHundredths( number ) );

      /* display number rounded to nearest thousandth */
      printf( "%f rounded to the nearest thousandth is %f\n\n",
            number, roundToThousandths( number ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* roundToInteger rounds n to nearest integer */
double roundToInteger( double n )
{
   return floor( n + .5 );

} /* end function roundToInteger */

/* roundToTenths rounds n to nearest tenth */
double roundToTenths( double n )
{
   return floor( n * 10 + .5 ) / 10;

} /* end function roundToTenths */

/* roundToHundredths rounds n to nearest hundredth */
double roundToHundredths( double n )
{
   return floor( n * 100 + .5 ) / 100;

} /* end function roundToHundredths */

/* roundToThousandths rounds n to nearest thousandth */
double roundToThousandths( double n )
{
   return floor( n * 1000 + .5 ) / 1000;

} /* end function roundToThousandths */
```

```
How many numbers do you want to process? 1
Enter number: 8.54739
8.547390 rounded to an integer is 9.000000
8.547390 rounded to the nearest tenth is 8.500000
8.547390 rounded to the nearest hundredth is 8.550000
8.547390 rounded to the nearest thousandth is 8.547000
```

5.12 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: Using `srand` enables the sequence of pseudo random numbers produced by `rand` to change each time the program is executed. The program should not be randomized while in the debugging stages because repetition is helpful in debugging.

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 toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specific times.

5.13 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() % 113;`

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

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

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

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

5.14 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: `printf( "%d\n", 2 * ( 1 + rand() % 5 ) );`

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

ANS: `printf( "%d\n", 1 + 2 * ( 1 + rand() % 5 ) );`

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

ANS: `pritnf( "%d\n", 6 + 4 * ( rand() % 5 ) );`

5.15 Define a function called `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`. Test your program with the side values specified in Fig. 5.18.

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

```
/* Exercise 5.15 Solution */
#include <stdio.h>
#include <math.h>

double hypotenuse( double s1, double s2 ); /* function prototype */

int main()
{
   int i;         /* loop counter */
   double side1; /* value for first side */
   double side2; /* value for second side */

   /* loop 3 times */
   for ( i = 1; i <= 3; i++ ) {
      printf( "Enter the sides of the triangle: " );
      scanf( "%lf%lf", &side1, &side2 );

      /* calculate and display hypotenuse value */
      printf( "Hypotenuse:  %.1f\n\n", hypotenuse( side1, side2 ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* hypotenuse calculates value of hypotenuse of
   a right triangle given two side values */
double hypotenuse( double s1, double s2 )
{
   return sqrt( pow( s1, 2 ) + pow( s2, 2 ) );

} /* end function hypotenuse */
```

```
Enter the sides of the triangle: 3.0 4.0
Hypotenuse:  5.0

Enter the sides of the triangle: 5.0 12.0
Hypotenuse:  13.0

Enter the sides of the triangle: 8.0 15.0
Hypotenuse:  17.0
```

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

$$\texttt{base}^{\texttt{exponent}}$$

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

ANS:

```
/* Exercise 5.16 Solution */
#include <stdio.h>

int integerPower( int b, int e );

int main()
{
   int exp;  /* integer exponent */
   int base; /* integer base */

   printf( "Enter integer base and exponent: " );
   scanf( "%d%d", &base, &exp );

   printf( "%d to the power %d is: %d\n",
           base, exp, integerPower( base, exp ) );

   return 0; /* indicate successful termination */

} /* end main */

/* integerPower calculates and returns b raised to the e power */
int integerPower( int b, int e )
{
   int product = 1; /* resulting product */
   int i;           /* loop counter */

   /* multiply product times b (e repetitions) */
   for ( i = 1; i <= e; i++ ) {
      product *= b;
   } /* end for */

   return product; /* return resulting product */

} /* end function integerPower */
```

```
Enter integer base and exponent: 5 3
5 to the power 3 is: 125
```

5.17 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 `1` (true) if the second is a multiple of the first, and `0` (false) otherwise. Use this function in a program that inputs a series of pairs of integers.

ANS:

```
/* Exercise 5.17 Solution */
#include <stdio.h>

int multiple( int a, int b ); /* function prototype */

int main()
{
   int x; /* first integer */
   int y; /* second integer */
   int i; /* loop counter */

   /* loop 3 times */
   for ( i = 1; i <= 3; i++ ) {
      printf( "Enter two integers: " );
      scanf( "%d%d", &x, &y );

      /* determine if second is multiple of first */
      if ( multiple( x, y ) ) {
         printf( "%d is a multiple of %d\n\n", y, x );
      } /* end if */
      else {
         printf( "%d is not a multiple of %d\n\n", y, x );
      } /* end else */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* multiple determines if b is multiple of a */
int multiple( int a, int b )
{
   return !( b % a );

} /* end function multiple */
```

```
Enter two integers: 2 10
10 is a multiple of 2

Enter two integers: 5 17
17 is not a multiple of 5

Enter two integers: 3 696
696 is a multiple of 3
```

5.18 Write a program that inputs a series of integers and passes them one at a time to function `even` which uses the remainder operator to determine if an integer is even. The function should take an integer argument and return `1` if the integer is even and `0` otherwise.

ANS:

```
/* Exercise 5.18 Solution */
#include <stdio.h>

int even( int a ); /* function prototype */

int main()
{
   int x; /* current input */
   int i; /* loop counter */

   /* loop for 3 inputs */
   for ( i = 1; i <= 3; i++ ) {
      printf( "Enter an integer: " );
      scanf( "%d", &x );

      /* determine if input is even */
      if ( even( x ) ) {
         printf( "%d is an even integer\n\n", x );
      } /* end if */
      else {
         printf( "%d is not an even integer\n\n", x );
      } /* end else */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* even returns true if a is even */
int even( int a )
{
   return !( a % 2 );

} /* end function even */
```

```
Enter an integer: 7
7 is not an even integer

Enter an integer: 6
6 is an even integer

Enter an integer: 10000
10000 is an even integer
```

5.19 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:

```
Enter side: 4
****
****
****
****
```

ANS:

```c
/* Exercise 5.19 Solution */
#include <stdio.h>

void square( int s ); /* function prototype */

int main()
{
   int side; /* input side length */

   printf( "Enter side: " );
   scanf( "%d", &side );

   square( side ); /* display solid square of asterisks */

   return 0; /* indicate successful termination */

} /* end main */

/* square displays solid square of asterisks with specified side */
void square( int s )
{
   int i; /* outer loop counter */
   int j; /* inner loop counter */

   /* loop side times for number of rows */
   for ( i = 1; i <= s; i++ ) {

      /* loop side times for number of columns */
      for ( j = 1; j <= s; j++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

} /* end function square */
```

5.20 Modify the function created in Exercise 5.19 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:

```
Enter a character and the side length: # 5
#####
#####
#####
#####
#####
```

ANS:

```
/* Exercise 5.20 Solution */
#include <stdio.h>

void square( int side, char fillCharacter ); /* function prototype */

int main()
{
   int s;  /* side length */
   char c; /* fill character */

   printf( "Enter a character and the side length: " );
   scanf( "%c%d", &c, &s );

   square( s, c ); /* display solid square of input character */

   return 0; /* indicate successful termination */

} /* end main */

/* square displays solid square of fillCharacter with specified side */
void square( int side, char fillCharacter )
{
   int loop;  /* outer loop counter */
   int loop2; /* inner loop counter */

   /* loop side times for number of rows */
   for ( loop = 1; loop <= side; loop++ ) {

      /* loop side times for number of columns */
      for ( loop2 = 1; loop2 <= side; loop2++ ) {
         printf( "%c", fillCharacter );
      } /* end for */

      printf( "\n" );
   } /* end for */

} /* end function square */
```

5.21 Use techniques similar to those developed in Exercises 5.19 and 5.20 to produce a program that graphs a wide range of shapes.

5.22 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
```

ANS:

```
/* Exercise 5.22 Solution */
#include <stdio.h>

int quotient( int a, int b );  /* function prototype */
int remainder( int a, int b ); /* function prototype */

int main()
{
   int number;           /* input number */
   int divisor = 10000; /* current divisor */

   printf( "Enter an integer between 1 and 32767: " );
   scanf( "%d", &number );

   printf( "The digits in the number are:\n" );

   /* determine and print each digit */
   while ( number >= 10 ) {

      /* if number is >= current divisor, determine digit */
      if ( number >= divisor ) {

         /* use quotient to determine current digit */
         printf( "%d  ", quotient( number, divisor ) );

         /* update number to be remainder */
         number = remainder( number, divisor );

         /* update divisor for next digit */
         divisor = quotient( divisor, 10 );
      } /* end if */
      else { /* if number < current divisor, no digit */
         divisor = quotient( divisor, 10 );
      } /* end else */

   } /* end while */

   printf( "%d\n", number );

   return 0; /* indicate successful termination */

} /* end main */

/* Part A: determine quotient using integer division */
int quotient( int a, int b )
{
   return a / b;

} /* end function quotient */
```

```c

/* Part B: determine remainder using the remainder operator */
int remainder( int a, int b )
{
   return a % b;

} /* end function remainder */
```

5.23 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 times, both of which are within one 12-hour cycle of the clock.

ANS:

```
/* Exercise 5.23 Solution */
#include <stdio.h>
#include <math.h>

/* function prototype */
unsigned seconds( unsigned h, unsigned m, unsigned s );

int main()
{
   int hours;      /* current time's hours */
   int minutes;    /* current time's minutes */
   int secs;       /* current time's seconds */
   int first;      /* first time, in seconds */
   int second;     /* second time, in seconds */
   int difference; /* difference between two times, in seconds */

   printf( "Enter the first time as three integers: " );
   scanf( "%d%d%d", &hours, &minutes, &secs );

   /* calculate first time in seconds */
   first = seconds( hours, minutes, secs );

   printf( "Enter the second time as three integers: " );
   scanf( "%d%d%d", &hours, &minutes, &secs );

   /* calculate second time in seconds */
   second = seconds( hours, minutes, secs );

   /* calculate difference */
   difference = fabs( first - second );

   /* display difference */
   printf( "The difference between the times is %d seconds\n",
          difference );

   return 0; /* indicate successful termination */

} /* end main */

/* seconds returns number of seconds since clock "struck 12"
   given input time as hours h, minutes m, seconds s */
unsigned seconds( unsigned h, unsigned m, unsigned s )
{
   return 3600 * h + 60 * m + s;

} /* end function seconds */
```

```
Enter the first time as three integers: 4 20 39
Enter the second time as three integers: 7 20 39
The difference between the times is 10800 seconds
```

5.24 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.

ANS:

```c
/* Exercise 5.24 Solution */
#include <stdio.h>

int celcius( int fTemp );    /* function prototype */
int fahrenheit( int cTemp ); /* function prototype */

int main()
{
   int i; /* loop counter */

   /* display table of Fahrenheit equivalents of Celsius temperature */
   printf( "Fahrenheit equivalents of Celcius temperatures:\n" );
   printf( "Celcius\t\tFahrenheit\n" );

   /* display Fahrenheit equivalents of Celsius 0 to 100 */
   for ( i = 0; i <= 100; i++ ) {
      printf( "%d\t\t%d\n", i, fahrenheit( i ) );
   } /* end for */

   /* display table of Celsius equivalents of Fahrenheit temperature */
   printf( "\nCelcius equivalents of Fahrenheit temperatures:\n" );
   printf( "Fahrenheit\tCelcius\n" );

   /* display Celsius equivalents of Fahrenheit 32 to 212 */
   for ( i = 32; i <= 212; i++ ) {
      printf( "%d\t\t%d\n", i, celcius( i ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* celsius returns Celsius equivalent of fTemp,
   given in Fahrenheit */
int celcius( int fTemp )
{
   return ( int ) ( 5.0 / 9.0 * ( fTemp - 32 ) );

} /* end function celsius */

/* fahrenheit returns Fahrenheit equivalent of cTemp,
   given in Celsius */
int fahrenheit( int cTemp )
{
   return ( int ) ( 9.0 / 5.0 * cTemp + 32 );

} /* end function fahrenheit */
```

```
Fahrenheit equivalents of Celcius temperatures:
Celcius         Fahrenheit
0               32
1               33
2               35
3               37
4               39
5               41
6               42
7               44
8               46
9               48
.
.
.
Celcius equivalents of Fahrenheit temperatures:
Fahrenheit      Celcius
32              0
33              0
34              1
35              1
36              2
37              2
38              3
39              3
40              4
41              5
.
.
.
```

5.25 Write a function that returns the smallest of three floating point numbers.
ANS:

```c
/* Exercise 5.25 Solution */
#include <stdio.h>

/* function prototype */
double smallest3( double a, double b, double c );

int main()
{
   double x; /* first input */
   double y; /* second input */
   double z; /* third input */

   printf( "Enter three doubleing point values: " );
   scanf( "%lf%lf%lf", &x, &y, &z );

   /* determine smallest value */
   printf( "The smallest value is %f\n", smallest3( x, y, z ) );

   return 0; /* indicate successful termination */

} /* end main */

/* smallest3 returns the smallest of a, b and c */
double smallest3( double a, double b, double c )
{
   double smallest = a; /* assume a is the smallest */

   if ( b < smallest ) { /* if b is smaller */
      smallest = b;
   } /* end if */

   if ( c < smallest ) { /* if c is smaller */
      smallest = c;
   } /* end if */

   return smallest; /* return smallest value */

} /* end function smallest3 */
```

```
Enter three doubleing point values: 3.3 4.4 5.5
The smallest value is 3.300000
```

```
Enter three doubleing point values: 4.4 5.5 3.3
The smallest value is 3.300000
```

```
Enter three doubleing point values: 4.4 3.3 5.5
The smallest value is 3.300000
```

5.26 An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number because 6 = 1 + 2 + 3. Write a function `perfect` that determines if 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.

ANS:

```c
/* Exercise 5.26 Solution */
#include <stdio.h>

int perfect( int value ); /* function prototype */

int main()
{
   int j; /* loop counter */

   printf( "For the integers from 1 to 1000:\n" );

   /* loop from 2 to 1000 */
   for ( j = 2; j <= 1000; j++ ) {

      /* if current integer is perfect */
      if ( perfect( j ) ) {
         printf( "%d is perfect\n", j );
      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* perfect returns true if value is perfect integer,
   i.e., if value is equal to sum of its factors */
int perfect( int value )
{
   int factorSum = 1; /* current sum of factors */
   int i;             /* loop counter */

   /* loop through possible factor values */
   for ( i = 2; i <= value / 2; i++ ) {

      /* if i is factor */
      if ( value % i == 0 ) {
         factorSum += i; /* add to sum */
      } /* end if */

   } /* end for */

   /* return true if value is equal to sum of factors */
   if ( factorSum == value ) {
      return 1;
   } /* end if */
   else {
      return 0;
   } /* end else */

} /* end function perfect */
```

```
For the integers from 1 to 1000:
6 is perfect
28 is perfect
496 is perfect
```

5.27 An integer is said to be *prime* if it is divisible only by 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 if 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 if 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.

ANS:

```
/* Exercise 5.27 Solution Part B Solution */
#include <stdio.h>

int prime( int n );

int main()
{
   int loop;      /* loop counter */
   int count = 0; /* total number of primes found */

   printf( "The prime numbers from 1 to 10000 are:\n" );

   /* loop through 1 to 10000 */
   for ( loop = 1; loop <= 10000; loop++ ) {

      /* if current number is prime */
      if ( prime( loop ) ) {
         ++count;
         printf( "%6d", loop );

         /* new line after 10 values displayed */
         if ( count % 10 == 0 ) {
            printf( "\n" );
         } /* end if */

      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* prime returns 1 if n is prime */
int prime( int n )
{
   int loop2; /* loop counter */

   /* loop through possible factors */
   for ( loop2 = 2; loop2 <= n / 2; loop2++ ) {

      /* if factor found, not prime */
      if ( n % loop2 == 0 ) {
         return 0;
      } /* end if */

   } /* end for */

   return 1; /* return 1 if prime */

} /* end function prime */
```

```
The prime numbers from 1 to 10000 are:
     1     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
   .
   .
   .
  9733  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
```

```c
/* Exercise 5.27 Part C Solution */
#include <stdio.h>
#include <math.h>

int prime( int n ); /* function prototype */

int main()
{
   int j;         /* loop counter */
   int count = 0; /* total number of primes found */

   printf( "The prime numbers from 1 to 10000 are:\n" );

   /* loop through numbers 1 to 10000 */
   for ( j = 1; j <= 10000; j++ ) {

      /* if current number prime */
      if ( prime( j ) ) {
         ++count;
         printf( "%5d", j );

         /* new line after 10 values displayed */
         if ( count % 10 == 0 ) {
            printf( "\n" );
         } /* end if */

      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* prime returns 1 if n is prime */
int prime( int n )
{
   int i; /* loop counter */

   /* loop through possible factors */
   for ( i = 2; i <= ( int ) sqrt( n ); i++ ) {

      /* if factor found, not prime */
      if ( n % i == 0 ) {
         return 0;
      } /* end if */

   } /* end for */
```

```
   return 1;

} /* end function prime */
```

5.28 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.

ANS:

```
/* Exercise 5.28 Solution */
#include <stdio.h>

int reverseDigits( int n );

int main()
{
   int number; /* input number */

   printf( "Enter a number between 1 and 9999: " );
   scanf( "%d", &number );

   /* find number with digits reversed */
   printf( "The number with its digits reversed is: %d\n",
          reverseDigits( number ) );

   return 0; /* indicate successful termination */

} /* end main */

/* reverseDigits returns number obtained by
   reversing digits of n */
int reverseDigits( int n )
{
   int reverse = 0;    /* reversed number */
   int divisor = 1000; /* current divisor */
   int multiplier = 1; /* current multiplier */

   /* loop until single-digit number */
   while ( n > 9 ) {

      /* if n >= current divisor, determine digit */
      if ( n >= divisor ) {

         /* update reversed number with current digit */
         reverse += n / divisor * multiplier;

         n %= divisor; /* update n */
         divisor /= 10; /* update divisor */
         multiplier *= 10; /* update multiplier */
      } /* end if */
      else { /* else, no digit */
         divisor /= 10; /* update divisor */
      } /* end else */

   } /* end while */

   reverse += n * multiplier;

   return reverse; /* return reversed number */

} /* end function reverseDigits */
```

```
Enter a number between 1 and 9999: 6
The number with its digits reversed is: 6
```

```
Enter a number between 1 and 9999: 9273
The number with its digits reversed is: 3729
```

5.29 The *greatest common divisor (GCD)* of two integers is the largest integer that evenly divides each of the two numbers. Write function `gcd` that returns the greatest common divisor of two integers.

ANS:

```
/* Exercise 5.29 Solution */
#include <stdio.h>

int gcd( int x, int y ); /* function prototype */

int main()
{
   int j; /* loop counter */
   int a; /* first number */
   int b; /* second number */

   /* loop for 5 pairs of inputs */
   for ( j = 1; j <= 5; j++ ) {
      printf( "Enter two integers: " );
      scanf( "%d%d", &a, &b );

      /* find greatest common divisor of a and b */
      printf( "The greatest common divisor "
         "of %d and %d is %d\n\n", a, b, gcd( a, b ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* gcd find greatest common divisor of x and y */
int gcd( int x, int y )
{
   int i;
   int greatest = 1; /* current gcd, 1 is minimum */

   /* loop from 2 to smaller of x and y */
   for ( i = 2; i <= ( ( x < y ) ? x : y ); i++ ) {

      /* if current i divides both x and y */
      if ( x % i == 0 && y % i == 0 ) {
         greatest = i; /* update greatest common divisor */
      } /* end if */

   } /* end for */

   return greatest; /* return greatest common divisor found */

} /* end function gcd */
```

```
Enter two integers: 75 225
The greatest common divisor of 75 and 225 is 75

Enter two integers: 99 30
The greatest common divisor of 99 and 30 is 3

Enter two integers: 17 22
The greatest common divisor of 17 and 22 is 1

Enter two integers: 100 92
The greatest common divisor of 100 and 92 is 4

Enter two integers: 10005 15
The greatest common divisor of 10005 and 15 is 15
```

5.30 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.

ANS:

```c
/* Exercise 5.30 Solution */
#include <stdio.h>

int qualityPoints( int average ); /* function prototype */

int main()
{
   int average; /* current average */
   int loop;    /* loop counter */

   /* loop for 5 inputs */
   for ( loop = 1; loop <= 5; loop++ ) {
      printf( "\nEnter the student's average: " );
      scanf( "%d", &average );

      /* determine and display corresponding quality points */
      printf( "%d on a 4 point scale is %d\n",
         average, qualityPoints( average ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* qualityPoints takes average in range 0 to 100 and
   returns corresponding quality points on 0 to 4 scale */
int qualityPoints( int average )
{

   /* 90 <= average <= 100 */
   if ( average >= 90 ) {
      return 4;
   } /* end if */
   else if ( average >= 80 ) { /* 80 <= average <= 89 */
      return 3;
   } /* end else if */
   else if ( average >= 70 ) { /* 70 <= average <= 79 */
      return 2;
   } /* end else if */
   else if ( average >= 60 ) { /* 60 <= average <= 69 */
      return 1;
   } /* end else if */
   else { /* 0 <= average < 60 */
      return 0;
   } /* end else */

} /* end function qualityPoints */
```

```
Enter the student's average: 92
92 on a 4 point scale is 4

Enter the student's average: 87
87 on a 4 point scale is 3

Enter the student's average: 75
75 on a 4 point scale is 2

Enter the student's average: 63
63 on a 4 point scale is 1

Enter the student's average: 22
22 on a 4 point scale is 0
```

5.31 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 for a total of approximately 50 heads and 50 tails.]

ANS:

```c
/* Exercise 5.31 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int flip(); /* function prototype */

int main()
{
   int loop;           /* loop counter */
   int headCount = 0; /* total Heads count */
   int tailCount = 0; /* total Tails count */

   srand( time( NULL ) ); /* seed random number generator */

   /* simulate coin toss 100 times */
   for ( loop = 1; loop <= 100; loop++ ) {

      /* simulate coin toss, 0 refers to tails */
      if ( flip() == 0 ) {
         tailCount++; /* update Tails count */
      } /* end if */
      else {
         headCount++; /* update Heads count */
      } /* end else */

      if ( loop % 10 == 0 ) {
         printf( "\n" );
      } /* end if */

   } /* end for */

   /* display totals */
   printf( "\nThe total number of Heads was %d\n", headCount );
   printf( "The total number of Tails was %d\n", tailCount );

   return 0; /* indicate successful termination */

} /* end main */

/* flip uses random number to simulate coin toss */
int flip() {
   int HorT = rand() %2; /* scale by 2 for binary result */

   /* display result of flip */
   if ( HorT == 0 ) {
      printf( "Tails " );
   } /* end if */
   else {
      printf( "Heads " );
   } /* end else */

   return HorT; /* return result of coin toss */

} /* end function flip */
```

```
Tails Heads Tails Tails Tails Tails Heads Tails Tails Tails
Tails Tails Tails Heads Tails Heads Tails Tails Heads Tails
Tails Heads Heads Tails Tails Heads Tails Tails Tails Tails
Tails Tails Heads Heads Heads Heads Heads Heads Tails Tails
Heads Heads Heads Heads Heads Tails Tails Tails Tails Tails
Tails Tails Tails Heads Heads Tails Tails Tails Tails Heads
Tails Tails Tails Heads Heads Tails Tails Heads Tails Tails
Heads Tails Tails Heads Tails Tails Tails Tails Heads Tails
Tails Tails Tails Tails Heads Tails Heads Heads Tails Tails
Heads Tails Tails Heads Tails Tails Heads Tails Tails Tails

The total number of Heads was 34
The total number of Tails was 66
```

5.32 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.

ANS:

```c
/* Exercise 5.32 solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void multiplication( void ); /* function prototype */

int main( void )
{
   srand( time( NULL ) ); /* seed random number generator */
   multiplication(); /* begin multiplication practice */

   return 0; /* indicate successful termination */

} /* end main */

/* multiplication produces pairs of random numbers and
   prompts user for product */
void multiplication( void )
{
   int x;            /* first factor */
   int y;            /* second factor */
   int response = 0; /* user response for product */

   /* use sentinel-controlled repetition */
   printf( "Enter -1 to end.\n" );

   /* loop while sentinel value not read from user */
   while ( response != -1 ) {
      x = rand() % 10; /* generate 1-digit random number */
      y = rand() % 10; /* generate another 1-digit random number */

      printf( "How much is %d times %d? ", x, y );
      scanf( "%d", &response );

      /* loop while not sentinel value or correct response */
      while ( response != -1 && response != x * y ) {
         printf( "No. Please try again.\n? " );
         scanf( "%d", &response );
      } /* end while */

      /* correct response */
      if ( response != -1 ) {
         printf( "Very good!\n\n" );
      } /* end if */

   } /* end while */

   printf( "That's all for now. Bye.\n" );

} /* end function multiplication */
```

```
Enter -1 to end.
How much is 0 times 7? 0
Very good!

How much is 0 times 0? 0
Very good!

How much is 2 times 6? 18
No. Please try again.
? 12
Very good!

How much is 5 times 0? 0
Very good!

How much is 9 times 2? 18
Very good!

How much is 6 times 1? -1
That's all for now. Bye.
```

5.33 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. Modify the program of Exercise 5.32 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` statement with `printf` statements to issue the responses.

ANS:

```c
/* Exercise 5.33 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void correctMessage( void );   /* function prototype */
void incorrectMessage( void ); /* function prototype */
void multiplication( void );   /* function prototype */

int main()
{
   srand( time( NULL ) ); /* seed random number generator */
   multiplication(); /* begin multiplication practice */

   return 0; /* indicate successful termination */

} /* end main */

/* correctMessage randomly chooses response to correct answer */
void correctMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      /* display a random response */
      case 0:
         printf( "Very good!\n\n" );
         break; /* exit switch */

      case 1:
         printf( "Excellent!\n\n" );
         break; /* exit switch */

      case 2:
         printf( "Nice work!\n\n" );
         break; /* exit switch */

      case 3:
         printf( "Keep up the good work!\n\n" );
         break; /* exit switch */
   } /* end switch */
```

```
} /* end function correctMessage */

/* incorrectMessage randomly chooses response to incorrect answer */
void incorrectMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      /* display random response */
      case 0:
         printf( "No. Please try again.\n? " );
         break; /* exit switch */

      case 1:
         printf( "Wrong. Try once more.\n? " );
         break; /* exit switch */

      case 2:
         printf( "Don't give up!\n? " );
         break; /* exit switch */

      case 3:
         printf( "No. Keep trying.\n? " );
         break; /* exit switch */
   } /* end switch */

} /* end function incorrectMessage */

/* multiplication produces pairs of random numbers and
   prompts user for product */
void multiplication( void )
{
   int x;            /* first factor */
   int y;            /* second factor */
   int response = 0; /* user response for product */

   /* use sentinel-controlled repetition */
   printf( "Enter -1 to end.\n" );

   /* loop while sentinel value not read from user */
   while ( response != -1 ) {
      x = rand() % 10; /* generate 1-digit random number */
      y = rand() % 10; /* generate another 1-digit random number */

      printf( "How much is %d times %d? ", x, y );
      scanf( "%d", &response );

      /* loop while not sentinel value or correct response */
      while ( response != -1 && response != x * y ) {
         incorrectMessage();
         scanf( "%d", &response );
      } /* end while */

      /* correct response */
      if ( response != -1 ) {
         correctMessage();
      } /* end if */

   } /* end while */
```

```c

   printf( "That's all for now. Bye.\n" );
} /* end function multiplication */
```

```
Enter -1 to end.
How much is 7 times 6? 42
Very good!

How much is 8 times 5? 40
Excellent!

How much is 7 times 2? 15
No. Please try again.
? 14
Keep up the good work!

How much is 9 times 6? 54
Keep up the good work!

How much is 3 times 7? -1
That's all for now. Bye.
```

5.34 More sophisticated computer-aided instructions 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 5.33 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.

ANS:

```c
/* Exercise 5.34 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void multiplication( void );   /* function prototype */
void correctMessage( void );   /* function prototype */
void incorrectMessage( void ); /* function prototype */

int main()
{
   srand( time( NULL ) ); /* seed random number generator */
   multiplication(); /* begin multiplication practice */

   return 0; /* indicate successful termination */

} /* end main */

/* multiplication produces pairs of random numbers and
   prompts user for product */
void multiplication( void )
{
   int i;         /* loop counter */
   int x;         /* first factor */
   int y;         /* second factor */
   int response;  /* user response for product */
   int right = 0; /* total number of right responses */
   int wrong = 0; /* total number of wrong responses */

   /* loop 10 times */
   for ( i = 1; i <= 10; i++ ) {
      x = rand() % 10; /* generate 1-digit random number */
      y = rand() % 10; /* generate another 1-digit random number */

      printf( "How much is %d times %d? ", x, y );
      scanf( "%d", &response );

      /* loop while not correct response */
      while ( response != x * y ) {
         wrong++; /* update total number of wrong responses */
         incorrectMessage();
         scanf( "%d", &response );
      } /* end while */

      right++; /* update total number of correct responses */
      correctMessage();
   } /* end for */

   /* determine if help is needed */
   if ( ( double ) right / ( right + wrong ) < .75 ) {
      printf( "Please ask your instructor for extra help.\n" );
   } /* end if */

```

```
   printf( "That's all for now. Bye.\n" );
} /* end function multiplication */

/* correctMessage randomly chooses response to correct answer */
void correctMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      /* display random response */
      case 0:
         printf( "Very good!\n\n" );
         break; /* exit switch */

      case 1:
         printf( "Excellent!\n\n" );
         break; /* exit switch */

      case 2:
         printf( "Nice work!\n\n" );
         break; /* exit switch */

      case 3:
         printf( "Keep up the good work!\n\n" );
         break; /* exit switch */
   } /* end switch */

} /* end function correctMessage */

/* incorrectMessage randomly chooses response to incorrect answer */
void incorrectMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      /* display random response */
      case 0:
         printf( "No. Please try again.\n? " );
         break; /* exit switch */

      case 1:
         printf( "Wrong. Try once more.\n? " );
         break; /* exit switch */

      case 2:
         printf( "Don't give up!\n? " );
         break; /* exit switch */

      case 3:
         printf( "No. Keep trying.\n? " );
         break; /* exit switch */
   } /* end switch */

} /* end function incorrectMessage */
```

```
How much is 3 times 9? 27
Excellent!

How much is 1 times 3? 3
Very good!

How much is 8 times 1? 8
Very good!

How much is 3 times 6? 24
No. Please try again.
? 18
Excellent!

...

How much is 1 times 9? 9
Very good!

How much is 4 times 0? 4
Wrong. Try once more.
? 0
Excellent!

How much is 5 times 8? 40
Nice work!

That's all for now. Bye.
```

5.35 Write a C 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. [*Note:* The searching technique employed in this problem is called *binary search.* We will say more about this in the next problem.]

ANS:

```c
/* Exercise 5.35 solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void guessGame( void ); /* function prototype */

int main()
{
   srand( time( NULL ) ); /* seed random number generator */
   guessGame();

   return 0; /* indicate successful termination */

} /* end main */

/* guessGame generates numbers between 1 and 1000
   and checks user's guess */
void guessGame( void )
{
   int x;        /* randomly generated number */
   int guess;    /* user's guess */
   int response; /* response to continue game, 1=yes, 2=no */

   /* loop until user types 2 to quit game */
   do {

      /* generate random number between 1 and 1000
         1 is shift, 1000 is scaling factor */
      x = 1 + rand() % 1000;

      /* prompt for guess */
      printf( "\nI have a number between 1 and 1000.\n" );
      printf( "Can you guess my number?\n" );
      printf( "Please type your first guess.\n? " );
      scanf( "%d", &guess );

```

```
      /* loop until correct number */
      while ( guess != x ) {

         /* if guess is too low */
         if ( guess < x ) {
            printf( "Too low. Try again.\n? " );
         } /* end if */
         else { /* guess is too high */
            printf( "Too high. Try again.\n? " );
         } /* end else */

         scanf( "%d", &guess );
      } /* end while */

      /* prompt for another game */
      printf( "\nExcellent! You guessed the number!\n" );
      printf( "Would you like to play again?\n" );
      printf( "Please type ( 1=yes, 2=no )? " );
      scanf( "%d", &response );
   } while ( response == 1 ); /* end do...while */

} /* end function guessGame */
```

```
I have a number between 1 and 1000.
Can you guess my number?
Please type your first guess.
? 500
Too low. Try again.
? 750
Too high. Try again.
? 625
Too low. Try again.
? 687
Too high. Try again.
? 656
Too low. Try again.
? 671
Too low. Try again.
? 678
Too high. Try again.
? 675
Too high. Try again.
? 673
Too high. Try again.
? 672

Excellent! You guessed the number!
Would you like to play again?
Please type ( 1=yes, 2=no )? 2
```

5.36 Modify the program of Exercise 5.35 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 1 to 1000 can be guessed in 10 or fewer tries.

ANS:

```c
/* Exercise 5.36 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void guessGame( void ); /* function prototype */

int main()
{
   srand( time( NULL ) ); /* seed random number generator */
   guessGame();

   return 0; /* indicate successful termination */

} /* end main */

/* guessGame generates numbers between 1 and 1000
   and checks user's guess */
void guessGame( void )
{
   int x;         /* randomly generated number */
   int guess;     /* user's guess */
   int total = 1; /* number of guesses */
   int response;  /* response to continue game, 1=yes, 0=no */

   /* loop until user enters 0 to quit game */
   do {

      /* generate random number between 1 and 1000
         1 is shift, 1000 is scaling factor */
      x = 1 + rand() % 1000;

      /* prompt for guess */
      printf( "\nI have a number between 1 and 1000.\n" );
      printf( "Can you guess my number?\n" );
      printf( "Please type your first guess.\n? " );
      scanf( "%d", &guess );

      /* loop while not correct answer */
      while ( guess != x ) {

         /* guess is incorrect; display hint */
         if ( guess < x ) {
            printf( "Too low. Try again.\n? " );
         } /* end if */
         else {
            printf( "Too high. Try again.\n? " );
         } /* end else */

         scanf( "%d", &guess );
         total++;
      } /* end while */

```

```
         printf( "\nExcellent! You guessed the number!\n" );

         /* determine if user knows "secret" */
         if ( total < 10 ) {
            printf( "Either you know the secret or you got lucky!\n" );
         } /* end if */
         else if ( total == 10 ) {
            printf( "Ahah! You know the secret!\n" );
         } /* end else if */
         else {
            printf( "You should be able to do better!\n\n" );
         } /* end else */

         /* prompt for another game */
         printf( "Would you like to play again?\n" );
         printf( "Please type ( 1=yes, 2=no )? " );
         scanf( "%d", &response );
      } while ( response == 1 ); /* end do...while */

   } /* end function guessGame */
```

```
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 high. Try again.
? 125
Too high. Try again.
? 62
Too high. Try again.
? 31
Too low. Try again.
? 46
Too high. Try again.
? 39
Too low. Try again.
? 42

Excellent! You guessed the number!
Either you know the secret or you got lucky!
Would you like to play again?
Please type ( 1=yes, 2=no )? 2
```

5.37 Write a recursive function `power( base, exponent )` that when invoked returns

$$\texttt{base}^{\texttt{exponent}}$$

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

$$\texttt{base}^{\texttt{exponent}} = \texttt{base} * \texttt{base}^{\texttt{exponent - 1}}$$

and the terminating condition occurs when `exponent` is equal to `1` because

$$\texttt{base}^{1} = \texttt{base}$$

ANS:

```c
/* Exercise 5.37 Solution */
#include <stdio.h>

long power( long base, long exponent ); /* function prototype */

int main()
{
   long b; /* base */
   long e; /* exponent */

   printf( "Enter a base and an exponent: " );
   scanf( "%ld%ld", &b, &e );

   /* calculate and display b raised to the e power */
   printf( "%ld raised to the %ld is %ld\n", b, e, power( b, e ) );

   return 0; /* indicate successful termination */

} /* end main */

/* power recursively calculates base raised to the exponent
   assume exponent >= 1 */
long power( long base, long exponent )
{

   /* base case: exponent equals 1, return base */
   if ( exponent == 1 ) {
      return base;
   } /* end if */
   else { /* recursive step */
      return base * power( base, exponent - 1 );
   } /* end else */

} /* end function power */
```

```
Enter a base and an exponent: 5 10
5 raised to the 10 is 9765625
```

5.38 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 *non-recursive* function `fibonacci( n )` that calculates the nth 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. Let the program loop until it fails because of an excessively high value.

ANS:

```
/* Exercise 5.38 Part A Solution */
#include <stdio.h>
#define MAX 23   /* the maximum number for which the  */
                 /* fibonacci value can be calculated */
                 /* on 2-byte integer systems         */
int fibonacci( int n );

int main()
{
   int loop; /* loop counter */

   /* calculate and display Fibonacci value for 0 to MAX */
   for ( loop = 0; loop <= MAX; loop++ ) {
      printf( "fibonacci( %d ) = %d\n", loop, fibonacci( loop ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* fibonacci nonrecursively calculates nth Fibonacci number */
int fibonacci( int n )
{
   int j;          /* loop counter */
   int fib[ MAX ]; /* define array of size MAX */

   fib[ 0 ] = 0;
   fib[ 1 ] = 1;

   /* loop to find nth Fibonacci value */
   for ( j = 2; j <= n; j++ ) {
      fib[ j ] = fib[ j - 1 ] + fib[ j - 2 ];
   } /* end for */

   return fib[ n ]; /* return nth Fibonacci value */

} /* end function fibonacci */
```

```
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 5.38 Part B Solution */
#include <stdio.h>
#define SIZE 100

double fibonacci( int n ); /* function prototype */

int main()
{
   int loop; /* loop counter */

   /* loop SIZE times and calculate Fibonacci values */
   for ( loop = 0; loop < SIZE; loop++ ) {
      printf( "fibonacci( %d ) = %.1f\n", loop, fibonacci( loop ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* fibonacci nonrecursively calculates nth Fibonacci number */
double fibonacci( int n )
{
   int j;              /* loop counter */
   double fib[ SIZE ]; /* define double array of size SIZE */

   fib[ 0 ] = 0.0;
   fib[ 1 ] = 1.0;

   /* loop to find nth Fibonacci value */
   for ( j = 2; j <= n; j++ ) {
      fib[ j ] = fib[ j - 1 ] + fib[ j - 2 ];
   } /* end for */

   return fib[ n ]; /* return nth Fibonacci value */

} /* end function fibonacci */
```

```
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( 8 ) = 21.0
fibonacci( 9 ) = 34.0
fibonacci( 10 ) = 55.0
fibonacci( 11 ) = 89.0
fibonacci( 12 ) = 144.0
.
.
.
fibonacci( 96 ) = 51680708854858326000.0
fibonacci( 97 ) = 83621143489848426000.0
fibonacci( 98 ) = 135301852344706760000.0
fibonacci( 99 ) = 218922995834555200000.0
```

5.39 *(Towers of Hanoi)* Every budding computer scientist must grapple with certain classic problems, and the Towers of Hanoi (see Fig. 5.19) 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 stack 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 the disks. 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 disk-to-disk peg 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 (and hence the recursion) as follows:

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.

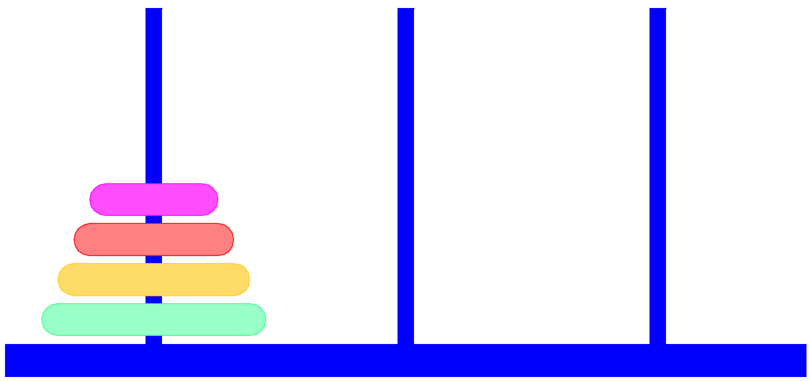


Fig. 5.18 The Towers of Hanoi for the case with four disks.

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

ANS:

```c
/* Exercise 5.39 solution */
#include <stdio.h>

/* function prototype */
void tower( int c, int start, int end, int temp );

int main()
{
   int n; /* number of disks */

   printf( "Enter the starting number of disks: " );
   scanf( "%d", &n );

   /* print instructions for moving disks from
      peg 1 to peg 3 using peg 2 for temporary storage */
   tower( n, 1, 3, 2 );

   return 0; /* indicate successful termination */

} /* end main */

/* tower recursively prints instructions for moving disks
   from start peg to end peg using temp peg for temporary storage */
void tower( int c, int start, int end, int temp )
{

   /* base case */
   if ( c == 1 ) {
      printf( "%d --> %d\n", start, end );
      return;
   } /* end if */

   /* move c - 1 disks from start to temp */
   tower( c - 1, start, temp, end );

   /* move last disk from start to end    */
   printf( "%d --> %d\n", start, end );

   /* move c - 1 disks from temp to end   */
   tower( c - 1, temp, end, start );
} /* end function tower */
```

```
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
```

5.40 Any program that can be implemented recursively can be implemented iteratively, although sometimes with considerably more difficulty and considerably 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 5.39. Investigate issues of performance, clarity, and your ability to demonstrate the correctness of the programs.

5.41 (*Visualizing Recursion*) It is interesting to watch recursion "in action." Modify the factorial function of Fig. 5.14 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.

ANS: *Note*: The `printf` in function `printRecursion` uses the conversion specification `%*d`. The `*` enables the programmer to specify the field width as a variable argument in the printf. In this case variable `n` is used as the field width, and its value is output.

```c
/* Exercise 5.41 Solution */
#include <stdio.h>

long factorial( long number ); /* function prototype */
void printRecursion( int n );  /* function prototype */

int main()
{
   int i; /* loop counter */

   /* calculate factorial( i ) and display result */
   for ( i = 0; i <= 10; i++ ) {
      printf( "%2d! = %ld\n", i, factorial( i ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* recursive definition of function factorial */
long factorial( long number )
{

   /* base case */
   if ( number <= 1 ) {
      return 1;
   } /* end if */
   else { /* recursive step */
      printRecursion( number ); /* add outputs and indentation */
      return ( number * factorial( number - 1 ) );
   } /* end else */

} /* end function factorial */

/* printRecursion adds outputs and indentation to help
   visualize recursion */
void printRecursion( int n )
{
   printf( "number = %*d\n", n, n );
} /* end function printRecursion */
```

```
 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 =         9
number =        8
number =       7
number =      6
number =     5
number =    4
number =   3
number =  2
 9! = 362880
number =         10
number =         9
number =        8
number =       7
number =      6
number =     5
number =    4
number =   3
number =  2
10! = 3628800
```

5.42 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 remainder operator.

ANS:

```c
/* Exercise 5.42 Solution */
#include <stdio.h>

/* function prototype */
unsigned int gcd( unsigned int xMatch, unsigned int yMatch );

int main()
{
   unsigned int x;     /* first integer */
   unsigned int y;     /* second integer */
   unsigned int gcDiv; /* greatest common divisor of x and y */

   printf( "Enter two integers: " );
   scanf( "%u%u", &x, &y );

   gcDiv = gcd( x, y );

   printf( "Greatest common divisor of %u and %u is %u\n",
          x, y, gcDiv );

   return 0; /* indicate successful termination */

} /* end main */

/* gcd recursively finds greatest common divisor
   of xMatch and yMatch */
unsigned int gcd( unsigned int xMatch, unsigned int yMatch )
{

   /* base case */
   if ( yMatch == 0 ) {
      return xMatch;
   } /* end if */
   else { /* recursive step */
      return gcd( yMatch, xMatch % yMatch );
   } /* end else */

} /* end function gcd */
```

```
Enter two integers: 10112 50500
Greatest common divisor of 10112 and 50500 is 4
```

5.43 Can `main` be called recursively? Write a program containing a function `main`. Include `static` local variable `count` initialized to 1. Postincrement and print the value of `count` each time `main` is called. Run your program. What happens?

ANS:

```
/* Exercise 5.43 Solution */
#include <stdio.h>

int main()
{
   static int count = 1; /* static local variable count */

   printf( "%d\n", count );
   count++;

   main(); /* recursively call int main() */

   return 0; /* indicate successful termination */

} /* end main */
```

```
1
2
3
4
5
6
7
8
9
10
...
```

5.44 Exercises 5.32 through 5.34 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.

ANS:

```c
/* Exercise 5.44 Part A Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int randValue( int level );     /* function prototype */
void multiplication( void );    /* function prototype */
void correctMessage( void );    /* function prototype */
void incorrectMessage( void ); /* function prototype */

int main()
{
   srand( time( NULL ) ); /* seed random number generator */
   multiplication(); /* being multiplication practice */

   return 0; /* indicate successful termination */

} /* end main */

/* randValue generates random numbers based on grade level */
int randValue( int level )
{

   /* level determines size of random number */
   switch ( level ) {

      case 1:
         return rand() % 10;

      case 2:
         return rand() % 100;

      case 3:
         return rand() % 1000;

      default:
         return rand() % 10;
   } /* end switch */

} /* end function randValue */

/* multiplication produces pairs of random numbers and
   prompts user for product; level determines size of numbers */
void multiplication( void )
{
   int i;                    /* loop counter */
   int x;                    /* first factor */
   int y;                    /* second factor */
   int gradeLevel;           /* grade-level capability */
   int right = 0;            /* total number of right responses */
   int wrong = 0;            /* total number of wrong responses */
   unsigned int response; /* user response for product */

   printf( "Enter the grade-level ( 1 to 3 ): " );
   scanf( "%d", &gradeLevel );

   /* loop 10 times */
   for ( i = 1; i <= 10; i++ ) {

      /* generate random numbers depending on level */
      x = randValue( gradeLevel );
```

```
      y = randValue( gradeLevel );

      printf( "How much is %d times %d? ", x, y );
      scanf( "%u", &response );

      /* loop while response is incorrect */
      while ( response != x * y ) {
         ++wrong; /* update total number of wrong answers */
         incorrectMessage();
         scanf( "%u", &response );
      } /* end while */

      ++right; /* update total number of right answers */
      correctMessage();
   } /* end for */

   /* if < 75% right */
   if ( ( double ) right / ( right + wrong) < .75 ) {
      printf( "Please ask your instructor for extra help.\n" );
   } /* end if */

   printf( "That's all for now. Bye.\n" );
} /* end function multiplication */

/* correctMessage randomly chooses response to correct answer */
void correctMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      case 0:
         printf( "Very good!\n\n" );
         break; /* exit switch */

      case 1:
         printf( "Excellent!\n\n" );
         break; /* exit switch */

      case 2:
         printf( "Nice work!\n\n" );
         break; /* exit switch */

      case 3:
         printf( "Keep up the good work!\n\n" );
         break; /* exit switch */
   } /* end switch */

} /* end function correctMessage */

/* incorrectMessage randomly chooses response to incorrect answer */
void incorrectMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      case 0:
         printf( "No. Please try again.\n? " );
         break; /* exit switch */

      case 1:
         printf( "Wrong. Try once more.\n? " );
         break; /* exit switch */

      case 2:
         printf( "Don't give up!\n? " );
         break; /* exit switch */
```

```
      case 3:
         printf( "No. Keep trying.\n? " );
         break; /* exit switch */
   } /* end switch */

} /* end function incorrectMessage */
```

```
Enter the grade-level ( 1 to 3 ): 1
How much is 6 times 0? 0
Keep up the good work!

How much is 6 times 3? 18
Keep up the good work!
...
```

```
Enter the grade-level ( 1 to 3 ): 2
How much is 5 times 63? 315
Excellent!

How much is 29 times 13? 392
No. Please try again.
? 377
Excellent!
...
```

```
Enter the grade-level ( 1 to 3 ): 3
How much is 799 times 343? 274057
Keep up the good work!

How much is 201 times 349? 0
Don't give up!
...
```

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.

ANS:

```
/* Exercise 5.44 Part B Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int menu( void );               /* function prototype */
void arithmetic( void );        /* function prototype */
void correctMessage( void );    /* function prototype */
void incorrectMessage( void ); /* function prototype */

int main()
{
   srand( time( NULL ) ); /* seed random number generator */
   arithmetic(); /* begin arithmetic process */

   return 0; /* indicate successful termination */

} /* end main */

/* menu displays user menu of choices */
int menu( void )
{
   int choice; /* user's choice */

   /* display menu and read user's choice */
   do {
      printf( "Choose type of problem to study.\n" );
      printf( "Enter: 1 for addition, 2 for subtraction\n" );
      printf( "Enter: 3 for multiplication, 4 for division\n" );
      printf( "Enter: 5 for a combination of 1 through 4\n " );
      printf( "? " );
      scanf( "%d", &choice );
   } while ( choice < 1 || choice > 5 ); /* end do...while */

   return choice; /* return user's choice */

} /* end function menu */

/* incorrectMessage randomly chooses response to incorrect answer */
void incorrectMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      case 0:
         printf( "No. Please try again.\n? " );
         break; /* exit switch */

      case 1:
         printf( "Wrong. Try once more.\n? " );
         break; /* exit switch */

      case 2:
         printf( "Don't give up!\n? " );
         break; /* exit switch */

      case 3:
         printf( "No. Keep trying.\n? " );
         break; /* exit switch */
   } /* end switch */

} /* end function incorrectMessage */
```

```
/* correctMessage randomly chooses response to correct answer */
void correctMessage( void )
{

   /* generate random number between 0 and 3 */
   switch ( rand() % 4 ) {

      case 0:
         printf( "Very good!\n\n" );
         break; /* exit switch */

      case 1:
         printf( "Excellent!\n\n" );
         break; /* exit switch */

      case 2:
         printf( "Nice work!\n\n" );
         break; /* exit switch */

      case 3:
         printf( "Keep up the good work!\n\n" );
         break; /* exit switch */
   } /* end switch */

} /* end function correctMessage */


void arithmetic( void )
{
   int i;           /* loop counter */
   int x;           /* first number */
   int y;           /* second number */
   int response;    /* user response for product */
   int answer;      /* correct answer */
   int selection;   /* menu selection */
   int right = 0;   /* total correct responses */
   int wrong = 0;   /* total incorrect responses */
   int type;        /* type of problems chosen */
   int problemMix;  /* random choice of type of problem */
   char operator;   /* arithmetic operator */

   selection = menu();
   type = selection;

   /* loop 10 times */
   for ( i = 1; i <= 10; i++ ) {
      x = rand() % 10; /* generate first random number */
      y = rand() % 10; /* generate second random number */

      /* if option 5, randomly select type */
      if ( selection == 5 ) {
         problemMix = 1 + rand() % 4;
         type = problemMix;
      } /* end if */

      /* generate answer and define operator depending on option */
      switch ( type ) {

         /* option 1: addition */
         case 1:
            operator = '+';
            answer = x + y;
            break; /* exit switch */

         /* option 2: subtraction */
         case 2:
            operator = ‘‘-';
```

```
            answer = x - y;
            break; /* exit switch */

         /* option 3: multiplication */
         case 3:
            operator = '*';
            answer = x * y;
            break; /* exit switch */

         /* option 4: integer division */
         case 4:
            operator = '/';

            /* eliminate divide by zero error */
            if ( y == 0 ) {
               y = 1;
               answer = x / y;
            } /* end if */
            else {
               x *= y; /* create "nice" division */
               answer = x / y;
            } /* end else */

            break; /* exit switch */
      } /* end switch */

      printf( "How much is %d %c %d? ", x, operator, y );

      scanf( "%d", &response );

      /* while not correct answer */
      while ( response != answer ) {
         ++wrong;
         incorrectMessage();
         scanf( "%d", &response );
      } /* end while */

      ++right;
      correctMessage();
   } /* end for */

   /* if < 75% right, suggest help */
   if ( ( double ) right / ( right + wrong) < .75 ) {
      printf( "Please ask your instructor for extra help.\n" );
   } /* end if */

   printf( "That's all for now. Bye.\n" );
} /* end function arithmetic */
```

```
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 9 * 9? 81
Nice work!

How much is 3 - 1? 2
Keep up the good work!

How much is 1 * 3? 3
Nice work!
.
.
.
How much is 1 - 9? -8
Nice work!

That's all for now. Bye.
```

5.45 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`.

ANS:

```
/* Exercise 5.45 Solution */
#include <stdio.h>
#include <math.h>

/* function prototype */
double distance( double xOne, double yOne, double xTwo, double yTwo );

int main()
{
   double x1;   /* x coordinate of first point */
   double y1;   /* y coordinate of first point */
   double x2;   /* x coordinate of second point */
   double y2;   /* y coordinate of second point */
   double dist; /* distance between two points */

   /* prompt for first point coordinates */
   printf( "Enter the first point: " );
   scanf( "%lf%lf", &x1, &y1 );

   /* prompt for second point coordinates */
   printf( "Enter the second point: " );
   scanf( "%lf%lf", &x2, &y2 );

   dist = distance( x1, y1, x2, y2 ); /* calculate distance */

   printf( "Distance between ( %.2f, %.2f )"
           " and ( %.2f, %.2f ) is %.2f\n",
           x1, y1, x2, y2, dist );

   return 0; /* indicate successful termination */

} /* end main */

/* distance calculates distance between 2 points
   given by (xOne, yOne) and (xTwo, yTwo) */
double distance( double xOne, double yOne, double xTwo, double yTwo )
{
   double distance; /* distance between two points */

   distance = sqrt( pow( xOne - xTwo, 2 ) + pow( yOne - yTwo, 2 ) );

   return distance;

} /* end function distance */
```

```
Enter the first point: 3 4
Enter the second point: 0 0
Distance between ( 3.00, 4.00 ) and ( 0.00, 0.00 ) is 5.00
```

5.46 What does the following program do?

```
#include <stdio.h>

/* function main begins program execution */
int main()
{
   int c; /* variable to hold character input by user */

   if ( ( c = getchar() ) != EOF ) {
      main();
      printf( "%c", c );
   } /* end if */

   return 0; /* indicates successful termination */

} /* end main */
```

ANS: Inputs a character and recursively calls `main()` until the `EOF` character is entered. Every character entered is then output in reverse order.

```
a b c

c b a
```

5.47 What does the following program do?

```c
#include <stdio.h>

int mystery( int a, int b ); /* function prototype */

/* function main begins program execution */
int main()
{
   int x; /* first integer */
   int y; /* second integer */

   printf( "Enter two integers: " );
   scanf( "%d%d", &x, &y );

   printf( "The result is %d\n", mystery( x, y ) );

   return 0; /* indicates successful termination */

} /* end main */

/* Parameter b must be a positive integer
   to prevent infinite recursion */
int mystery( int a, int b )
{
   /* base case */
   if ( b == 1 ) {
      return a;
   } /* end if */
   else { /* recursive step */
      return a + mystery( a, b - 1 );
   } /* end else */

} /* end function mystery */
```

ANS: The problem mimics multiplication by adding up a, b times.

```
Enter two integers: 87 6
The result is 522
```

5.48 After you determine what the program of Exercise 5.47 does, modify the program to function properly after removing the restriction of the second argument being nonnegative.

ANS:

```c
/* Exercise 5.48 Solution */
#include <stdio.h>

int mystery( int a, int b ); /* function prototype */

int main()
{
   int x; /* first integer */
   int y; /* second integer */

   printf( "Enter two integers: " );
   scanf( "%d%d", &x, &y );

   printf( "The result is %d\n", mystery( x, y ) );

   return 0; /* indicate successful termination */

} /* end main */

/* mystery multiplies a * b using recursion */
int mystery( int a, int b )
{

   /* if a and b or just b are negative */
   if ( (  a < 0 && b < 0 ) || b < 0 ) {
      a *= -1; /* multiply a and b by -1 to make positive */
      b *= -1;
   } /* end if */

   /* base case */
   if ( b == 1 ) {
      return a;
   } /* end if */
   else { /* recursive step */
      return a + mystery( a, b - 1 );
   } /* end else */

} /* end function mystery */
```

```
Enter two integers: -97 6
The result is -582
```

```
Enter two integers: 97 -6
The result is -582
```

```
Enter two integers: -97 -6
The result is 582
```

5.49 Write a program that tests as many of the math library functions in Fig. 5.2 as you can. Exercise each of these functions by having your program print out tables of return values for a diversity of argument values.

ANS:

```c
/* Exercise 5.49 Solution */
#include <stdio.h>
#include <math.h>

int main()
{
   int loop;     /* integer loop counter */
   int count;    /* loop counter */
   double loop2; /* double loop counter */

   /* loop and test each math function */
   for ( count = 1; count < 14; count++) {

      /* test math function based on count */
      switch ( count ) {

         /* print table headers */
         case 1:
            printf( "funct    " );

            for ( loop = 1; loop < 6; loop++ ) {
               printf( "%10d ", loop );
            } /* end for */

            break; /* exit switch */

         /* display sqrt for range of values */
         case 2:
            printf( "\nsqrt()    " );

            for ( loop = 1; loop < 6; loop++ ) {
               printf( "%10.2lf ", sqrt( loop ) );
            } /* end for */

            break; /* exit switch */

         /* display exp for range of values */
         case 3:
            printf( "exp()     " );

            for ( loop = 1; loop < 6; loop++ ) {
               printf( "%10.2lf ", exp( loop ) );
            } /* end for */

            break; /* exit switch */

         /* display natural log for range of values */
         case 4:
            printf( "log()     " );

            for ( loop = 1; loop < 6; loop++ ) {
               printf( "%10.2lf ", log( loop ) );
            } /* end for */

            break; /* exit switch */

         /* display log base 10 for range of values */
         case 5:
            printf( "log10()   " );

            for ( loop = 1; loop < 6; loop++ ) {
               printf( "%10.2lf ", log10( loop ) );
            } /* end for */
```

```

            break; /* exit switch */

         /* display pow function, test with 2 as base */
         case 6:
            printf( "pow( 2,x )" );

            for ( loop = 1; loop < 6; loop++ ) {
               printf( "%10.2lf ", pow( 2, loop ) );
            } /* end for */

            break; /* exit switch */

         /* display table headers */
         case 7:
            printf( "\n\nfunct     " );

            for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
               printf( "%10.2lf ", loop2 );
            } /* end for */

            break; /* exit switch */

         /* display fabs for range of values */
         case 8:
            printf( "\nfabs()    " );

            for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
               printf( "%10.2lf ", fabs( loop2 ) );
            } /* end for */

            break; /* exit switch */

         /* display ceil for range of values */
         case 9:
            printf( "ceil()    " );

            for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
               printf( "%10.2lf ", ceil( loop2 ) );
            } /* end for */

            break; /* exit switch */

         /* display floor for range of values */
         case 10:
            printf( "floor()   " );

            for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
               printf( "%10.2lf ", floor( loop2 ) );
            } /* end for */

            break; /* exit switch */

         /* display sin for range of values */
         case 11:
            printf( "sin()     " );

            for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
               printf( "%10.2lf ", sin( loop2 ) );
            } /* end for */

            break; /* exit switch */

         /* display cos for range of values */
         case 12:
            printf( "cos()     " );

```

```
               for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
                  printf( "%10.2lf ", cos( loop2 ) );
               } /* end for */

               break; /* exit switch */

            /* display tan for range of values */
            case 13:
               printf( "tan()      " );

               for ( loop2 = -1.5; loop2 < 3.0; loop2 += 1.1 ) {
                  printf( "%10.2lf ", tan( loop2 ) );
               } /* end for */

               break; /* exit switch */
         } /* end switch */

         printf( "\n" );
      } /* end for */

      return 0; /* indicate successful termination */

   } /* end main */
```

```
funct              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


funct          -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
```

5.50 Find the error in each of the following program segments and explain how to correct it:

a)
```
double cube( float );    /* function prototype */
...
cube( float number )    /* function definition */
{
   return number * number * number;
}
```

ANS: Function definition is missing return type.

```
double cube( float );  /* function prototype */
...
double cube( float number )  /* function definition */
{
   return number * number * number;
}
```

b) `register auto int x = 7;`

ANS: Too many storage class definitions. Auto class definition is not necessary.

```
register int x = 7;  /* auto removed */
```

c) `int randomNumber = srand();`

ANS: `srand()` seeds the random number generator, and has a void return type. Function `rand()` produces random numbers.

```
int randomNumber = rand();
```

d)
```
double y = 123.45678;
int x;
x = y;
printf( "%f\n", (double) x );
```

ANS: Decimal value is lost when a `double` is assigned to an integer. Type-casting the `int` to `double` cannot bring back the original decimal value. Only `123.000000` can be printed.

```
double y = 123.45678;
double x;

x = y;
printf( "%f\n", x );
```

e)
```
double square( double number )
{
   double number;

   return number * number;
}
```

ANS: `number` is defined twice.

```
double square( double number )
{
   return number * number;
}
```

f)
```
int sum( int n )
{
   if ( n == 0 )
      return 0;
   else
      return n + sum( n );
}
```

ANS: Infinite recursion.

```
int sum( int n )
{
   if ( n == 0 )
      return 0;
   else
      return n + sum( n - 1 );
}
```

5.51 Modify the craps program of Fig. 5.10 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 if `bankBalance` 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!"`.

ANS:

```
/* Exercise 5.51 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* enumeration constants represent game status */
enum Status { CONTINUE, WON, LOST };

int rollDice( void ); /* function prototype */
enum Status craps( void ); /* function prototype */
void chatter( void ); /* function prototype */

int main()
{
   enum Status result;     /* result of current game */
   int wager = 0;          /* wager for current game */
   int bankBalance = 1000; /* current bank balance */

   srand( time( NULL ) ); /* seed random number generator */

   /* display current balance and prompt for wager */
   printf( "You have $%d in the bank.\n", bankBalance );
   printf( "Place your wager: " );
   scanf( "%d", &wager );

   /* loop while not valid wager */
   while( wager <= 0 || wager > 1000 ) {
      printf( "Please bet a valid amount.\n" );
      scanf( "%d", &wager );
   } /* end while */

   result = craps(); /* play game of craps */

   /* if player lost current game */
   if ( result == LOST ) {

      /* decrease balance by wager and display new balance */
      bankBalance -= wager;
      printf( "Your new bank balance is $%d\n", bankBalance );

      /* if balance is 0 */
      if ( bankBalance == 0 ) {
         printf( "Sorry. You are Busted! Thank You For Playing.\n" );
      } /* end if */

   } /* end if */
   else { /* player won game */

      /* increase balance by wager and display new balance */
      bankBalance += wager;
      printf( "Your new bank balance is $%d\n", bankBalance );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```c

/* roll dice, calculate sum and display results */
int rollDice( void )
{
   int die1;    /* first die value */
   int die2;    /* second die value */
   int workSum; /* sum of dice */

   die1 = 1 + rand() % 6; /* pick random die1 value */
   die2 = 1 + rand() % 6; /* pick random die2 value */
   workSum = die1 + die2; /* sum die1 and die2 */

   /* display results of this roll */
   printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );

   return workSum; /* return sum of dice */

} /* end function rollDice */

/* craps plays one game of craps, returns result of game */
enum Status craps( void )
{
   enum Status gameStatus; /* can contain CONTINUE, WON or LOST */
   int sum;                /* current roll of dice */
   int myPoint;            /* point value */

   sum = rollDice(); /* first roll of dice */

   /* determine game status and point based on sum of dice */
   switch ( sum ) {

      /* win on first roll */
      case 7:
      case 11:
         gameStatus = WON;
         chatter();
         break; /* exit switch */

      /* lose on first roll */
      case 2:
      case 3:
      case 12:
         gameStatus = LOST;
         chatter();
         break; /* exit switch */

      /* remember point */
      default:
         gameStatus = CONTINUE;
         myPoint = sum;
         printf( "Point is %d\n", myPoint );
         chatter();
         break; /* exit switch */
   } /* end switch */

   /* while game not complete */
   while ( gameStatus == CONTINUE ) {
      chatter();
      sum = rollDice(); /* roll dice again */

      /* determine game status */
      if ( sum == myPoint ) {
         gameStatus = WON; /* win by making point */
      } /* end if */
      else {

         if ( sum == 7 ) {
            gameStatus = LOST; /* lose by rolling 7 */
         } /* end if */
```

```
      } /* end else */

   } /* end while */

   /* display won or lost message and return status */
   if ( gameStatus == WON ) {
      printf( "Player wins\n" );
      return WON;
   } /* end if */
   else {
      printf( "Player loses\n" );
      return LOST;
   } /* end else */

} /* end function craps */

/* chatter displays messages at random to create "chatter" */
void chatter( void )
{
   int select; /* random number */

   select = 1 + rand() % 6;

   /* choose message at random */
   switch ( select ) {

      case 1:
         printf( "Oh, you're going for broke, huh?\n" );
         break; /* exit switch */

      case 2:
         printf( "Aw cmon, take a chance!\n" );
         break; /* exit switch */

      case 3:
         printf( "Hey, I think this guy is going to break the bank!!\n" );
         break; /* exit switch */

      case 4:
         printf( "You're up big. Now's the time to cash in your chips!\n" );
         break; /* exit switch */

      case 5:
         printf( "Way too lucky! Those dice have to be loaded!\n" );
         break; /* exit switch */

      case 6:
         printf( "Bet it all! Bet it all!\n" );
         break; /* exit switch */

      default:
         break; /* exit switch */
   } /* end switch */

} /* end function chatter */
```

```
You have $1000 in the bank.
Place your wager: 1000
Player rolled 4 + 5 = 9
Point is 9
You're up big. Now's the time to cash in your chips!
Oh, you're going for broke, huh?
Player rolled 5 + 6 = 11
Hey, I think this guy is going to break the bank!!
Player rolled 3 + 1 = 4
Bet it all! Bet it all!
Player rolled 5 + 5 = 10
Aw cmon, take a chance!
Player rolled 6 + 6 = 12
Bet it all! Bet it all!
Player rolled 2 + 1 = 3
Hey, I think this guy is going to break the bank!!
Player rolled 5 + 6 = 11
Hey, I think this guy is going to break the bank!!
Player rolled 2 + 1 = 3
Aw cmon, take a chance!
Player rolled 2 + 4 = 6
You're up big. Now's the time to cash in your chips!
Player rolled 2 + 3 = 5
Oh, you're going for broke, huh?
Player rolled 6 + 3 = 9
Player wins
Your new bank balance is $2000
```

6

C Arrays: Solutions

EXERCISES

6.6 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 parentheses is called a(n) ________.

ANS: subscript.

d) The names of the five elements of array `p` are ________, ________, ________, ________ and ________.

ANS: `p[ 0 ], p[ 1 ], p[ 2 ], p[ 3 ], p[ 4 ].`

e) The contents of a particular element of an array is called the ________ of that element.

ANS: value.

f) Naming an array, stating its type and specifying the number of elements in the array is called ________ the array.

ANS: defining.

g) The process of placing the elements of an array into either ascending or descending order is called ________.

ANS: sorting.

h) 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.

i) An *m*-by-*n* array contains ________ rows, ________ columns and ________ elements.

ANS: `m, n, m * n.`

j) The name of the element in row 3 and column 5 of array `d` is ________.

ANS: `d[ 3 ][ 5 ].`

6.7 State which of the following are *true* and which are *false*. If *false*, explain why.

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. We specify the name and the subscript of the element.

b) An array definition reserves space for the array.

ANS: True.

c) To indicate that 100 locations should be reserved for integer array `p`, the programmer writes the definition

```
p[ 100 ];
```

ANS: True.

d) A C program that initializes the elements of a 15-element array to zero must contain one `for` statement.
ANS: False. The elements of an array can be initialized in the definition.
e) A C program that totals the elements of a double-subscripted array must contain nested `for` statements.
ANS: False. It is possible to total the elements of a double-subscripted array by enumerating all the elements in an assignment statement.
f) The mean, median and mode of the following set of values are 5, 6 and 7, respectively: 1, 2, 5, 6, 7, 7, 7.
ANS: True.

6.8 Write statements to accomplish each of the following:
a) Display the value of the seventh element of character array `f`.
ANS: `printf( "%c\n", f[ 6 ] );`
b) Input a value into element 4 of single-subscripted floating-point array `b`.
ANS: `scanf( "%f", &b[ 4 ] );`
c) Initialize each of the 5 elements of single-subscripted integer array `g` to `8`.
ANS:
```
for ( loop = 0; loop <= 4; loop++ )
   g[ loop ] = 8;
```
d) Total the elements of floating-point array `c` of 100 elements.
ANS:
```
for ( loop = 0; loop <= 99; loop++ )
   sum += c[ loop ];
```
e) Copy array `a` into the first portion of array `b`. Assume `double a[ 11 ], b[ 34 ];`
ANS:
```
for ( loop = 0; loop <= 10; loop++ )
   b[ loop ] = a[ loop ];
```
f) Determine and print the smallest and largest values contained in 99-element floating-point array `w`.
ANS:
```
smallest = largest = w[ 0 ];

for ( loop = 1; loop <= 98; loop++ )
   if ( w[ loop ] < smallest )
      smallest = w[ loop ];
   else if ( w[ loop ] > largest )
      largest = w[ loop ];
```

6.9 Consider a 2-by-5 integer array `t`.
a) Write a definition for `t`.
ANS: `int t[ 2 ][ 5 ];`
b) How many rows does `t` have?
ANS: `2`
c) How many columns does `t` have?
ANS: `5`
d) How many elements does `t` have?
ANS: `10`
e) Write the names of all the elements in the second row of `t`.
ANS: `t[ 1 ][ 0 ], t[ 1 ][ 1 ], t[ 1 ][ 2 ], t[ 1 ][ 3 ], t[ 1 ][ 4 ].`
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[ 1 ][ 2 ] = 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[ 0 ][ 3 ] = 0;
t[ 0 ][ 4 ] = 0;
t[ 1 ][ 0 ] = 0;
```

```
t[ 1 ][ 1 ] = 0;
t[ 1 ][ 2 ] = 0;
t[ 1 ][ 3 ] = 0;
t[ 1 ][ 4 ] = 0;
```

i) Write a nested `for` statement that initializes each element of `t` to zero.

ANS:

```
for ( i = 0; i <= 1; i++ )
   for ( j = 0; j <= 4; j++)
      t[ i ][ j ] = 0;
```

j) Write a statement that inputs the values for the elements of `t` from the terminal.

ANS:

```
for ( i = 0; i <= 1; i++ )
   for ( j = 0; j <= 4; j++) {
      printf( "Enter an integer: " );
      scanf( "%d", &t[ i ][ j ] )
   }
```

k) Write a series of statements that determine and print the smallest value in array `t`.

ANS:

```
smallest = t[ 0 ][ 0 ];

for ( i = 0; i <= 1; i++ )
   for ( j = 0; j <= 4; j++)
      if ( t[ i ][ j ] < smallest )
         smallest = t[ i ][ j ];

printf( " smallest is %d\n", smallest );
```

l) Write a statement that displays the elements of the first row of `t`.

ANS:

```
for ( i = 0; i <= 4; i++ )
   printf( "%d ", t[ 0 ][ i ] );
```

m) Write a statement that totals the elements of the fourth column of `t`.

ANS: `sum = t[ 0 ][ 3 ] + t[ 1 ][ 3 ];`

n) Write a series of statements that print the array `t` in tabular format. List the column subscripts as headings across the top and list the row subscripts at the left of each row.

ANS:

```
printf( "  0\t1\t2\t3\t4\n" );

for ( i = 0; i <= 1; i++ ) {
   printf( "%d ", i );

   for ( j = 0; j <= 4; j++ )
      printf( "%d\t", t[ i ][ j ] );

   printf( "\n" );
}
```

6.10 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 $3000 in sales in a week receives $200 plus 9 percent of $3000, or a total of $470. Write a C 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

ANS:

```c
/* Exercise 6.10 Solution */
#include <stdio.h>

int main()
{
   int salaries[ 11 ] = { 0 }; /* array to hold salary counts */
   int sales;                  /* current employee's sales */
   double salary;              /* current employee's salary */
   double i = 0.09;            /* commission percentage */

   /* prompt user for gross sales */
   printf( "Enter employee gross sales ( -1 to end ): " );
   scanf( "%d", &sales );

   /* while sentinel value not read from user */
   while ( sales != -1 ) {

      /* calculate salary based on sales */
      salary = 200.0 + sales * i;
      printf( "Employee Commission is $%.2f\n", salary );

      /* update appropriate salary range */
      if ( salary >= 200 && salary < 1000 ) {
         ++salaries[ ( int ) salary / 100 ];
      } /* end if */
      else if ( salary >= 1000 ) {
         ++salaries[ 10 ];
      } /* end else if */

      /* prompt user for another employee sales amount */
      printf( "\nEnter employee gross sales ( -1 to end ): " );
      scanf( "%d", &sales );
   } /* end while */

   /* display table of ranges and employees in each range */
   printf( "\nEmployees in the range:\n" );
   printf( "$200-$299 : %d\n", salaries[ 2 ] );
   printf( "$300-$399 : %d\n", salaries[ 3 ] );
   printf( "$400-$499 : %d\n", salaries[ 4 ] );
   printf( "$500-$599 : %d\n", salaries[ 5 ] );
   printf( "$600-$699 : %d\n", salaries[ 6 ] );
   printf( "$700-$799 : %d\n", salaries[ 7 ] );
   printf( "$800-$899 : %d\n", salaries[ 8 ] );
   printf( "$900-$999 : %d\n", salaries[ 9 ] );
   printf( "Over $1000: %d\n", salaries[ 10 ] );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter employee gross sales ( -1 to end ): 3000
Employee Commission is $470.00

Enter employee gross sales ( -1 to end ): 1000
Employee Commission is $290.00

Enter employee gross sales ( -1 to end ): 10000
Employee Commission is $1100.00

Enter employee gross sales ( -1 to end ): 8000
Employee Commission is $920.00

Enter employee gross sales ( -1 to end ): 200
Employee Commission is $218.00

Enter employee gross sales ( -1 to end ): 7000
Employee Commission is $830.00

Enter employee gross sales ( -1 to end ): -1

Employees in the range:
$200-$299 : 2
$300-$399 : 0
$400-$499 : 1
$500-$599 : 0
$600-$699 : 0
$700-$799 : 0
$800-$899 : 1
$900-$999 : 1
Over $1000: 1
```

6.11 The bubble sort presented in Fig. 6.15 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 has 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.

ANS:

```
/* Exercise 6.11 Solution */
#include <stdio.h>
#define MAX 10

int main()
{

   /* initialize array a with initializer list */
   int a[ MAX ] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
   int i;    /* loop counter */
   int pass; /* loop counter */
   int hold; /* temporary variable for swapping */
   int swap; /* flag to break loop if elements are sorted */

   printf( "Data items in original order\n" );

   /* display original, unsorted array */
   for ( i = 0; i < MAX; i++ ) {
      printf( "%4d", a[ i ] );
   } /* end for */

   printf( "\n\n" );

   /* begin sorting the array */
   for ( pass = 1; pass < MAX; pass++ ) {
      swap = 0;

      /* traverse and compare unsorted part of array */
      for ( i = 0; i < MAX - pass; i++ ) {

         /* compare adjacent array elements */
         if ( a[ i ] > a[ i + 1 ] ) {
            swap = 1; /* raise flag if any elements are swapped */
            hold = a[ i ];
            a[ i ] = a[ i + 1 ];
            a[ i + 1 ] = hold;
         } /* end if */

      } /* end for */

      printf( "After Pass %d: ", pass );

      /* display array after each pass */
      for ( i = 0; i <= MAX-pass; i++ ) {
         printf( "  %d", a[ i ] );
      } /* end for */

      printf( "\n" );

```

```
         /* break loop if array is sorted */
         if ( !swap ) {
            break;
         } /* end if */

      } /* end for */

      printf( "\nData items in ascending order\n" );

      /* display array in sorted order */
      for ( i = 0; i < 10; i++ ) {
         printf( "%4d", a[ i ] );
      } /* end for */

      printf( "\n" );

      return 0; /* indicate successful termination */

   } /* end main */
```

```
Data items in original order
  10   9   8   7   6   5   4   3   2   1

After Pass 1:   9  8  7  6  5  4  3  2  1  10
After Pass 2:   8  7  6  5  4  3  2  1  9
After Pass 3:   7  6  5  4  3  2  1  8
After Pass 4:   6  5  4  3  2  1  7
After Pass 5:   5  4  3  2  1  6
After Pass 6:   4  3  2  1  5
After Pass 7:   3  2  1  4
After Pass 8:   2  1  3
After Pass 9:   1  2

Data items in ascending order
   1   2   3   4   5   6   7   8   9  10
```

6.12 Write single statements that perform each of the following single-subscripted array operations:

a) Initialize the 10 elements of integer array `counts` to zeros.

ANS:

```
for ( i = 0; i <= 9; i++ )
   counts[ i ] = 0;
```

b) Add 1 to each of the 15 elements of integer array `bonus`.

ANS:

```
for ( i = 0; i <= 14; i++ )
   ++bonus[ i ];
```

c) Read the 12 values of floating-point array `monthlyTemperatures` from the keyboard.

ANS:

```
for ( i = 0; i <= 11; i++ ) {
   printf( "Enter a temperature: " );
   scanf( "%f", &monthlyTemperatures[ i ] );
}
```

d) Print the 5 values of integer array `bestScores` in column format.

ANS:

```
for ( i = 0; i <= 4; i++ ) {
   printf( "%d\t", bestScores[ i ] );
```

6.13 Find the error(s) in each of the following statements:

a) Assume: `char str[ 5 ];`

```
scanf( "%s", str );     /* User types hello */
```

ANS: `str` needs a minimum length of 6; one element for each letter in `hello` and an element for the terminating `NULL` character.

b) Assume: `int a[ 3 ];`

```
printf( "$d  %d  %d\n", a[ 1 ], a[ 2 ], a[ 3 ] );
```

ANS: `printf( "%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );`

c) `double f[ 3 ] = { 1.1, 10.01, 100.001, 1000.0001 };`

ANS: Too many variables defined.

```
double f[ 3 ] = { 1.1, 10.01, 100.01 };
```

d) Assume: `double d[ 2 ][ 10 ];`

```
d[ 1, 9 ] = 2.345;
```

ANS: `d[ 1 ][ 9 ] = 2.345;`

6.14 Modify the program of Fig. 6.16 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.

ANS:

```
/* Exercise 6.14 Solution */
#include <stdio.h>
#define SIZE 100

void mean( int answer[] );              /* function prototype */
void median( int answer[] );            /* function prototype */
void mode( int freq[], int answer[] ); /* function prototype */

int main()
{

   /* array of responses */
   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[ 10 ] = { 0}; /* array of response frequencies */

   mean( response ); /* process mean */
   median( response ); /* process median */
   mode( frequency, response ); /* process mode */

   return 0; /* indicates successful termination */

} /* end main */

/* calculate average of all response values */
void mean( int answer[] )
{
   int j;         /* loop counter */
   int total = 0; /* total of all response values */

   printf( "%s\n%s\n%s\n", "******", " Mean", "******" );

   /* total response values */
   for ( j = 0; j <= SIZE - 1; j++ ) {
      total += answer[ j ];
   } /* end for */

   /* output results */
   printf( "The mean is the average value of the data\n" );
   printf( "items. The mean is equal to the total of\n" );
   printf( "all the data items divided by the number\n" );
   printf( "of data items ( %d ). ,", SIZE );
   printf( "The mean value for this run is: " );
   printf( "%d / %d = %.4f\n\n", total, SIZE, ( double ) total / SIZE );
} /* end function mean */

/*sort an array and determine median element's value */
void median( int answer[] )
{
   int loop;     /* loop counter */
   int pass;     /* loop counter */
   int hold;     /* temporary variable for swapping */
   int firstRow; /* flag to indicate first row of array */

   printf( "\n%s\n%s\n%s\n", "******", "Median", "******" );
```

```
   printf( "The unsorted array of responses is\n" );

   /* display unsorted array */
   for ( loop = 0, firstRow = 1; loop <= SIZE - 1; loop++ ) {

      /* start a new line */
      if ( loop % 20 == 0 && !firstRow ) {
         printf( "\n" );
      } /* end if */

      printf( "%2d", answer[ loop ] );
      firstRow = 0;
   } /* end for */

   printf( "\n\n" );

   /* sort array */
   for ( pass = 0; pass <= SIZE - 2; pass++ ) {

      /* compare elements and swap if necessary */
      for ( loop = 0; loop <= SIZE - 2; loop++ ) {

         /* swap elements */
         if ( answer[ loop ] > answer[ loop + 1 ] ) {
            hold = answer[ loop ];
            answer[ loop ] = answer[ loop + 1 ];
            answer[ loop + 1 ] = hold;
         } /* end if */

      } /* end for */

   } /* end for */

   printf( "The sorted array is\n" );

   /* display sorted array */
   for ( loop = 0, firstRow = 1; loop <= SIZE - 1; loop++ ) {

      /* start a new line */
      if ( loop % 20 == 0 && !firstRow ) {
         printf( "\n" );
      } /* end if */

      printf( "%2d", answer[ loop ] );
      firstRow = 0;
   } /* end for */

   printf( "\n\n" );

   /* even number of elements */
   if ( SIZE % 2 == 0 ) {
      printf( "The median is the average of elements %d",
         ( SIZE + 1 ) / 2 );
      printf( " and %d of", 1 + ( SIZE + 1 ) / 2 );
      printf( " the sorted %d element array.\n", SIZE );
      printf( "For this run the median is " );
      printf( "%.1f\n\n", ( double )( answer[ ( SIZE + 1 ) / 2 ] +
         answer[ ( SIZE + 1 ) / 2 + 1 ] ) / 2 );
   } /* end if */
   else { /* odd number of elements  */
      printf( "The median is element %d of ", ( SIZE + 1 ) / 2 );
      printf( "the sorted %d element array.\n", SIZE );
      printf( "For this run the median is " );
      printf( "%d\n\n", answer[ ( SIZE + 1 ) / 2 - 1 ] );
   } /* end else */

} /* end function median */

```

```c
/* determine most frequent response */
void mode( int freq[], int answer[] )
{
   int rating;             /* loop counter */
   int loop;               /* loop counter */
   int largest = 0;        /* represents largest frequency */
   int array[ 10 ] = { 0}; /* array used to hold largest frequencies */
   int count = 0;          /* flag to count number of modes */

   printf( "\n%s\n%s\n%s\n", "******", " Mode", "******" );

   /* set all frequencies to 0 */
   for ( rating = 1; rating <= 9; rating++ ) {
      freq[ rating ] = 0;
   } /* end for */

   /* traverse array and increment corresponding frequency */
   for ( loop = 0; loop <= SIZE - 1; loop++ ) {
      ++freq[ answer[ loop ] ];
   } /* end for */

   printf( "%s%11s%19s\n\n", "Response", "Frequency", "Histogram" );
   printf( "%54s\n", "1    1    2    2" );
   printf( "%54s\n\n", "5    0    5    0    5" );

   /* display values and frequency */
   for ( rating = 1; rating <= 9; rating++ ) {
      printf( "%8d%11d          ", rating, freq[ rating ] );

      /* test if current frequency is greater than largest frequency */
      if ( freq[ rating ] > largest ) {
         largest = freq[ rating ];

         /* set values of array to 0 */
         for ( loop = 0; loop < 10; loop++ ) {
            array[ loop ] = 0;
         } /* end for */

         /* add new largest frequency to array */
         array[ rating ] = largest;
         ++count;
      } /* end if */
      /* if current frequency equals largest, add current to array */
      else if ( freq[ rating ] == largest ) {
         array[ rating ] = largest;
         ++count;
      } /* end else if */

      /* display histogram */
      for ( loop = 1; loop <= freq[ rating ]; loop++ ) {
         printf( "*" );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );

   /* if more than one mode */
   if ( count > 1 ) {
      printf( "The modes are:  " );
   } /* end if */
   else { /* only one mode */
      printf( "The mode is: " );
   } /* end else */

   /* display mode(s) */
   for ( loop = 1; loop <= 9; loop++ ) {
```

```c

      if ( array[ loop ] != 0 ) {
         printf( "%d with a frequency of %d\n\t\t", loop, array[ loop ] );
      } /* end if */

   } /* end for */

   printf( "\n" );
} /* end function mode */
```

```
******
 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 = 6.6200


******
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

       1          5          *****
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         23          ***********************
       9         20          ********************

The modes are:  7 with a frequency of 23
                8 with a frequency of 23
```

6.15 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.

ANS:

```c
/* Exercise 6.15 Solution */
#include <stdio.h>
#define MAX 20

int main()
{
   int a[ MAX ] = { 0 }; /* array for user input */
   int i;                /* loop counter */
   int j;                /* loop counter */
   int k = 0;            /* number of values currently entered */
   int duplicate;        /* flag for duplicate values */
   int value;            /* current value */

   printf( "Enter 20 integers between 10 and 100:\n" );

   /* get 20 integers from user */
   for ( i = 0; i <= MAX - 1; i++ ) {
      duplicate = 0;
      scanf( "%d", &value );

      /* test if integer is a duplicate */
      for ( j = 0; j < k; j++ ) {

         /* if duplicate, raise flag and break loop */
         if ( value == a[ j ] ) {
            duplicate = 1;
            break;
         } /* end if */

      } /* end for */

      /* if number is not a duplicate enter it in array */
      if ( !duplicate ) {
         a[ k++ ] = value;
      } /* end if */

   } /* end for */

   printf( "\nThe nonduplicate values are:\n" );

   /* display array of nonduplicates */
   for ( i = 0; a[ i ] != 0; i++ ) {
      printf( "%d  ", a[ i ] );
   } /* end for */

   printf( "\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter 20 integers between 10 and 100:
10 11 12 13 14 15 16 17 18 19 20 21 10 11 12 13 14 15 16 17

The nonduplicate values are:
10   11   12   13   14   15   16   17   18   19   20   21
```

6.16 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 <= 2; row++ )
   for ( column = 0; column <= 4; column++ )
      sales[ row ][ column ] = 0;
```

ANS:

```
1)  sales[ 0 ][ 0 ]
2)  sales[ 0 ][ 1 ]
3)  sales[ 0 ][ 2 ]
4)  sales[ 0 ][ 3 ]
5)  sales[ 0 ][ 4 ]
6)  sales[ 1 ][ 0 ]
7)  sales[ 1 ][ 1 ]
8)  sales[ 1 ][ 2 ]
9)  sales[ 1 ][ 3 ]
10) sales[ 1 ][ 4 ]
11) sales[ 2 ][ 0 ]
12) sales[ 2 ][ 1 ]
13) sales[ 2 ][ 2 ]
14) sales[ 2 ][ 3 ]
15) sales[ 2 ][ 4 ]
```

6.17 What does the following program do?

```c
/* ex06_17.c */
/* What does this program do? */
#include <stdio.h>
#define SIZE 10

int whatIsThis( const int b[], int p ); /* function prototype */

/* function main begins program execution */
int main()
{
   int x; /* holds return value of function whatIsThis */

   /* initialize array a */
   int a[ SIZE ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

   x = whatIsThis( a, SIZE );

   printf( "Result is %d\n", x );

   return 0; /* indicates successful termination */

} /* end main */

/* what does this function do? */
int whatIsThis( const int b[], int p )
{
   /* base case */
   if ( p == 1 ) {
      return b[ 0 ];
   } /* end if */
   else { /* recursion step */

      return b[ p - 1 ] + whatIsThis( b, p - 1 );
   } /* end else */

} /* end function whatIsThis */
```

ANS: The program recursively sums the elements in a.

```
Result is 55
```

6.18 What does the following program do?

```c
/* ex06_18.c */
/* What does this program do? */
#include <stdio.h>
#define SIZE 10

/* function prototype */
void someFunction( const int b[], int startIndex, int size );

/* function main begins program execution */
int main()
{
   int a[ SIZE ] = { 8, 3, 1, 2, 6, 0, 9, 7, 4, 5 }; /* initialize a */

   printf( "Answer is:\n" );
   someFunction( a, 0, SIZE );
   printf( "\n" );

    return 0; /* indicates successful termination */

} /* end main */

/* What does this function do? */
void someFunction( const int b[], int startIndex, int size )
{
   if ( startIndex < size ) {
      someFunction( b, startIndex + 1, size );
      printf( "%d  ", b[ startIndex ] );
   } /* end if */

} /* end function someFunction */
```

ANS: The program recursively outputs the values of a in reverse order.

```
Answer is:
5  4  7  9  0  6  2  1  3  8
```

6.19 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 6.23 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. 6.23 The 36 possible outcomes of rolling two dice.

ANS:

```c
/* Exercise 6.19 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
   long i;                  /* loop counter */
   int j;                   /* loop counter */
   int x;                   /* first die */
   int y;                   /* second die */
   int sum[ 13 ] = { 0 }; /* count occurrences of each combination */

   /* array expected contains counts for the expected
      number of times each sum occurs in 36 rolls of the dice */
   int expected[ 13 ] = { 0, 0, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1};

   srand( time( NULL ) ); /* seed random number generator */

   /* roll dice 36,000 times */
   for ( i = 1; i <= 36000; i++ ) {
      x = 1 + rand() % 6;
      y = 1 + rand() % 6;
      ++sum[ x + y ];
   } /* end for */

   printf( "%10s%10s%10s%10s\n", "Sum", "Total", "Expected", "Actual" );

   /* display results of rolling dice */
   for ( j = 2; j <= 12; j++ ) {
      printf( "%10d%10d%9.3f%%%9.3f%%\n", j, sum[ j ],
         100.0 * expected[ j ] / 36, 100.0 * sum[ j ] / 36000 );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
     Sum      Total   Expected     Actual
       2       1018     2.778%     2.828%
       3       2008     5.556%     5.578%
       4       3020     8.333%     8.389%
       5       4024    11.111%    11.178%
       6       4891    13.889%    13.586%
       7       6011    16.667%    16.697%
       8       5065    13.889%    14.069%
       9       3984    11.111%    11.067%
      10       2970     8.333%     8.250%
      11       1989     5.556%     5.525%
      12       1020     2.778%     2.833%
```

6.20 Write a program that runs 1000 games of craps (without human intervention) and answers each of the following questions:

a) How many games are won on the first roll, second roll, …, twentieth roll and after the twentieth roll?
b) How many games are lost on the first roll, second roll, …, twentieth roll and after the twentieth 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?

ANS:

```
/* Exercise 6.20 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

enum Outcome { CONTINUE, WIN, LOSE };

int rollDice( void ); /* function prototype */

int main()
{
   enum Outcome gameStatus;  /* game status indicator */
   int sum;                  /* sum of rolled dice */
   int myPoint;              /* current point */
   int i;                    /* game counter */
   int roll;                 /* roll counter */
   int length = 0;           /* average length of game */
   int wins[ 22 ] = { 0 };   /* wins per roll */
   int losses[ 22 ] = { 0 }; /* losses per roll */
   int winSum = 0;           /* total wins */
   int loseSum = 0;          /* total losses */

   srand( time( NULL ) );

   /* play 1000 times */
   for ( i = 1; i <= 1000; i++ ) {
      sum = rollDice();
      roll = 1;

      /* test if game won or lost on first roll */
      switch ( sum ) {

         case 7:
         case 11:
            gameStatus = WIN;
            break; /* exit switch */

         case 2:
         case 3:
         case 12:
            gameStatus = LOSE;
            break; /* exit switch */

         default:
            gameStatus = CONTINUE;
            myPoint = sum;
            break; /* exit switch */
      } /* end switch */

      /* continue while game not won or lost */
      while ( gameStatus == 0 ) {
         sum = rollDice();
         roll++;

         /* win on next roll */
         if ( sum == myPoint ) {
            gameStatus = WIN;
         } /* end if */
```

```
         else { /* lose on next roll */

            if ( sum == 7 ) {
               gameStatus = LOSE;
            } /* end if */

         } /* end else */

      } /* end while */

      /* if more than 21 rolls taken, set number of rolls to 21 */
      if ( roll > 21 ) {
     roll = 21;
      } /* end if */

      /* determine how many rolls were taken and increment
         corresponding counter in wins or losses array */
      if ( gameStatus == WIN ) {
         wins[ roll ]++;
         winSum++;
      } /* end if */
      else {
         losses[ roll ]++;
         loseSum++;
      } /* end else */

   } /* end for */

   printf( "Games won or lost after the 20th roll\n"
           "are displayed as the 21st roll.\n\n" );

   /* display number of games won and lost for each number of rolls */
   for ( i = 1; i <= 21; i++ ) {
      printf( "%3d games won and %3d games lost on roll %d.\n",
         wins[ i ], losses[ i ], i );
   } /* end for */

   /* calculate chances of winning */
   printf( "\nThe chances of winning are %d/%d = %.2f%%\n", winSum,
      winSum + loseSum, 100.0 * winSum / ( winSum + loseSum ) );

   /* calculate average length of game */
   for ( i = 1; i <= 21; i++ ) {
      length += wins[ i ] * i + losses[ i ] * i;
   } /* end for */

   printf( "The average game length is %.2f rolls.\n",
      length / 1000.0 );

   return 0; /* indicate successful termination */

} /* end main */

/* function to simulate dice rolling */
int rollDice( void )
{
   int die1;    /* first die */
   int die2;    /* second die */
   int workSum; /* dice sum */

   die1 = 1 + rand() % 6;
   die2 = 1 + rand() % 6;
   workSum = die1 + die2;

   return workSum; /* return total of two dice */

} /* end function rollDice */
```

```
Games won or lost after the 20th roll
are displayed as the 21st roll.

212 games won and 102 games lost on roll 1.
 63 games won and 109 games lost on roll 2.
 54 games won and  92 games lost on roll 3.
 45 games won and  70 games lost on roll 4.
 40 games won and  54 games lost on roll 5.
 17 games won and  34 games lost on roll 6.
  9 games won and  21 games lost on roll 7.
 10 games won and  11 games lost on roll 8.
  7 games won and   9 games lost on roll 9.
  3 games won and   2 games lost on roll 10.
  6 games won and  12 games lost on roll 11.
  4 games won and   4 games lost on roll 12.
  1 games won and   1 games lost on roll 13.
  1 games won and   0 games lost on roll 14.
  0 games won and   1 games lost on roll 15.
  1 games won and   1 games lost on roll 16.
  0 games won and   0 games lost on roll 17.
  0 games won and   1 games lost on roll 18.
  1 games won and   1 games lost on roll 19.
  0 games won and   0 games lost on roll 20.
  0 games won and   1 games lost on roll 21.

The chances of winning are 474/1000 = 47.40%
The average game length is 3.36 rolls.
```

6.21 (*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. The president has asked you 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"
Please type 2 for "economy"
```

If the person types 1, then your program should assign a seat in the first class section (seats 1-5). If the person types 2, then your program should assign a seat in the economy section (seats 6-10). Your program should then 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 indicate that all seats are empty. As each seat is assigned, set the corresponding elements of the array to 1 to indicate that the seat is 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 economy section (and vice versa). If yes, then make the appropriate seat assignment. If no, then print the message `"Next flight leaves in 3 hours."`

ANS:

```c
/* Exercise 6.21 Solution */
#include <stdio.h>
#include <ctype.h>

int main()
{
   int plane[ 11 ] = { 0 }; /* seats on the plane */
   int i = 0;               /* counter */
   int firstClass= 1;       /* first class seats start at 1 */
   int economy = 6;         /* economy seats start at 6 */
   int choice;              /* user's choice */
   char response[ 2 ];      /* user's response */

   /* loop 10 times */
   while ( i < 10 ) {
      printf( "\n%s\n%s\n? ", "Please type 1 for \"first class\"",
              "Please type 2 for \"economy\"" );
      scanf( "%d", &choice );

      /* if user selects first class */
      if ( choice == 1 ) {

         /* if seat are available in first class */
         if ( !plane[ firstClass ] && firstClass <= 5 ) {
            printf( "Your seat assignment is %d\n", firstClass );
            plane[ firstClass++ ] = 1;
            i++;
         } /* end if */
         /* if no first class seats, but economy seats available */
         else if ( firstClass > 5 && economy <= 10 ) {

            /* ask if passenger would like to sit in economy */
            printf( "The first class section is full.\n" );
            printf( "Would you like to sit in the economy" );
            printf( " section ( Y or N )? " );
            scanf( "%s", response );

            /* if response is yes, then assign seat */
            if ( toupper( response[ 0 ] ) == 'Y' ) {
               printf( "Your seat assignment is %d\n", economy );
               plane[ economy++ ] = 1;
               i++;
            } /* end if */
            else { /* print next departure */
               printf( "Next flight leaves in 3 hours.\n" );
            } /* end else */
```

```c

         } /* end else if */
         else { /* print next departure */
            printf( "Next flight leaves in 3 hours.\n" );
         } /* end else */

      } /* end if */
      else { /* if user selects economy */

         /* if seats available, assign seat */
         if ( !plane[ economy ] && economy <= 10 ) {
            printf( "Your seat assignment is %d\n", economy );
            plane[ economy++ ] = 1;
            i++;
         } /* end if */
         /* if only first class seats are available */
         else if ( economy > 10 && firstClass <= 5 ) {

            /* ask if first class is suitable */
            printf( "The economy section is full.\n" );
            printf( "Would you like to sit in first class" );
            printf( " section ( Y or N )? " );
            scanf( "%s", response );

            /* if response is yes, assign seat */
            if ( toupper( response[ 0 ] ) == 'Y' ) {
               printf( "Your seat assignment is %d\n", firstClass );
               plane[ firstClass++ ] = 1;
               i++;
            } /* end if */
            else { /* print next departure */
               printf( "Next flight leaves in 3 hours.\n" );
            } /* end else */

         } /* end else if */
         else { /* print next departure */
            printf( "Next flight leaves in 3 hours.\n" );
         } /* end else */

      } /* end else */

   } /* end while */

   printf( "\nAll seats for this flight are sold.\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Please type 1 for "first class"
Please type 2 for "economy"
? 2
Your seat assignment is 6

Please type 1 for "first class"
Please type 2 for "economy"
? 1
Your seat assignment is 1

Please type 1 for "first class"
Please type 2 for "economy"
? 2
Your seat assignment is 7
.
.
.
Please type 1 for "first class"
Please type 2 for "economy"
? 1
The first class section is full.
Would you like to sit in the economy section ( Y or N )? n
Next flight leaves in 3 hours.

Please type 1 for "first class"
Please type 2 for "economy"
? 1
The first class section is full.
Would you like to sit in the economy section ( Y or N )? y
Your seat assignment is 9

Please type 1 for "first class"
Please type 2 for "economy"
? 2
Your seat assignment is 10

All seats for this flight are sold.
```

6.22 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:

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.

ANS:

```c
/* Exercise 6.22 Solution */
#include <stdio.h>

int main()
{

   /* total sales for each salesperson and each product */
   double sales[ 4 ][ 5 ] = { 0.0 };
   double productSales[ 5 ] = { 0.0 }; /* total product sales */
   double value;      /* current sales */
   double totalSales; /* total overall sales */
   int salesPerson;   /* current salesperson */
   int product;       /* current product */
   int i;             /* loop counter */
   int j;             /* loop counter */

   printf( "Enter the salesperson, product, and total sales.\n" );
   printf( "Enter -1 for the salesperon to end input.\n" );
   scanf( "%d", &salesPerson );

   /* continue receiving input for each salesperson
      while -1 is not entered */
   while ( salesPerson != -1 ) {
      scanf( "%d%lf", &product, &value );
      sales[ salesPerson ][ product ] = value;
      scanf( "%d", &salesPerson );
   } /* end while */

   /* display table */
   printf( "\n%s\n%s\n%s\n%s\n%s\n", "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.\n" );
   printf( " %8d%8d%8d%8d%8d\n", 1, 2, 3, 4, 5 );

   /* display salespeople and sales */
   for ( i = 0; i <= 3; i++ ) {
      totalSales = 0.0;
      printf( "%d", i);

      /* add total sales and display individual sales */
      for ( j = 0; j <= 4; j++ ) {
         totalSales += sales[ i][ j ];
         printf( "%8.2f", sales[ i ][ j ] );
         productSales[ j ] += sales[ i ][ j ];
      } /* end for */

      printf( "%8.2f\n", totalSales );
   } /* end for */

   printf( " " );

```

```c
   /* display total product sales */
   for ( j = 0; j <= 4; j++ ) {
      printf( "%8.2f", productSales[ j ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the salesperson, product, and total sales.
Enter -1 for the salesperon to end input.
0 0 1.00
0 1 2.00
0 2 3.00
0 3 4.00
0 4 5.00
1 0 1.00
1 1 2.00
1 2 3.00
1 3 4.00
1 4 5.00
2 0 1.00
2 1 2.00
2 2 3.00
2 3 4.00
2 4 5.00
3 0 1.00
3 1 2.00
3 2 3.00
3 3 4.00
3 4 5.00
-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
0    1.00    2.00    3.00    4.00    5.00   15.00
1    1.00    2.00    3.00    4.00    5.00   15.00
2    1.00    2.00    3.00    4.00    5.00   15.00
3    1.00    2.00    3.00    4.00    5.00   15.00
     4.00    8.00   12.00   16.00   20.00
```

6.23 (*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 pen is up, the turtle moves about freely without writing anything. In this problem you will simulate the operation of the turtle and create a computerized sketchpad as well.

Use a 50-by-50 array `floor` which 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 position 0,0 of the floor with its pen up. The set of turtle commands your program must process are shown in Fig. 6.24.

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:

```
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.

ANS:

```c
/* Exercise 6.23 Solution */
#include <stdio.h>

#define TRUE 1
#define FALSE 0
#define MAX 100 /* the maximum number of commands */

/* function prototypes */
void getCommands( int commands[][ 2 ] );
int turnRight( int d );
int turnLeft( int d );
void movePen( int down, int a[][ 50 ], int dir, int dist );
void printArray( int a[][ 50 ] );

int main()
{
   int floor[ 50 ][ 50 ] = { 0 };        /* floor grid */
   int penDown = FALSE;                  /* pen down flag */
   int command;                          /* current command */
```

```c
   int direction = 0;                       /* direction indicator */
   int commandArray[ MAX ][ 2 ] = { 0 }; /* array of commands */
   int distance;                          /* distance to move */
   int count = 0;                         /* command counter */

   getCommands( commandArray );
   command = commandArray[ count ][ 0 ];

   /* continue receiving input while -9 is not entered */
   while ( command != 9 ) {

      /* determine what command was entered and perform action */
      switch ( command ) {

         case 1:
            penDown = FALSE;
            break; /* exit switch */

         case 2:
            penDown = TRUE;
            break; /* exit switch */

         case 3:
            direction = turnRight( direction );
            break; /* exit switch */

         case 4:
            direction = turnLeft( direction );
            break; /* exit switch */

         case 5:
            distance = commandArray[ count ][ 1 ];
            movePen( penDown, floor, direction, distance );
            break; /* exit switch */

         case 6:
            printf( "\nThe drawing is:\n\n" );
            printArray( floor );
            break; /* exit switch */
      } /* end switch */

      command = commandArray[ ++count ][ 0 ];
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */

/* getCommands prompts user for commands */
void getCommands( int commands[][ 2 ] )
{
   int i;           /* counter */
   int tempCommand; /* temporary command holder */

   printf( "Enter command ( 9 to end input ): " );
   scanf( "%d", &tempCommand );

   /* recieve commands until -9 or 100 commands are entered */
   for ( i = 0; tempCommand != 9 && i < MAX; i++ ) {
      commands[ i ][ 0 ] = tempCommand;

      /* ignore comma after 5 is entered */
      if ( tempCommand == 5 ) {
         scanf( ",%d", &commands[ i ][ 1 ] );
      } /* end if */

      printf( "Enter command ( 9 to end input ): " );
      scanf( "%d", &tempCommand );
   } /* end for */
```

```

   commands[ i ][ 0 ] = 9; /* last command */
} /* end function getCommands */

/* turnRight turns turtle to the right */
int turnRight( int d )
{
   return ++d > 3 ? 0 : d;

} /* end function turnRight */

/* turnLeft turns turtle to the left */
int turnLeft( int d )
{
   return --d < 0 ? 3 : d;

} /* end function turnLeft */

/* movePen moves the pen */
void movePen( int down, int a[][ 50 ], int dir, int dist )
{
   int i;                 /* loop counter */
   int j;                 /* loop counter */
   static int xPos = 0; /* x coordinate */
   static int yPos = 0; /* y coordinate */

   /* determine which way to move pen */
   switch ( dir ) {

      case 0: /* move to the right */

         /* move dist spaces or until edge of floor */
         for ( j = 1; j <= dist && yPos + j < 50; j++ ) {

            /* draw 1 if pen is down */
            if ( down ) {
               a[ xPos ][ yPos + j ] = 1;
            } /* end if */

         } /* end for */

         yPos += j - 1;
         break; /* exit switch */

      case 1: /* move down */

         /* move dist spaces or until edge of floor */
         for ( i = 1; i <= dist && xPos + i < 50; i++ ) {

            /* draw 1 if pen is down */
            if ( down ) {
               a[ xPos + i ][ yPos ] = 1;
            } /* end if */

         } /* end for */

         xPos += i - 1;
         break; /* exit switch */

      case 2: /* move to the left */

         /* move dist spaces or until edge of floor */
         for ( j = 1; j <= dist && yPos - j >= 0; j++ ) {

            /* draw 1 if pen is down */
            if ( down ) {
               a[ xPos ][ yPos - j ] = 1;
            } /* end if */
```

```c

         } /* end for */

         yPos -= j - 1;
         break; /* exit switch */

      case 3: /* move up */

         /* move dist spaces or until edge of floor */
         for ( i = 1; i <= dist && xPos - i >= 0; i++ ) {

            /* draw 1 if pen is down */
            if ( down ) {
               a[ xPos - i ][ yPos ] = 1;
            } /* end if */

         } /* end for */

         xPos -= i - 1;
         break; /* exit switch */
   } /* end switch */

} /* end function movePen */

/* printArray prints array drawing */
void printArray( int a[][ 50 ] )
{
   int i; /* counter */
   int j; /* counter */

   /* loop through array */
   for ( i = 0; i < 50; i++ ) {

      /* loop through array */
      for ( j = 0; j < 50; j++ ) {
         putchar( a[ i ][ j ] ? '*' : ' ' );
      } /* end for */

      putchar( '\n' );
   } /* end for */

} /* end function printArray */
```

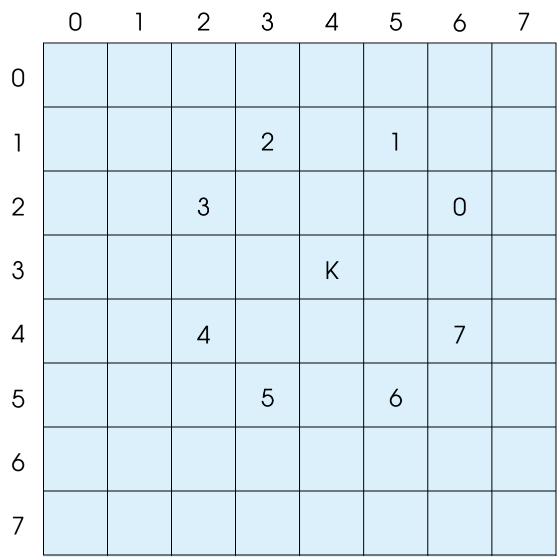
```
Enter command ( 9 to end input ): 2
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 3
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 3
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 3
Enter command ( 9 to end input ): 5,12
Enter command ( 9 to end input ): 1
Enter command ( 9 to end input ): 6
Enter command ( 9 to end input ): 9

The drawing is:

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

6.24 (*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. 6.25.

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? Were you close to the estimate?

b) Now let us develop a program that will move the knight around a chessboard. The board itself 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. 6.25 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 on the board. 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 in some sense 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 that 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 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, the center squares are therefore rated as `8`s, the corner squares are rated as `2`s, 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 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.

ANS:

```
/* Exercise 6.24 Part C Solution */
/* Knight's Tour - access version */
/* runs one tour */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TRUE  1
#define FALSE 0

/* function prototypes */
void clearBoard( int workBoard[][ 8 ] );
void printBoard( int workBoard[][ 8 ] );
int validMove( int row, int column, int workBoard[][ 8 ] );

int main()
{
   int board[ 8 ][ 8 ]; /* chess board */
```

```

   /* array of accesibility */
   int access[ 8 ][ 8 ] = { 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 };

   /* eight horizontal and vertical moves for the knight */
   int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
   int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
   int currentRow;      /* current row */
   int currentColumn;   /* current column */
   int moveNumber = 0;  /* move counter */
   int testRow;         /* possible next row */
   int testColumn;      /* possible next column */
   int minRow;          /* row with minimum access number */
   int minColumn;       /* column with minimum access number */
   int minAccess = 9;   /* impossible access number  */
   int accessNumber;    /* current access number */
   int moveType;        /* current move type */
   int done;            /* flag to indicate end */

   srand( time( NULL ) );

   clearBoard( board ); /* initialize array board */
   currentRow = rand() % 8;
   currentColumn = rand() % 8;
   board[ currentRow ][ currentColumn ] = ++moveNumber;
   done = FALSE;

   /* continue while knight still has valid moves */
   while ( !done ) {
      accessNumber = minAccess;

      /* loop through all move types */
      for ( moveType = 0; moveType < 8; moveType++ ) {
         testRow = currentRow + vertical[ moveType ];
         testColumn = currentColumn + horizontal[ moveType ];

         /* make sure move is valid */
         if ( validMove( testRow, testColumn, board ) ) {

            /* if move is valid and has lowest accessNumber,
               set square to accessNumber */
            if ( access[ testRow ][ testColumn ] < accessNumber ) {
               accessNumber = access[ testRow ][ testColumn ];
               minRow = testRow;
               minColumn = testColumn;
            } /* end if */

            --access[ testRow ][ testColumn ];
         } /* end if */

      } /* end for */

      /* end if knight has no moves */
      if ( accessNumber == minAccess ) {
         done = TRUE;
      } /* end if */
      else {
         currentRow = minRow;
         currentColumn = minColumn;
         board[ currentRow ][ currentColumn ] = ++moveNumber;
      } /* end else */
```

```
   } /* end while */

   printf( "The tour ended with %d moves.\n", moveNumber );

   /* determine and print if a full tour was made */
   if ( moveNumber == 64 ) {
      printf( "This was a full tour!\n\n" );
   } /* end if */
   else {
      printf( "This was not a full tour.\n\n" );
   } /* end else */

   printf( "The board for this test is:\n\n" );
   printBoard( board );

   return 0; /* indicate successful termination */

} /* end main */

/* function to clear chess board */
void clearBoard( int workBoard[][ 8 ] )
{
   int row; /* row counter */
   int col; /* column counter */

   /* set all squares to zero */
   for ( row = 0; row < 8; row++ ) {

      for ( col = 0; col < 8; col++ ) {
         workBoard[ row ][ col ] = 0;
      } /* end for */

   } /* end for */

} /* end function clearBoard */

/* function to print chess board */
void printBoard( int workBoard[][ 8 ] )
{
   int row; /* row counter */
   int col; /* column counter */

   printf( "   0  1  2  3  4  5  6  7\n" );

   /* print squares */
   for ( row = 0; row < 8; row++ ) {
      printf( "%d", row );

      for ( col = 0; col < 8; col++ ) {
         printf( "%3d", workBoard[ row ][ col ] );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );
} /* end function printBoard */

/* function to determine if move is legal */
int validMove( int row, int column, int workBoard[][ 8 ] )
{

   /* NOTE: This test stops as soon as it becomes false */
   return ( row >= 0 && row <= 7 && column >= 0 &&
      column <= 7 && workBoard[ row ][ column ] == 0 );

} /* end function validMove */
```

```
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 33 30 19  4 23 28 17  2
1 20  5 32 29 18  3 24 27
2 31 34 49 22 37 26  1 16
3  6 21 36 59 50 47 38 25
4 35 60 51 48 39 58 15 46
5 10  7 62 57 54 45 40 43
6 61 52  9 12 63 42 55 14
7  8 11 64 53 56 13 44 41
```

6.25 (*Knight's Tour: Brute Force Approaches*) In Exercise 6.24 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 many 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 chess board (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?

ANS:

```c
/* Exercise 6.25 Part A Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NO 0
#define YES 1

/* function prototypes */
int validMove( int row, int column, int workBoard[][ 8 ] );
void printBoard( int board[][ 8 ] );

int main()
{
   int currentRow;     /* current row */
   int currentColumn;  /* current column */
   int moveType;       /* current move type */
   int moveNumber = 0; /* move counter */
   int testRow;        /* possible next row */
   int testColumn;     /* possible next column */
   int count;          /* counter */
   int done;           /* flag to indicate end */
   int goodMove;       /* result of call to validMove */

   /* horizontal and vertical moves for the knight, and board */
   int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
   int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };
   int board[ 8 ][ 8 ] = { 0 };

   srand( time( NULL ) );

   currentRow = rand() % 8;
   currentColumn = rand() % 8;
   board[ currentRow ][ currentColumn ] = ++moveNumber;
   done = NO;

   /* continue while knight can still move */
   while ( !done ) {
      moveType = rand() % 8;
      testRow = currentRow + vertical[ moveType ];
      testColumn = currentColumn + horizontal[ moveType ];
```

```c
        goodMove = validMove( testRow, testColumn, board );

        /* test if desired move is valid */
        if ( goodMove ) {
           currentRow = testRow;
           currentColumn = testColumn;
           board[ currentRow ][ currentColumn ] = ++moveNumber;
        } /* end if */
        else {

           /* if move is not legal try another random move */
           for ( count = 0; count < 7 && !goodMove; count++ ) {
              moveType = ++moveType % 8;
              testRow = currentRow + vertical[ moveType ];
              testColumn = currentColumn + horizontal[ moveType ];
              goodMove = validMove( testRow, testColumn, board );

              /* test if new move is good */
              if ( goodMove ) {
                 currentRow = testRow;
                 currentColumn = testColumn;
                 board[ currentRow ][ currentColumn ] = ++moveNumber;
              } /* end if */

           } /* end for */

           /* if no valid moves, knight can no longer move */
           if ( !goodMove ) {
              done = YES;
           } /* end if */

        } /* end else */

        /* if 64 moves have been made, a full tour is complete */
        if ( moveNumber == 64 ) {
           done = YES;
        } /* end if */

     } /* end while */

     printf( "The tour has ended with %d moves.\n", moveNumber );

     /* test if full tour was made */
     if ( moveNumber == 64 ) {
        printf( "This was a full tour!\n" );
     } /* end if */
     else {
        printf( "This was not a full tour.\n" );
     } /* end else */

     printf( "The board for this random test was:\n\n" );
     printBoard( board ); /* print the board */

     return 0; /* indicate successful termination */

  } /* end main */

  /* function to test whether a square is on the board
     and has not been visited yet */
  int validMove( int row, int column, int workBoard[][ 8 ] )
  {
     /* NOTE: This test stops as soon as it becomes false */
     return ( row >= 0 && row < 8 && column >= 0 &&
        column < 8 && workBoard[ row ][ column ] == 0 );

  } /* end function validMove */

```

```
/* function to print the chess board */
void printBoard( int board[][ 8 ] )
{
   int row; /* row counter */
   int col; /* column counter */

   printf( "   0  1  2  3  4  5  6  7\n" );


   /* print the rows and columns of the chess board */
   for ( row = 0; row < 8; row++ ) {
      printf( "%d", row );

      for ( col = 0; col < 8; col++ ) {
         printf( "%3d", board[ row ][ col ] );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );
} /* end function printBoard */
```

```
The tour has ended with 32 moves.
This was not a full tour.
The board for this random test was:

   0  1  2  3  4  5  6  7
0 13  0  0  0 11 28  0 32
1  0  0 12 29 24 31  8  0
2  0 14  0  0 27 10 25  6
3 18  0 16 23 30  7  0  9
4 15  0 19  0  0 26  5  0
5 20 17  0  0 22  0  2  0
6  0  0 21  0  0  0  0  4
7  0  0  0  0  0  3  0  1
```

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?

ANS:

```
/* Exercise 6.25 Part B Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NO 0
#define YES 1

int validMove( int, int, int [][ 8 ] );

int main()
{
   int currentRow;    /* current row */
   int currentColumn; /* current column */
   int moveType;      /* current move type */
   int moveNumber;    /* move counter */
   int testRow;       /* possible next row */
   int testColumn;    /* possible next column */
   int count;         /* counter */
   int i;             /* counter */
   int row;           /* row */
   int col;           /* column */
```

```
   int done;          /* flag to indicate end */
   int goodMove;      /* result of call to validMove */
   int board[ 8 ][ 8 ]; /* chess board */
   int moveTotal[ 65 ] = { 0 }; /* array of tour totals */

   /* horizontal and vertical moves for the knight */
   int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
   int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };

   srand( time( NULL ) );

   /* attempt 1000 tours */
   for ( i = 0; i < 1000; i++ ) {

      /* set all squares equal to 0 */
      for ( row = 0; row < 8; row++ ) {

         for ( col = 0; col < 8; col++ ) {
            board[ row ][ col ] = 0;
         } /* end for */

      } /* end for */

      moveNumber = 0;

      currentRow = rand() % 8;
      currentColumn = rand() % 8;
      board[ currentRow ][ currentColumn ] = ++moveNumber;
      done = NO;

      /* continue while knight still has valid moves */
      while ( !done ) {
         moveType = rand() % 8;
         testRow = currentRow + vertical[ moveType ];
         testColumn = currentColumn + horizontal[ moveType ];
         goodMove = validMove( testRow, testColumn, board );

         /* if desired move is valid, move knight to square */
         if ( goodMove ) {
            currentRow = testRow;
            currentColumn = testColumn;
            board[ currentRow ][ currentColumn ] = ++moveNumber;
         } /* end if */
         else {

            /* if move is invalid, test other possible moves */
            for ( count = 0; count < 7 && !goodMove; count++ ) {
               moveType = ++moveType % 8;
               testRow = currentRow + vertical[ moveType ];
               testColumn = currentColumn + horizontal[ moveType ];
               goodMove = validMove( testRow, testColumn, board );

               /* if move is valid, move knight to square */
               if ( goodMove ) {
                  currentRow = testRow;
                  currentColumn = testColumn;
                  board[ currentRow ][ currentColumn ] = ++moveNumber;
               } /* end if */

            } /* end for */

            /* if no valid moves, while loop exits */
            if ( !goodMove ) {
               done = YES;
            } /* end if */

         } /* end else */

```

```
            /* if full tour is made, while loop exits */
            if ( moveNumber == 64 ) {
               done = YES;
            } /* end if */

         } /* end while */

         ++moveTotal[ moveNumber ];
      } /* end for */

      /* dislay how many tours of each move number were made */
      for ( i = 1; i < 65; i++ ) {

         if ( moveTotal[ i ] ) {
            printf( "There were %d tours of %d moves.\n",
               moveTotal[ i ], i );
         } /* end if */

      } /* end for */

      return 0; /* indicate successful termination */

   } /* end main */

   /* function to determine if a move is legal */
   int validMove( int testRow, int testColumn, int board[][ 8 ] )
   {

      /* test if square is on board and if knight has previously
         visited it */
      if ( testRow >= 0 && testRow < 8 && testColumn >= 0 &&
         testColumn < 8 ) {
         return board[ testRow ][ testColumn ] != 0 ? NO : YES;

      } /* end if */
      else {
         return NO;
      } /* end else */

   } /* end function validMove */
```

```
There were 1 tours of 4 moves.
There were 3 tours of 5 moves.
There were 2 tours of 6 moves.
There were 3 tours of 7 moves.
There were 2 tours of 8 moves.
There were 4 tours of 10 moves.
There were 5 tours of 11 moves.
There were 4 tours of 12 moves.
There were 5 tours of 13 moves.
There were 7 tours of 14 moves.
There were 7 tours of 15 moves.
There were 9 tours of 16 moves.
There were 8 tours of 17 moves.
There were 10 tours of 18 moves.
There were 9 tours of 19 moves.
There were 9 tours of 20 moves.
There were 11 tours of 21 moves.
There were 19 tours of 22 moves.
There were 17 tours of 23 moves.
There were 18 tours of 24 moves.
There were 12 tours of 25 moves.
There were 20 tours of 26 moves.
There were 14 tours of 27 moves.
There were 18 tours of 28 moves.
There were 22 tours of 29 moves.
There were 21 tours of 30 moves.
There were 31 tours of 31 moves.
There were 28 tours of 32 moves.
There were 25 tours of 33 moves.
There were 32 tours of 34 moves.
There were 26 tours of 35 moves.
There were 40 tours of 36 moves.
There were 38 tours of 37 moves.
There were 38 tours of 38 moves.
There were 37 tours of 39 moves.
There were 33 tours of 40 moves.
There were 35 tours of 41 moves.
There were 34 tours of 42 moves.
There were 33 tours of 43 moves.
There were 36 tours of 44 moves.
There were 30 tours of 45 moves.
There were 35 tours of 46 moves.
There were 26 tours of 47 moves.
There were 37 tours of 48 moves.
There were 22 tours of 49 moves.
There were 17 tours of 50 moves.
There were 20 tours of 51 moves.
There were 21 tours of 52 moves.
There were 17 tours of 53 moves.
There were 19 tours of 54 moves.
There were 14 tours of 55 moves.
There were 3 tours of 56 moves.
There were 7 tours of 57 moves.
There were 3 tours of 59 moves.
There were 3 tours of 60 moves.
```

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.

6.26 (*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—that is, so that no two queens are in the same row, the same column, or along the same diagonal? Use the kind of thinking developed in Exercise 6.24 to formulate a heuristic for solving the Eight Queens problem. Run your program. [*Hint:* It is possible to assign a numeric value to each square of the chessboard indicating how many squares of an empty chessboard are "eliminated" once a queen is placed in that square. For example, each of the four corners would be assigned the value 22, as in Fig. Fig. 6.26.]

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

Fig. 6.26 The 22 squares eliminated by placing a queen in the upper-left corner.

Once these "elimination numbers" are placed in all 64 squares, an appropriate heuristic might be: Place the next queen in the square with the smallest elimination number. Why is this strategy intuitively appealing?

6.27 (*Eight Queens: Brute Force Approaches*) In this problem you will develop several brute force approaches to solving the Eight Queens problem introduced in Exercise 6.26.

a) Solve the Eight Queens problem, using the random brute force technique developed in Exercise 6.25.
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.

6.28 (*Duplicate elimination*) In Chapter 12, we explore the high-speed binary search tree data structure. One feature of a binary search tree is that duplicate values are discarded when insertions are made into the tree. This is referred to as duplicate elimination. Write a program that produces 20 random numbers between 1 and 20. The program should store all nonduplicate values in an array. Use the smallest possible array to accomplish this task.

```
/* Exercise 6.28 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SIZE 20

int main()
{
   int loop;                   /* loop counter */
   int randNumber;             /* current random number */
   int loop2;                  /* loop counter */
   int subscript = 0;          /* array subscript counter */
   int duplicate;              /* duplicate flag */
   int array[ SIZE ] = { 0 }; /* array of random numbers */

   srand( time( NULL ) );

   /* loop 20 times */
   for ( loop = 0; loop <= SIZE - 1; loop++ ) {
      duplicate = 0;
      randNumber = 1 + rand() % 20; /* generate random number */

      /* loop through current numbers in array */
      for ( loop2 = 0; loop2 <= subscript; loop2++ ) {

         /* compare randNumber with previous numbers */
         if ( randNumber == array[ loop2 ] ) {
            duplicate = 1;
            break;
         } /* end if */

      } /* end for */

      /* if not a duplicate */
      if ( !duplicate ) {
         array[ subscript++ ] = randNumber;
      } /* end if */

   } /* end while */

   printf( "Non-repetitive array values are:\n" );

   /* display array */
   for ( loop = 0; array[ loop ] != 0; loop++ ) {
      printf( "\t\t\t\tArray[ %d ] = %d\n", loop, array[ loop ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Non-repetitive array values are:
                                Array[ 0 ] = 11
                                Array[ 1 ] = 17
                                Array[ 2 ] = 3
                                Array[ 3 ] = 18
                                Array[ 4 ] = 9
                                Array[ 5 ] = 2
                                Array[ 6 ] = 20
                                Array[ 7 ] = 4
                                Array[ 8 ] = 1
                                Array[ 9 ] = 10
                                Array[ 10 ] = 7
                                Array[ 11 ] = 13
                                Array[ 12 ] = 19
                                Array[ 13 ] = 6
                                Array[ 14 ] = 8
                                Array[ 15 ] = 16
```

6.29 (*Knight's Tour: Closed Tour Test*) In the Knight's Tour, a full tour is 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 6.24 to test for a closed tour if a full tour has occurred.

ANS:

```
/* Exercise 6.29 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TRUE 1
#define FALSE 0

/* function prototypes */
void clearBoard( int workBoard[][ 8 ] );
void printBoard( int workBoard[][ 8 ] );
int validMove( int row, int column, int workBoard[][ 8 ] );

int main( void )
{

   int firstMoveRow;    /* starting row */
   int firstMoveCol;    /* starting column */
   int closedTour = 0;  /* closed tour flag */
   int currentRow;      /* current row */
   int currentColumn;   /* current column */
   int moveNumber = 0;  /* move counter */
   int testRow;         /* possible next row */
   int testColumn;      /* possible next column */
   int minRow;          /* minimum row access number */
   int minColumn;       /* minimum column access number */
   int minAccess = 9;   /* access number reset */
   int accessNumber;    /* current access number */
   int moveType;        /* current move type */
   int done;            /* flag to indicate end */
   int board[ 8 ][ 8 ]; /* chess board */

   /* horizontal and vertical moves for the knight */
   int horizontal[ 8 ] = { 2, 1, -1, -2, -2, -1, 1, 2 };
   int vertical[ 8 ] = { -1, -2, -2, -1, 1, 2, 2, 1 };

   /* access grid */
   int access[ 8 ][ 8 ] = { 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 };


   srand( time( NULL ) );

   clearBoard( board ); /* initialize array board */
   currentRow = rand() % 8;
   currentColumn = rand() % 8;
   firstMoveRow = currentRow; /* store first moves row */
   firstMoveCol = currentColumn; /* store first moves col */

   board[ currentRow ][ currentColumn ] = ++moveNumber;
   done = FALSE;

   /* loop while knight can still move */
   while ( !done ) {
      accessNumber = minAccess;
```

```
         /* test what moves knight can make */
         for ( moveType = 0; moveType < 8; moveType++ ) {
            testRow = currentRow + vertical[ moveType ];
            testColumn = currentColumn + horizontal[ moveType ];

            /* if the knight can make a valid move */
            if ( validMove( testRow, testColumn, board ) ) {

               /* if move has lowest accessNumber, move to that space */
               if ( access[ testRow ][ testColumn ] < accessNumber ) {
                  accessNumber = access[ testRow ][ testColumn ];
                  minRow = testRow;
                  minColumn = testColumn;
               } /* end if */

               --access[ testRow ][ testColumn ];
            } /* end if */

         } /* end for */

         /* if knight cannot access any more squares, loop terminates */
         if ( accessNumber == minAccess ) {
            done = TRUE;
         } /* end if */
         else {
            currentRow = minRow;
            currentColumn = minColumn;
            board[ currentRow ][ currentColumn ] = ++moveNumber;

            /* check for closed tour */
            if ( moveNumber == 64 ) {

               /* loop through possible next moves */
               for ( moveType = 0; moveType < 8; moveType++ ) {
                  testRow = currentRow + vertical[ moveType ];
                  testColumn = currentColumn + horizontal[ moveType ];

                  /* test if knight is one move away from start */
                  if ( testRow == firstMoveRow && testColumn ==
                     firstMoveCol ) {
                     closedTour = 1;
                  } /* end if */

               } /* end for */

            } /* end if */

         } /* end else */

      } /* end while */

      printf( "The tour ended with %d moves.\n", moveNumber );

      /* display results of tour */
      if ( moveNumber == 64 && closedTour == 1 ) {
         printf( "This was a closed tour!\n\n" );
      } /* end if */
      else if ( moveNumber == 64 ) {
         printf( "This was a full tour!\n\n" );
      } /* end else if */
      else {
         printf( "This was not a full tour.\n\n" );
      } /* end else */

      printf( "The board for this test is:\n\n" );
      printBoard( board );

```

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

} /* end main */

/* function to clear the chess board */
void clearBoard( int workBoard[][ 8 ] )
{
   int row; /* row counter */
   int col; /* col counter */

   /* set all values on board to 0 */
   for ( row = 0; row < 8; row++ ) {

      for ( col = 0; col < 8; col++ ) {
         workBoard[ row ][ col ] = 0;
      } /* end for */

   } /* end for */

} /* end function clearBoard */

/* function to print the chesboard */
void printBoard( int workBoard[][ 8 ] )
{
   int row; /* row counter */
   int col; /* column counter */

   printf( "   0  1  2  3  4  5  6  7\n" );

   /* print rows of chessboard */
   for ( row = 0; row < 8; row++ ) {
      printf( "%d", row );

      /* print columns of chess board */
      for ( col = 0; col < 8; col++ ) {
         printf( "%3d", workBoard[ row ][ col ] );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\n" );
} /* end function printBoard */

/* function to determine if a move is valid */
int validMove( int row, int column, int workBoard[][ 8 ] )
{

   /* NOTE: This test stops as soon as it becomes false */
   return ( row >= 0 && row < 8 && column >= 0 &&
      column < 8 && workBoard[ row ][ column ] == 0 );

} /* end function validMove */
```

```
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 32 13 34 57 30 15 42 19
1 35 58 31 14 47 18 29 16
2 12 33 60 49 56 41 20 43
3 59 36 55 46 25 48 17 28
4 54 11 50 61 40 27 44 21
5 37 62 53 26 45 24  3  6
6 10 51 64 39  8  5 22  1
7 63 38  9 52 23  2  7  4
```

6.30 (*The Sieve of Eratosthenes*) A prime integer is any integer that can be divided evenly only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It works as follows:

1) 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.
2) 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.).

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.

ANS:

```c
/* Exercise 6.30 Solution */
#include <stdio.h>
#define SIZE 1000

int main()
{
   int array[ SIZE ]; /* array to indicate prime numbers */
   int loop;          /* loop counter */
   int loop2;         /* loop counter */
   int count = 0;     /* total prime numbers */

   /* set all array elements to 1 */
   for ( loop = 0; loop < SIZE; loop++ ) {
      array[ loop ] = 1;
   } /* end for */

   /* test for multiples of current subscript */
   for ( loop = 1; loop < SIZE; loop++ ) {

      /* start with array subscript two */
      if ( array[ loop ] == 1 && loop != 1 ) {

         /* loop through remainder of array */
         for ( loop2 = loop; loop2 <= SIZE; loop2++ ) {

            /* set to zero all multiples of loop */
            if ( loop2 % loop == 0 && loop2 != loop ) {
               array[ loop2 ] = 0;
            } /* end if */

         } /* end for */

      } /* end if */

   } /* end for */

   /* display prime numbers in the range 2 - 197 */
   for ( loop = 2; loop < SIZE; loop++ ) {

      if ( array[ loop ] == 1 ) {
         printf( "%3d is a prime number.\n", loop );
         ++count;
      } /* end if */

   } /* end for */

   printf( "A total of %d prime numbers were found.\n", count );

   return 0; /* indicate successful termination */

} /* end main */
```

```
  2 is a prime number.
  3 is a prime number.
  5 is a prime number.
  7 is a prime number.
 11 is a prime number.
 13 is a prime number.
 17 is a prime number.
 19 is a prime number.
.
.
.
971 is a prime number.
977 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.
```

6.31 (*Bucket Sort*) A bucket sort begins with an 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.

The algorithm is as follows:

1) Loop through the single-subscripted array and place each of its values in a row of the bucket array based on its ones digit. For example, 97 is placed in row 7, 3 is placed in row 3 and 100 is placed in row 0.
2) Loop through the bucket array and copy the values back to the original array. The new order of the above values in the single-subscripted array is 100, 3 and 97.
3) Repeat this process for each subsequent digit position (tens, hundreds, thousands, etc.) and stop when the leftmost digit of the largest number has be processed.

On the second pass of the array, 100 is placed in row 0, 3 is placed in row 0 (it had only one digit) and 97 is placed in row 9. 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 3). The bucket sort is guaranteed to have all the values properly sorted after processing the leftmost digit of the largest number. The bucket sort knows it is done when all the values are copied into row zero of the double-subscripted array.

Note that the double-subscripted array of buckets is ten times the size of the integer array being sorted. This sorting technique provides better performance than a bubble sort, but requires much larger storage capacity. Bubble sort requires only one additional memory location for the type of data being sorted. Bucket sort is an example of a space-time trade-off. It uses more memory, 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 move the data between the two bucket arrays until all the data is copied into row zero of one of the arrays. Row zero then contains the sorted array.

ANS:

```c
/* Exercise 6.31 Solution */
#include <stdio.h>

/* symbolic constant SIZE must be defined as the array size
   for bucketSort to work */
#define SIZE 12

/* function prototypes */
void bucketSort( int a[] );
void distributeElements( int a[], int buckets[][ SIZE ], int digit );
void collectElements( int a[], int buckets[][ SIZE ] );
int numberOfDigits( int b[], int arraySize );
void zeroBucket( int buckets[][ SIZE ] );

int main()
{

   /* array to be sorted */
   int array[ SIZE ] = { 19, 13, 5, 27, 1, 26, 31, 16, 2, 9, 11, 21 };
```

```
   int i; /* loop counter */

   printf( "Array elements in original order:\n" );

   /* display the unsorted array */
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%3d", array[ i ] );
   } /* end for */

   putchar( '\n' );
   bucketSort( array ); /* sort the array */

   printf( "\nArray elements in sorted order:\n" );

   /* display sorted array */
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%3d", array[ i ] );
   } /* end for */

   putchar( '\n' );

   return 0; /* indicate successful termination */

} /* end main */

/* Perform the bucket sort algorithm */
void bucketSort( int a[] )
{
   int totalDigits;                  /* largest # of digits in array */
   int i;                            /* loop counter */
   int bucket[ 10 ][ SIZE ] = { 0 }; /* initialize bucket array */

   totalDigits = numberOfDigits( a, SIZE );

   /* put elements in buckets for sorting
      one sorted, get elements from buckets */
   for ( i = 1; i <= totalDigits; i++ ) {
      distributeElements( a, bucket, i );
      collectElements( a, bucket );

      /* set all bucket contents to zero */
      if ( i != totalDigits ) {
         zeroBucket( bucket );
      } /* end if */

   } /* end for */

} /* end function bucketSort */

/* Determine the number of digits in the largest number */
int numberOfDigits( int b[], int arraySize )
{
   int largest = b[ 0 ]; /* assume first element is largest */
   int i;                /* loop counter */
   int digits = 0;       /* total number of digits */

   /* find largest array element */
   for ( i = 1; i < arraySize; i++ ) {

      if ( b[ i ] > largest ) {
         largest = b[ i ];
      } /* end if */

   } /* end for */

   /* find number of digits of largest element */
   while ( largest != 0 ) {
      ++digits;
```

```
      largest /= 10;
   } /* end while */

   return digits; /* return number of digits */

} /* end function numberOfDigits */

/* Distribute elements into buckets based on specified digit */
void distributeElements( int a[], int buckets[][ SIZE ], int digit )
{
   int divisor = 10;  /* used to get specific digit */
   int i;             /* loop counter */
   int bucketNumber;  /* current bucket number */
   int elementNumber; /* current element number */

   /* determine the divisor */
   for ( i = 1; i < digit; i++ ) {
      divisor *= 10;
   } /* end for */

   /* bucketNumber example for hundreds digit: */
   /* ( 1234 % 1000 - 1234 % 100 ) / 100 --> 2   */
   for ( i = 0; i < SIZE; i++ ) {
      bucketNumber = ( a[ i ] % divisor - a[ i ] % ( 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[ i ];
   } /* end for */

} /* end function distributeElements */

/* Return elements to original array */
void collectElements( int a[], int buckets[][ SIZE ] )
{
   int i;             /* loop counter */
   int j;             /* loop counter */
   int subscript = 0; /* current subscript */

   /* retrieve elements from buckets */
   for ( i = 0; i < 10; i++ ) {

      for ( j = 1; j <= buckets[ i ][ 0 ]; j++ ) {
         a[ subscript++ ] = buckets[ i ][ j ];
      } /* end for */

   } /* end for */

} /* end function collectElements */

/* Set all buckets to zero */
void zeroBucket( int buckets[][ SIZE ] )
{
   int i; /* loop counter */
   int j; /* loop counter */

   for ( i = 0; i < 10; i++ ) {

      for ( j = 0; j < SIZE; j++ ) {
         buckets[ i ][ j ] = 0;
      } /* end for */

   } /* end for */

} /* end function zeroBucket */
```

```
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

6.32 (*Selection Sort*) A selection sort searches an array looking for the smallest element in the array. When the smallest element is found, it is swapped with the first element of the array. The process is then repeated for the subarray beginning with the second element of the array. Each pass of the array results in one element being placed in its proper location. This sort requires similar processing capabilities 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 a recursive function `selectionSort` to perform this algorithm.

ANS:

```c
/* Exercise 6.32 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXRANGE 1000
#define SIZE 10

void selectionSort( int array[], int size ); /* function prototype */

int main()
{
   int sortThisArray[ SIZE ] = { 0 }; /* array to be sorted */
   int loop; /* loop counter */

   srand( time( NULL ) ); /* seed random number generator */

   /* fill array with random numbers between 1-1000 */
   for ( loop = 0; loop < SIZE; loop++ ) {
      sortThisArray[ loop ] = 1 + rand() % MAXRANGE;
   } /* end for */

   printf( "\nUnsorted array is:\n" );

   /* display unsorted array */
   for ( loop = 0; loop < SIZE; loop++ ) {
      printf( " %d ", sortThisArray[ loop ] );
   } /* end for */

   selectionSort( sortThisArray, SIZE ); /* sort array */

   printf( "\n\nSorted array is:\n" );

   /* display sorted array */
   for ( loop = 0; loop < SIZE; loop++ ) {
      printf( " %d ", sortThisArray[ loop ] );
   } /* end for */

   printf( "\n\n" );

   return 0; /* indicate successful termination */

} /* end main */

/* function to sort an array */
void selectionSort( int array[], int size )
{
   int temp; /* temporary variable used for swapping */
```

```
   int loop; /* loop counter */

   /* sort array until only one element is left */
   if ( size >= 1 ) {

      /* find smallest element and put it in first position */
      for ( loop = 0; loop <= size - 1; loop++ ) {

         /* swap elements */
         if ( array[ loop ] < array[ 0 ] ) {
            temp = array[ loop ];
            array[ loop ] = array[ 0 ];
            array[ 0 ] = temp;
         } /* end if */

      } /* end for */

      /* recursive call to selectionSort */
      selectionSort( &array[ 1 ], size - 1 );
   } /* end for */

} /* end function selectionSort */
```

```
Unsorted array is:
 629  748  87  955  484  505  799  377  11  287

Sorted array is:
 11  87  287  377  484  505  629  748  799  955
```

6.33 (*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 you ignore blanks, “a man a plan a canal panama.” Write a recursive function `testPalindrome` that returns 1 if the string stored in the array is a palindrome and 0 otherwise. The function should ignore spaces and punctuation in the string.

ANS:

```
/* Exercise 6.33 solution */
#include <stdio.h>
#define SIZE 80

/* function prototype */
int testPalindrome( char array[], int left, int right );

int main()
{
   char c;                /* temporarily holds keyboard input */
   char string[ SIZE ]; /* original string */
   char copy[ SIZE ];   /* copy of string without spaces */
   int count = 0;       /* length of string */
   int copyCount;       /* length of copy */
   int i;               /* counter */

   printf( "Enter a sentence:\n" );

   /* get sentence to test from user */
   while ( ( c = getchar() ) != '\n' && count < SIZE ) {
      string[ count++ ] = c;
   } /* end while */

   string[ count ] = '\0'; /* terminate string */

   /* make a copy of string without spaces */
   for ( copyCount = 0, i = 0; string[ i ] != '\0'; i++ ) {

      if ( string[ i ] != ' ' ) {
         copy[ copyCount++ ] = string[ i ];
      } /* end if */

   } /* end for */

   /* print whether or not the sentence is a palindrome */
   if ( testPalindrome( copy, 0, copyCount - 1 ) ) {
      printf( "\"%s\" is a palindrome\n", string );
   } /* end if */
   else {
      printf( "\"%s\" is not a palindrome\n", string );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */

/* function to see if the sentence is a palindrome */
int testPalindrome( char array[], int left, int right )
{

   /* test array to see if a palindrome */
   if ( left == right || left > right ) {
      return 1;
   } /* end if */
   else if ( array[ left ] != array[ right ] ) {
      return 0;
   } /* end else if */
   else {
      return testPalindrome( array, left + 1, right - 1 );
   } /* end else */

} /* end function testPalindrome */
```

```
Enter a sentence:
able was i ere i saw elba
"able was i ere i saw elba" is a palindrome
```

```
Enter a sentence:
hi there
"hi there" is not a palindrome
```

6.34 (*Linear Search*) Modify the program of Fig. 6.18 to use a recursive `linearSearch` function to perform the 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.

ANS:

```
/* Exercise 6.34 Solution */
#include <stdio.h>
#define SIZE 100

/* function prototypes */
int linearSearch( int array[], int key, int low, int high );

int main()
{
   int array[ SIZE ]; /* array to be searched */
   int loop;          /* loop counter */
   int searchKey;     /* element to search for */
   int element;       /* result of linear search */

   /* initialize array elements */
   for ( loop = 0; loop < SIZE; loop++ ) {
      array[ loop ] = 2 * loop;
   } /* end for */

   /* obtain search key from user */
   printf( "Enter the integer search key: " );
   scanf( "%d", &searchKey );

   /* search array for search key */
   element = linearSearch( array, searchKey, 0, SIZE - 1 );

   /* display message if search key was found */
   if ( element != -1 ) {
      printf( "Found value in element %d\n", element );
   } /* end if */
   else {
      printf( "Value not found\n" );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */

/* function to search array for specified key */
int linearSearch( int array[], int key, int low, int high )
{

   /* recursively search array */
   if ( array[ low ] == key ) {
      return low;
   } /* end if */
   else if ( low == high ) {
      return -1;
   } /* end else if */
   else { /* recursive call */
      return linearSearch( array, key, low + 1, high );
   } /* end else */

} /* end function linearSearch */
```

```
Enter the integer search key: 8
Found value in element 4
```

```
Enter the integer search key: 48
Found value in element 24
```

```
Enter the integer search key: 99
Value not found
```

6.35 (*Binary Search*) Modify the program of Fig. 6.19 to use a recursive `binarySearch` function 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.

ANS:

```
/* Exercise 6.35 Solution */
#include <stdio.h>
#define SIZE 15

/* function prototypes */
int binarySearch( int b[], int searchKey, int low, int high );
void printHeader( void );
void printRow( int b[], int low, int mid, int high );

int main()
{
   int a[ SIZE ]; /* array to be searched */
   int i;         /* loop counter */
   int key;       /* search key */
   int result;    /* result of search */

   /* initialize array elements */
   for ( i = 0; i < SIZE; i++ ) {
      a[ i ] = 2 * i;
   } /* end for */

   /* obtain key from user */
   printf( "Enter a number between 0 and 28: " );
   scanf( "%d", &key );

   printHeader();

   /* search array for key */
   result = binarySearch( a, key, 0, SIZE - 1 );

   /* display results of the search */
   if ( result != -1 ) {
      printf( "\n%d found in array element %d\n", key, result );
   } /* end if */
   else {
      printf( "\n%d not found\n", key );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */

/* function to search array for specified key */
int binarySearch( int b[], int searchKey, int low, int high )
{
   int middle; /* middle of array */

   /* find middle of array and print current subarray */
   if ( low <= high ) {
      middle = ( low + high ) / 2;
      printRow( b, low, middle, high );

      /* determine if middle element is the key and if not,
         recursively call binarySearch */
      if ( searchKey == b[ middle ] ) {
         return middle;
      } /* end if */
      else if ( searchKey < b[ middle ] ) {
         /* recursive call on bottom half of array */
         return binarySearch( b, searchKey, low, middle - 1 );
      } /* end else if */
```

```
      else {
         /* recursive call on upper half of array */
         return binarySearch( b, searchKey, middle + 1, high );
      } /* end else */

   } /* end if */

   return -1; /* searchKey not found */

} /* end function binarySearch */

/* Print a header for the output */
void printHeader( void )
{
   int i; /* loop counter */

   printf( "\nSubscripts:\n" );

   /* print subscripts of array */
   for ( i = 0; i < SIZE; i++ ) {
      printf( "%3d ", i );
   } /* end for */

   printf( "\n" );

   /* print dividing line */
   for ( i = 1; i <= 4 * SIZE; i++ ) {
      printf( "-" );
   } /* end for */

   printf( "\n" );
} /* end function printHeader */

/* print one row of output showing the current
   part of the array being processed.  */
void printRow( int b[], int low, int mid, int high )
{
   int i; /* loop counter */

   /* print subarray currently being processed */
   for ( i = 0; i < SIZE; i++ ) {

      if ( i < low || i > high ) {
         printf( "    " );
      } /* end if */
      else if ( i == mid ) { /* mark middle value */
         printf( "%3d*", b[ i ] );
      } /* end else if */
      else {
         printf( "%3d ", b[ i ] );
      } /* end else */

   } /* end for */

   printf( "\n" );
} /* end function printRow */
```

```
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
```

```
Enter a number between 0 and 28: 10

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
  0   2   4   6*  8  10  12
                  8  10* 12

10 found in array element 5
```

6.36 (*Eight Queens)* Modify the Eight Queens program you created in Exercise 6.26 to solve the problem recursively.

6.37 (*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.

ANS:

```
/* Exercise 6.37 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define SIZE 10

/* function prototype */
void printArray( int array[], int low, int high );

int main()
{
   int array[ SIZE ]; /* array to be printed */
   int loop;          /* loop counter */

   srand( time( NULL ) );

   /* initialize array elements to random numbers */
   for ( loop = 0; loop < SIZE; loop++ ) {
      array[ loop ] = 1 + rand() % 500;
   } /* end for */

   printf( "Array values printed in main:\n" );

   /* print array elements */
   for ( loop = 0; loop < SIZE; loop++ ) {
      printf( "%d  ", array[ loop ] );
   } /* end for */

   printf( "\n\nArray values printed in printArray:\n" );
   printArray( array, 0, SIZE - 1 );
   printf( "\n" );

   return 0; /* indicate successful termination */

} /* end main */

/* function to recursively print an array */
void printArray( int array[], int low, int high )
{
   /* print first element of array passed */
   printf( "%d  ", array[ low ] );

   /* return if array only has 1 element */
   if ( low == high ) {
      return;
   } /* end if */
   else { /* call printArray with new subarray */
      printArray( array, low + 1, high );
   } /* end else */

} /* end function printArray */
```

```
Array values printed in main:
22  180  7  321  486  366  69  304  273  213

Array values printed in printArray:
22  180  7  321  486  366  69  304  273  213
```

6.38 (*Print a string backwards*) Write a recursive function `stringReverse` that takes a character array as an argument, prints it back to front and returns nothing. The function should stop processing and return when the terminating null character of the string is encountered.

ANS:

```c
/* Exercise 6.38 Solution */
#include <stdio.h>
#define SIZE 30

void stringReverse( char strArray[] ); /* function prototype */

int main()
{
   int loop; /* loop counter */

   /* initialize string strArray */
   char strArray[ SIZE ] = "Print this string backwards.";

   /* display original string */
   for ( loop = 0; loop < SIZE; loop++ ) {
      printf( "%c", strArray[ loop ] );
   } /* end for */

   printf( "\n" );
   stringReverse( strArray ); /* reverse the string */
   printf( "\n" );

   return 0; /* indicate successful termination */

} /* end main */

/* function to reverse a string */
void stringReverse( char strArray[] )
{

   /* return when null character is encountered */
   if ( strArray[ 0 ] == '\0' ) {
      return;
   } /* end if */

   /* recursively call stringReverse with new substring */
   stringReverse( &strArray[ 1 ] );
   printf( "%c", strArray[ 0 ] ); /* output string elements */
} /* end function stringReverse */
```

```
Print this string backwards.
.sdrawkcab gnirts siht tnirP
```

6.39 (*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 it receives an array of one element.

ANS:

```
/* Exercise 6.39 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 10
#define MAXRANGE 1000

/* function prototype */
int recursiveMinimum( int array[], int low, int high );

int main()
{
   int array[ SIZE ]; /* array to be searched */
   int loop;          /* loop counter */
   int smallest;      /* smallest element */

   srand( time( NULL ) );

   /* initialize elements of array to random numbers */
   for ( loop = 0; loop < SIZE; loop++ ) {
      array[ loop ] = 1 + rand() % MAXRANGE;
   } /* end for */

   printf( "Array members are:\n" );

   /* display array */
   for ( loop = 0; loop < SIZE; loop++ ) {
      printf( " %d ", array[ loop ] );
   } /* end for */

   /* find and display smallest array element */
   printf( "\n" );
   smallest = recursiveMinimum( array, 0, SIZE - 1 );
   printf( "\nSmallest element is: %d\n", smallest );

   return 0; /* indicate successful termination */

} /* end main */

/* function to recursively find minimum array element */
int recursiveMinimum( int array[], int low, int high )
{
   static int smallest = MAXRANGE; /* largest possible value */

   /* if first element of array is smallest so far,
      set smallest equal to that element */
   if ( array[ low ] < smallest ) {
      smallest = array[ low ];
   } /* end if */

   /* if only one element in array, return smallest */
   if ( low == high ) {
      return smallest;
   } /* end if */
   else { /* recursively call recursiveMinimum with new subarray */
      return recursiveMinimum( array, low + 1, high );
   } /* end else */

} /* end function recursiveMinimum */
```

```
Array members are:
 666  251  624  359  577  837  992  197  249  492

Smallest element is: 197
```

7

Pointers: Solutions

SOLUTIONS

7.7 Answer each of the following:

a) The __________ operator returns the location in memory where its operand is stored.

ANS: address (&).

b) The __________ operator returns the value of the object to which its operand points.

ANS: indirection (*).

c) To simulate call-by-reference when passing a nonarray variable to a function, it is necessary to pass the __________ of the variable to the function.

ANS: address.

7.8 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. It is not possible to know where these arrays will be stored in advance.

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. Array names cannot be modified to point to another location in memory.

7.9 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) Define an array of type `unsigned int` called `values` with five elements, and initialize the elements to the even integers from `2` to `10`. Assume the symbolic constant `SIZE` has been defined as `5`.

ANS: `unsigned int values[ SIZE ] = { 2, 4, 6, 8, 10 };`

b) Define a pointer `vPtr` that points to an object of type `unsigned int`.

ANS: `unsigned int *vPtr;`

c) Print the elements of array `values` using array subscript notation. Use a `for` statement and assume integer control variable `i` has been defined.

ANS:

```
for ( i = 0; i < SIZE; i++ )
   printf( "%d ", values[ i ] );
```

d) Give two separate statements that assign the starting address of array `values` to pointer variable `vPtr`.

ANS:

```
1) vPtr = values;
2) vPtr = &values[ 0 ];
```

e) Print the elements of array `values` using pointer/offset notation.

ANS:
```
for ( i = 0; i < SIZE; i++ )
    printf( "%d", *( 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++ )
    printf( "%d", *( values + i ) );
```
g) Print the elements of array `values` by subscripting the pointer to the array.

ANS:
```
for ( i = 0; i < SIZE; i++ )
    printf( "%d", vPtr[ i ] );
```
h) Refer to element 5 of array `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: `1002506; 8.`

j) Assuming `vPtr` points to `values[ 4 ]`, what address is referenced by `vPtr -= 4`. What value is stored at that location?

ANS: `1002500; 2.`

7.10 For each of the following, write a single statement that performs the indicated task. Assume that long integer variables `value1` and `value2` have been defined and that `value1` has been initialized to `200000`.

a) Define 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: `printf( "%ld\n", *lPtr );`

d) Assign the value of the object pointed to by `lPtr` to variable `value2`.

ANS: `value2 = *lPtr;`

e) Print the value of `value2`.

ANS: `printf( "%ld\n", value2 );`

f) Print the address of `value1`.

ANS: `printf( "%p\n", &value1 );`

g) Print the address stored in `lPtr`. Is the value printed the same as the address of `value1`?

ANS: `printf( "%p\n", lPtr); /* The value is the same */`

7.11 Do each of the following.

a) Write the function header for function `zero`, which takes a long integer array parameter `bigIntegers` and does not return a value.

ANS: `void zero( long int *bigIntegers);`

b) Write the function prototype for the function in *Part a*.

ANS: `void zero( long int * );`

c) Write the function header for function `add1AndSum`, which takes an integer array parameter `oneTooSmall` and returns an integer.

ANS: `int add1AndSum( int *oneTooSmall );`

d) Write the function prototype for the function described in *Part c*.

ANS: `int add1AndSum( int * );`

Note: Exercise 7.12 through Exercise 7.15 are reasonably challenging. Once you have done these problems, you ought to be able to implement most popular card games easily.

7.12 Modify the program in Fig. 7.24 so that the card-dealing function deals a five-card poker hand. Then write the following additional functions:

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).

7.13 Use the functions developed in Exercise 7.12 to write a program that deals two five-card poker hands, evaluates each hand, and determines which is the better hand.

7.14 Modify the program developed in Exercise 7.13 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 re-evaluate the dealer's hand. [*Caution:* This is a difficult problem!]

7.15 Modify the program developed in Exercise 7.14 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?

7.16 In the card shuffling and dealing program of Fig. 7.24, 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 the program of Fig. 7.24 as follows. Begin by initializing the `deck` array as shown in Fig. 7.29. 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. 7.30, for example). You may want your program to call the `shuffle` function several times to ensure a satisfactory shuffle.

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. 7.29 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. 7.30 Sample shuffled `deck` array.

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. 7.24 so that once a card is dealt, no further attempts are made to match that card number, and the program immediately proceeds with dealing the next card. In Chapter 10, we develop a dealing algorithm that requires only one operation per card.

ANS:

```c
/* Exercise 7.16 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* function prototypes */
void shuffle( int workDeck[][ 13 ] );
void deal( int workDeck[][ 13 ], char *workFace[], char *workSuit[] );

int main()
{
   int card = 1;         /* card counter */
   int row;              /* loop counter */
   int column;           /* loop counter */
   int deck[ 4 ][ 13 ]; /* array of cards */

   /* define arrays of card suits and faces */
   char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades"};
   char *face[ 13 ] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
      "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };

   srand( time( NULL ) );

   /* initialize deck */
   for ( row = 0; row <= 3; row++ ) {

      for ( column = 0; column <= 12; column++ ) {
         deck[ row ][ column ] = card++;
      } /* end for */

   } /* end for */

   shuffle( deck );
   deal( deck, face, suit );

   return 0; /* indicate successful termination */

} /* end main */

/* introduce another way to shuffle */
void shuffle( int workDeck[][ 13 ] )
{
   int temp;       /* temporary holder */
   int row;        /* loop counter */
   int column;     /* loop counter */
   int randRow;    /* random suit */
   int randColumn; /* random face */

   /* run through the loop and touch every element once */
   for ( row = 0; row <= 3; row++ ) {

      for ( column = 0; column <= 12; column++ ) {

```

```
         /* generate a random card */
         randRow = rand() % 4;
         randColumn = rand() % 13;

         /* swap random card with current card */
         temp = workDeck[ row ][ column ];
         workDeck[ row ][ column ] = workDeck[ randRow ][ randColumn ];
         workDeck[ randRow ][ randColumn ] = temp;
      } /* end for */

   } /* end for */

} /* end function shuffle */

/* deal the cards */
void deal( int workDeck2[][ 13 ], char *workFace[], char *workSuit[] )
{
   int card;   /* card counter */
   int row;    /* loop counter */
   int column; /* loop counter */

   /* loop through and print the cards */
   for ( card = 1; card <= 52; card++ ) {

      /* loop through rows */
      for ( row = 0; row <= 3; row++ ) {

         /* loop through columns */
         for ( column = 0; column <= 12; column++ ) {

            /* if current card equals card then deal */
            if ( workDeck2[ row ][ column ] == card ) {
               printf( "%5s of %-8s", workFace[ column ], workSuit[ row ] );
               card % 2 == 0 ? putchar( '\n' ) : putchar( '\t' );
               break; /* break loop */
            } /* end if */

         } /* end for */

      } /* end for */

   } /* end for */

} /* end function deal */
```

```
Eight of Spades          Ace of Spades
 Five of Hearts          Ace of Hearts
Eight of Diamonds      Queen of Spades
Deuce of Hearts        Seven of Hearts
Seven of Clubs           Six of Hearts
 Four of Clubs           Ace of Clubs
  Six of Spades          Ten of Diamonds
  Ten of Hearts         King of Hearts
 Four of Diamonds       Four of Hearts
 Jack of Diamonds      Three of Diamonds
Deuce of Spades        Queen of Clubs
Three of Hearts          Six of Clubs
 Nine of Hearts         Nine of Diamonds
 King of Spades        Seven of Diamonds
 Five of Spades        Seven of Spades
 Four of Spades          Ten of Spades
 King of Diamonds       Nine of Spades
Deuce of Clubs          Jack of Hearts
  Ace of Diamonds        Ten of Clubs
Eight of Hearts          Six of Diamonds
 Nine of Clubs          Five of Diamonds
Three of Clubs         Deuce of Diamonds
Queen of Hearts         King of Clubs
Queen of Diamonds       Jack of Clubs
 Five of Clubs         Three of Spades
 Jack of Spades        Eight of Clubs
```

7.17 (*Simulation: The Tortoise and the Hare*) In this problem, you will recreate one of the truly great moments in history, namely 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 lettuce. 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 rules of Fig. 7.31.

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 !!!!!
```

Then, for each tick of the clock (i.e., each repetition of a loop), print a 70 position line showing the letter `T` in the position of the tortoise and the letter `H` in the position of the hare. Occasionally, the contenders will 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 each line is printed, test if either animal has reached or passed square 70. If so, then 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 tick of the clock, 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 at how involved your audience gets!

ANS:

```c
/* Exercise 7.17 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* function prototypes */
void moveTortoise( int *turtlePtr );
void moveHare( int *rabbitPtr );
void printCurrentPositions( int *snapperPtr, int *bunnyPtr );

int main()
{
   int tortoise = 1; /* tortoise current position */
   int hare = 1;     /* hare current position */
   int timer = 0;    /* time elapsed during race */

```

```
   srand( time( NULL ) );

   printf( "ON YOUR MARK, GET SET\n" );
   printf( "BANG               !!!!\n" );
   printf( "AND THEY'RE OFF    !!!!\n" );

   /* loop through the events */
   while ( tortoise != 70 && hare != 70 ) {
      moveTortoise( &tortoise );
      moveHare( &hare );
      printCurrentPositions( &tortoise, &hare );
      ++timer;
   } /* end while */

   /* determine the winner and print message */
   if ( tortoise >= hare ) {
      printf( "\nTORTOISE WINS!!! YAY!!!\n" );
   } /* end if */
   else {
      printf( "Hare wins. Yuch.\n" );
   } /* end else */

   printf( "TIME ELAPSED = %d seconds", timer );

   return 0; /* indicate successful termination */

} /* end main */

/* progress for the tortoise */
void moveTortoise( int *turtlePtr )
{
   int x; /* random number */

   x = rand() % 10 + 1; /* generate random number from 1-10 */

   /* determine progress */
   if ( x >= 1 && x <= 5 ) { /* fast plod */
      *turtlePtr += 3;
   } /* end if */
   else if ( x == 6 || x == 7 ) { /* slip */
      *turtlePtr -= 6;
   } /* end else if */
   else { /* slow plod */
      ++( *turtlePtr );
   } /* end else */

   /* check boundaries */
   if ( *turtlePtr < 1 ) {
      *turtlePtr = 1;
   } /* end if */
   if ( *turtlePtr > 70 ) {
      *turtlePtr = 70;
   } /* end if */

} /* end function moveTortoise */

/* progress for the hare */
void moveHare( int *rabbitPtr )
{
   int y; /* random number */

```

```
   y = rand() % 10 + 1; /* generate random number from 1-10 */

   /* determine progress */
   if ( y == 3 || y == 4 ) { /* big hop */
      *rabbitPtr += 9;
   } /* end if */
   else if ( y == 5 ) { /* big slip */
      *rabbitPtr -= 12;
   } /* end else if */
   else if ( y >= 6 && y <= 8 ) { /* small hop */
      ++( *rabbitPtr );
   } /* end else if */
   else if ( y == 10 ) { /* small slip */
      *rabbitPtr -= 2;
   } /* end else if */

   /* check boundaries */
   if ( *rabbitPtr < 1 ) {
      *rabbitPtr = 1;
   } /* end if */

   if ( *rabbitPtr > 70 ) {
      *rabbitPtr = 70;
   } /* end if */

} /* end function moveHare */

/* display new position */
void printCurrentPositions( int *snapperPtr, int *bunnyPtr )
{
   int count; /* counter */

   /* loop through race */
   for ( count = 1; count <= 70; count++ )

      /* print current leader */
      if ( count == *snapperPtr && count == *bunnyPtr ) {
         printf( "OUCH!!!" );
      } /* end if */
      else if ( count == *bunnyPtr ) {
         printf( "H" );
      } /* end else if */
      else if ( count == *snapperPtr ) {
         printf( "T" );
      } /* end else if */
      else {
         printf( " " );
      } /* end else */

   printf( "\n" );
} /* end function printCurrentPositions */
```

```
ON YOUR MARK, GET SET
BANG               !!!!
AND THEY'RE OFF    !!!!
OUCH!!!
 H T
 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
                H      T
    H                   T
     H            T
      H              T
               H      T
               H         T
               H            T
                H            T
                H      T
    H                     T
H                   T
         H    T
        TH
         OUCH!!!
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
                                  TH
                                   T        H
                                      T             H
                                         T          H
                                            T         H
                                             T        H
                                                T      H
                                                   T            H
                                                      T              H
Hare wins. Yuch.
TIME ELAPSED = 88 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!

7.18 (*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`.

Each instruction written in SML occupies one word of the Simpletron's memory (and hence instructions are signed four-digit decimal numbers). We 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 either 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,* which specifies the operation to be performed. SML operation codes are summarized in Fig. Fig. 7.32.

Operation code	Meaning
Input/output operations:	
`#define READ 10`	Read a word from the terminal into a specific location in memory.
`#define WRITE 11`	Write a word from a specific location in memory to the terminal.
Load/store operations:	
`#define LOAD 20`	Load a word from a specific location in memory into the accumulator.
`#define STORE 21`	Store a word from the accumulator into a specific location in memory.
Arithmetic operations:	
`#define ADD 30`	Add a word from a specific location in memory to the word in the accumulator (leave result in accumulator).
`#define SUBTRACT 31`	Subtract a word from a specific location in memory from the word in the accumulator (leave result in accumulator).
`#define DIVIDE 32`	Divide a word from a specific location in memory into the word in the accumulator (leave result in accumulator).
`#define 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:	
`#define BRANCH 40`	Branch to a specific location in memory.
`#define BRANCHNEG 41`	Branch to a specific location in memory if the accumulator is negative.
`#define BRANCHZERO 42`	Branch to a specific location in memory if the accumulator is zero.
`#define HALT 43`	Halt—i.e., the program has completed its task.

Fig. 7.32 Simpletron Machine Language (SML) operation codes.

The last two digits of an SML instruction are the *operand,* which is the address of the memory location containing the word to which the operation applies. Now let us consider several simple SML programs.

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 preceding SML program 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 +1008 reads the next number into location 08. The *load* instruction, +2007, puts 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 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 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)*
08	+4300	*(Halt)*
09	+0000	*(Variable A)*
10	+0000	*(Variable B)*

The preceding SML program 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. Now write SML programs to accomplish each of the following tasks.

a) Use a sentinel-controlled loop to read 10 positive integers and compute and print their sum.

ANS:

```
00     +1009     (Read Value)
01     +2009     (Load Value)
02     +4106     (Branch negative to 06)
03     +3008     (Add Sum)
04     +2108     (Store Sum)
05     +4000     (Branch 00)
06     +1108     (Write Sum)
07     +4300     (Halt)
08     +0000     (Variable Sum)
09      +0000     (Variable Value)
```

b) Use a counter-controlled loop to read seven numbers, some positive and some negative, and compute and print their average.

ANS:

```
00     +2018     (Load Counter)
01     +3121     (Subtract Termination)
02     +4211     (Branch zero to 11)
03     +2018     (Load Counter)
04     +3019     (Add Increment)
05     +2118     (Store Counter)
06     +1017     (Read Value)
07     +2016     (Load Sum)
08     +3017     (Add Value)
09     +2116     (Store Sum)
10     +4000     (Branch 00)
11     +2016     (Load Sum)
12     +3218     (Divide Counter)
13     +2120     (Store Result)
14     +1120     (Write Result)
15     +4300     (Halt)
16     +0000     (Variable Sum)
17     +0000     (Variable Value)
18     +0000     (Variable Counter)
19     +0001     (Variable Increment)
20     +0000     (Variable Result)
21      +0007     (Variable Termination)
```

c) Read a series of numbers and determine and print the largest number. The first number read indicates how many numbers should be processed.

ANS:

```
00     +1017     (Read Endvalue)
01     +2018     (Load Counter)
02     +3117     (Subtract Endvalue)
03     +4215     (Branch zero to 15)
04     +2018     (Load Counter)
05     +3021     (Add Increment)
06     +2118     (Store Counter)
07     +1019     (Read Value)
08     +2020     (Load Largest)
09     +3119     (Subtract Value)
10     +4112     (Branch negative to 12)
11     +4001     (Branch 01)
12     +2019     (Load Value)
13     +2120     (Store Largest)
14     +4001     (Branch 01)
15     +1120     (Write Largest)
16     +4300     (Halt)
17     +0000     (Variable Endvalue)
18     +0000     (Variable Counter)
19     +0000     (Variable Value)
20     +0000     (Variable Largest)
21      +0001     (Variable Increment)
```

7.19 (*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 7.18.

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 7.18:

```
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) into the 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 `instructionCounter` to keep track of the location in memory that contains the instruction being performed. Use the variable `operationCode` to indicate the operation currently being performed—i.e., the left two digits of the instruction word. Use the 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 the variable `operationCode,` and "pick off" the right two digits and place them in `operand`.

When Simpletron begins execution, the special registers are initialized as follows:

```
accumulator            +0000
instructionCounter        00
instructionRegister    +0000
operationCode             00
operand                   00
```

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 `instructionCounter` 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[ instructionCounter ];
```

The operation code and the 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` statement, the behavior of various SML instructions is simulated as follows (we leave the others to the reader):

read: `scanf( "%d", &memory[ operand ] );`
load: `accumulator = memory[ operand ];`
add: `accumulator += memory[ operand ];`
Various branch instructions: We'll discuss these shortly.
halt: This instruction prints the message

```
*** Simpletron execution terminated ***
```

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*. To help you program your dump function, a sample dump format is shown in Fig. Fig. 7.33. Note that a dump after executing a Simpletron program would show the actual values of instructions and data values at the moment execution terminated.

```
REGISTERS:
accumulator            +0000
instructionCounter        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. 7.33 Sample dump of Simpletron's memory.

Let us proceed with the execution of our program's first instruction, namely the `+1009` in location `00`. As we have indicated, the `switch` statement simulates this by performing the C statement

```
scanf( "%d", &memory[ operand ] );
```

A question mark (`?`) should be displayed on the screen before the `scanf` 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:

```
++instructionCounter;
```

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

```
instructionCounter = operand;
```

The conditional "branch if accumulator is zero" instruction is simulated as

```
if ( accumulator == 0 )
    instructionCounter = operand;
```

At this point, you should implement your Simpletron simulator and run the SML programs you wrote in Exercise 7.18. 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 correct number.

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 and accumulator overflows (i.e., arithmetic operations resulting in values larger than `+9999` or smaller than `-9999`). 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 the program.

ANS:

```
/* Exercise 7.19 Solution */
#include <stdio.h>

/* define commands */
#define SIZE 100
#define SENTINEL -99999
#define TRUE 1
#define FALSE 0
#define READ 10
#define WRITE 11
#define LOAD 20
#define STORE 21
#define ADD 30
#define SUBTRACT 31
#define DIVIDE 32
#define MULTIPLY 33
#define BRANCH 40
#define BRANCHNEG 41
#define BRANCHZERO 42
#define HALT 43

/* function prototypes */
void load( int *loadMemory );
void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
             int *opCodePtr, int *opPtr );
void dump( int *memory, int accumulator, int instructionCounter,
          int instructionRegister, int operationCode,
          int operand );
int validWord( int word );

int main()
{
   int memory[ SIZE ]; /* define memory array */
   int ac = 0;         /* accumulator */
   int ic = 0;         /* instruction counter */
   int opCode = 0;     /* operation code */
   int op = 0;         /* operand */
   int ir = 0;         /* instruction register */
   int i;              /* counter */

```

```
   /* clear memory */
   for ( i = 0; i < SIZE; i++ ) {
      memory[ i ] = 0;
   } /* end for */

   load( memory );
   execute( memory, &ac, &ic, &ir, &opCode, &op );
   dump( memory, ac, ic, ir, opCode, op );

   return 0; /* indicate successful termination */

} /* end main */

/* function loads instructions */
void load( int *loadMemory )
{
   long int instruction; /* current instruction */
   int i = 0;            /* indexing variable */

   printf( "%s\n\n%s\n%s\n%s\n%s\n%s\n%s\n\n",
          "***           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.                             ***" );

   printf( "00 ? " );
   scanf( "%ld", &instruction ); /* read instruction */

   /* while sentinel is not read from user */
   while ( instruction != SENTINEL ) {

      /* test instruction for validity */
      if ( !validWord( instruction ) ) {
         printf( "Number out of range. Please enter again.\n" );
      } /* end if */
      else { /* load instruction */
         loadMemory[ i++ ] = instruction;
      } /* end else */

      printf( "%02d ? ", i );
      scanf( "%ld", &instruction );
   } /* end while */

} /* end function load */

/* carry out the commands */
void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
              int *opCodePtr, int *opPtr )
{
   int fatal = FALSE; /* fatal error flag */
   int temp;          /* temporary holding space */

   printf( "\n************START SIMPLETRON EXECUTION************\n\n" );

   /* separate operation code and operand */
   *irPtr = memory[ *icPtr ];
   *opCodePtr = *irPtr / 100;
   *opPtr = *irPtr % 100;
```

```

   /* loop while command is not HALT or fatal */
   while ( *opCodePtr != HALT && !fatal ) {

      /* determine appropriate action */
      switch ( *opCodePtr ) {

         /* read data into location in memory */
         case READ:
            printf( "Enter an integer: " );
            scanf( "%d", &temp );

            /* check for validity */
            while ( !validWord( temp ) ) {
               printf( "Number out of range. Please enter again: " );
               scanf( "%d", &temp );
            } /* end while */

            memory[ *opPtr ] = temp; /* write to memory */
            ++( *icPtr );
            break; /* exit switch */

         /* write data from memory to screen */
         case WRITE:
            printf( "Contents of %02d: %d\n", *opPtr, memory[ *opPtr ] );
            ++( *icPtr );
            break; /* exit switch */

         /* load data from memory into accumulator */
         case LOAD:
            *acPtr = memory[ *opPtr ];
            ++( *icPtr );
            break; /* exit switch */

         /* store data from accumulator into memory */
         case STORE:
            memory[ *opPtr ] = *acPtr;
            ++( *icPtr );
            break; /* exit switch */

         /* add data from memory to data in accumulator */
         case ADD:
            temp = *acPtr + memory[ *opPtr ];

            /* check validity */
            if ( !validWord( temp ) ) {
               printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
               printf( "*** Simpletron execution abnormally terminated ***\n" );
               fatal = TRUE;
            } /* end if */
            else {
               *acPtr = temp;
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         /* subtract data in memory from data in accumulator */
         case SUBTRACT:
            temp = *acPtr - memory[ *opPtr ];

```

```
            /* check validity */
            if ( !validWord( temp ) ) {
               printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
               printf( "*** Simpletron execution abnormally terminated ***\n" );
               fatal = TRUE;
            } /* end if */
            else {
               *acPtr = temp;
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         /* divide data in memory into data in accumulator */
         case DIVIDE:

            /* check for divide by zero error */
            if ( memory[ *opPtr ] == 0 ) {
               printf( "*** FATAL ERROR: Attempt to divide by zero     ***\n" );
               printf( "*** Simpletron execution abnormally terminated ***\n" );
               fatal = TRUE;
            } /* end if */
            else {
               *acPtr /= memory[ *opPtr ];
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         /* multiple data in memory by data in accumulator */
         case MULTIPLY:
            temp = *acPtr * memory[ *opPtr ];

            /* check validity */
            if ( !validWord( temp ) ) {
               printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
               printf( "*** Simpletron execution abnormally terminated ***\n" );
               fatal = TRUE;
            } /* end if */
            else {
               *acPtr = temp;
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         /* branch to specific location in memory */
         case BRANCH:
            *icPtr = *opPtr;
            break; /* exit switch */

         /* branch to location in memory if accumulator is negative */
         case BRANCHNEG:

            /* if accumulator is negative */
            if ( *acPtr < 0 ) {
               *icPtr = *opPtr;
            } /* end if */
            else {
               ++( *icPtr );
            } /* end else */
```

```

            break; /* exit switch */

         /* branch to location in memory if accumulator is zero */
         case BRANCHZERO:

            /* if accumulator is zero */
            if ( *acPtr == 0 ) {
               *icPtr = *opPtr;
            } /* end if */
            else {
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         default:
            printf( "*** FATAL ERROR: Invalid opcode detected       ***\n" );
            printf( "*** Simpletron execution abnormally terminated ***\n" );
            fatal = TRUE;
            break; /* exit switch */
      } /* end switch */

      /* separate next operation code and operand */
      *irPtr = memory[ *icPtr ];
      *opCodePtr = *irPtr / 100;
      *opPtr = *irPtr % 100;
   } /* end while */

   printf( "\n*************END SIMPLETRON EXECUTION*************\n" );
} /* end function execute */

/* print out name and content of each register and memory */
void dump( int *memory, int accumulator, int instructionCounter,
          int instructionRegister, int operationCode,
          int operand )
{
   int i; /* counter */

   printf( "\n%s\n%-23s%+05d\n%-23s%5.2d\n%-23s%+05d\n%-23s%5.2d\n%-23s%5.2d",
          "REGISTERS:", "accumulator", accumulator, "instructioncounter",
          instructionCounter, "instructionregister", instructionRegister,
          "operationcode", operationCode, "operand", operand );

   printf( "\n\nMEMORY:\n   " );

   /* print column headers */
   for ( i = 0; i <= 9; i++ ) {
      printf( "%5d ", i );
   } /* end for */

   /* print row headers and memory contents */
   for ( i = 0; i < SIZE; i++ ) {

      /* print in increments of 10 */
      if ( i % 10 == 0 ) {
         printf( "\n%2d ", i );
      } /* end if */

      printf( "%+05d ", memory[ i ] );
   } /* end for */
```

```

   printf( "\n" );
} /* end function dump */

/* function tests validity of word */
int validWord( int word )
{
   return word >= -9999 && word <= 9999;

} /* end function validWord */

```

```
***            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 ? 1007
01 ? 1008
02 ? 2007
03 ? 3008
04 ? 2109
05 ? 1109
06 ? 4300
07 ? 0000
08 ? 0000
09 ? 0000
10 ? -99999

************START SIMPLETRON EXECUTION************

Enter an integer: 23
Enter an integer: 17
Contents of 09: 40

*************END SIMPLETRON EXECUTION*************

REGISTERS:
accumulator            +0040
instructioncounter        06
instructionregister    +4300
operationcode             43
operand                   00

MEMORY:
       0     1     2     3     4     5     6     7     8     9
 0 +1007 +1008 +2007 +3008 +2109 +1109 +4300 +0023 +0017 +0040
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
```

7.20 Modify the card shuffling and dealing program of Fig. 7.24 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. 7.24.

ANS:

```c
/* Exercise 7.20 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* function prototype */
void shuffleAndDeal( int workdeck[][ 13 ], char *workface[],
                     char *worksuit[] );

int main()
{

   /* define card suit array and card face array */
   char *suit[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades"};
   char *face[ 13 ] = { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
                     "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
   int deck[ 4 ][ 13 ] = { 0 }; /* array of cards */

   srand( time( NULL ) );

   shuffleAndDeal( deck, face, suit );

   return 0; /* indicate successful termination */

} /* end main */

/* integrate shuffling and dealing operation */
void shuffleAndDeal( int workdeck[][ 13 ], char *workface[],
                     char *worksuit[] )
{
   int card;   /* card loop counter */
   int row;    /* current suit */
   int column; /* current face */

   /* loop through the deck of cards, shuffle and print */
   for ( card = 1; card <= 52; card++ ) {

      /* choose random card until not equal to zero */
      do {
         row = rand() % 4;
         column = rand() % 13;
      } while( workdeck[ row ][ column ] != 0 ); /* end do...while */

      workdeck[ row ][ column ] = card;

      /* deal card */
      printf( "%5s of %-8s", workface[ column ], worksuit[ row ] );

      card % 2 == 0 ? printf( "\n" ) : printf( "\t" );
   } /* end for */

} /* end function shuffleAndDeal */
```

```
Seven of Spades          King of Diamonds
  Six of Spades          King of Hearts
Three of Clubs          Three of Diamonds
 Jack of Diamonds        Jack of Spades
Queen of Clubs          Eight of Hearts
 Four of Hearts         Deuce of Clubs
  Six of Clubs          Eight of Spades
Three of Hearts          Five of Hearts
Seven of Clubs            Ace of Hearts
  Ten of Hearts          Five of Diamonds
Queen of Hearts         Eight of Clubs
 Five of Clubs          Deuce of Diamonds
Deuce of Spades          Jack of Hearts
  Ace of Clubs           Nine of Diamonds
 Five of Spades          Nine of Clubs
Deuce of Hearts          King of Spades
 Nine of Hearts         Queen of Spades
 King of Clubs           Four of Spades
Seven of Diamonds         Ace of Diamonds
  Six of Hearts           Ten of Diamonds
 Nine of Spades         Queen of Diamonds
Three of Spades          Jack of Clubs
 Four of Diamonds         Ace of Spades
  Ten of Spades           Ten of Clubs
 Four of Clubs          Eight of Diamonds
  Six of Diamonds       Seven of Hearts
```

7.21 What does this program do?

```
/* ex07_21.c */
/* What does this program do? */
#include <stdio.h>

void mystery1( char *s1, const char *s2 ); /* prototype */

int main()
{
   char string1[ 80 ]; /* create char array */
   char string2[ 80 ]; /* create char array */

   printf( "Enter two strings: " );
   scanf( "%s%s" , string1, string2 );

   mystery1( string1, string2 );

   printf("%s", string1 );

   return 0; /* indicates successful termination */

} /* end main */

/* What does this function do? */
void mystery1( char *s1, const char *s2 )
{
   while ( *s1 != '\0' ) {
      s1++;
   } /* end while */

   for ( ; *s1 = *s2; s1++, s2++ ) {
      ;   /* empty statement */
   } /* end for */

} /* end function mystery1 */
```

ANS: Concatenates strings.

```
Enter two strings: string1 string2
string1string2
```

7.22 What does this program do?

```
/* ex07_22.c */
/* what does this program do? */
#include <stdio.h>

int mystery2( const char *s ); /* prototype */

int main()
{
   char string[ 80 ]; /* create char array */

   printf( "Enter a string: ");
   scanf( "%s", string );

   printf( "%d\n", mystery2( string ) );

   return 0; /* indicates successful termination */

} /* end main */

/* What does this function do? */
int mystery2( const char *s )
{
   int x; /* counter */

   /* loop through string */
   for ( x = 0; *s != '\0'; s++ ) {
      x++;
   } /* end for */

   return x;

} /* end function mystery2 */
```

ANS: Determines the length of a string.

```
Enter a string: string1
7
```

7.23 Find the error in each of the following program segments. If the error can be corrected, explain how.

a)
```
int *number;
printf( "%d\n", *number );
```
ANS: `number` has not been assigned to point to a location in memory.

b)
```
float *realPtr;
long *integerPtr;
integerPtr = realPtr;
```
ANS: A pointer cannot be assigned to a different type, other than `void *`.

c)
```
int * x, y;
x = y;
```
ANS: There are two possible solutions. 1) The indirection operator (*) is not distributive and would be required for `y`, which would result in a valid pointer assignment. 2) `y` as it is defined is a valid integer variable, and would require the address operator (&) in the pointer assignment statement.

d)
```
char s[] = "this is a character array";
int count;
for ( ; *s != '\0'; s++)
   printf( "%c ", *s );
```
ANS: `s` should be defined as `char *`, a constant pointer cannot be moved.

e)
```
short *numPtr, result;
void *genericPtr = numPtr;
result = *genericPtr + 7;
```
ANS: A `void *` pointer cannot be dereferenced.

f)
```
float x = 19.34;
float xPtr = &x;
printf( "%f\n", xPtr );
```
ANS: `xPtr` is not defined as a pointer so it should be dereferenced as well.

g)
```
char *s;
printf( "%s\n", s );
```
ANS: `s` has not been assigned a value, it does not point to anything.

7.24 (*Quicksort*) In the examples and exercises of Chapter 6, we discussed the sorting techniques 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 to 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 by partitioning both subarrays in the same manner as the original array.

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.

ANS:

```
/* Exercise 7.24 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX 10

/* function prototypes */
void quicksort( int *array, int first, int last );
int partition( int *array, int left, int right );
void swap( int *ptr1, int *ptr2 );

int main()
{
   int loop; /* loop counter */
   int arrayToBeSorted[ MAX ] = { 0 }; /* array to sort */
```

```c

   srand( time( NULL ) );

   /* randomly generate content */
   for ( loop = 0; loop < MAX; loop++ ) {
      arrayToBeSorted[ loop ] = rand() % 1000;
   } /* end for */

   printf( "Initial array values are: \n" );

   /* print out values of the array */
   for ( loop = 0; loop < MAX; loop++ ) {
       printf( "%4d", arrayToBeSorted[ loop ] );
   } /* end for */

   printf( "\n\n" );

   /* if there is only one element */
   if ( MAX == 1 ) {
      printf( "Array is sorted: %d\n", arrayToBeSorted[ 0 ] );
   } /* end if */
   else { /* call quicksort */
      quicksort( arrayToBeSorted, 0, MAX - 1 );
      printf( "The sorted array values are:\n" );

      /* display sorted array */
      for ( loop = 0; loop < MAX; loop++ ) {
         printf( "%4d", arrayToBeSorted[ loop ] );
      } /* end for */

      printf( "\n" );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */

/* recursive function to sort array */
void quicksort( int array[], int first, int last )
{
   int currentLocation; /* current location in array */

   /* if array is sorted, return */
   if ( first >= last ) {
      return;
   } /* end if */

   currentLocation = partition( array, first, last ); /* place an element */
   quicksort( array, first, currentLocation - 1 ); /* sort left side   */
   quicksort( array, currentLocation + 1, last ); /* sort right side  */

} /* end function quicksort */

/* partition the array into multiple sections */
int partition( int array[], int left, int right )
{
   int position = left; /* final location of first element */

   /* infinite loop */
   while ( 1 ) {

```

```
      /* loop through the portion of the array */
      while ( array[ position ] <= array[ right ] &&
         position != right ) {
         --right;
      } /* end while */

      /* if correct position is found */
      if ( position == right ) {
         return position ;
      } /* end if */

      /* swap positions */
      if ( array[ position ] > array[ right ] ) {
         swap( &array[ position  ], &array[ right ] );
         position = right;
      } /* end if */

      /* loop through the portion of the array */
      while ( array[ left ] <= array[ position ] &&
         left != position ) {
         ++left;
      } /* end while */

      /* if correct position is found */
      if ( position == left ) {
         return position;
      } /* end if */

      /* swap positions */
      if ( array[ left ] > array[ position ] ) {
         swap( &array[ position ], &array[ left ] );
         position = left;
      } /* end if */

   } /* end while */

} /* end function partition */

/* swap locations */
void swap( int *ptr1, int *ptr2 )
{
   int temp; /* temporary holder */

   temp = *ptr1;
   *ptr1 = *ptr2;
   *ptr2 = temp;
} /* end function swap */
```

```
Initial array values are:
 276 980 550 654 811 764 571 469  12 161

The sorted array values are:
  12 161 276 469 550 571 654 764 811 980
```

7.25 (*Maze Traversal*) The following grid is a double-subscripted array representation of a maze.

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

The # symbols represent the walls of the maze, and the periods (.) represent squares in the possible paths through the maze.

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming 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.

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.

ANS:

```c
/* Exercise 7.25 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 <stdio.h>
#include <stdlib.h>

#define DOWN  0   /* move down  */
#define RIGHT 1   /* move right */
#define UP    2   /* move up    */
#define LEFT  3   /* move left  */

#define X_START 2 /* starting X and Y coordinate for maze */
#define Y_START 0

/* function prototypes */
void mazeTraversal( char maze[ 12 ][ 12 ], int xCoord, int yCoord,
                    int direction );
void printMaze( const char maze[][ 12 ] );
int validMove( const char maze[][ 12 ], int r, int c );
int coordsAreEdge( int x, int y );

int main()
{

   /* maze grid */
   char maze[ 12 ][ 12 ] =
   { { '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1'},
   { '1', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '1'},
   { '0', '0', '1', '0', '1', '0', '1', '1', '1', '1', '0', '1'},
   { '1', '1', '1', '0', '1', '0', '0', '0', '0', '1', '0', '1'},
```

```
      { '1', '0', '0', '0', '0', '1', '1', '1', '0', '1', '0', '0'},
      { '1', '1', '1', '1', '0', '1', '0', '1', '0', '1', '0', '1'},
      { '1', '0', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1'},
      { '1', '1', '0', '1', '0', '1', '0', '1', '0', '1', '0', '1'},
      { '1', '0', '0', '0', '0', '0', '0', '0', '0', '1', '0', '1'},
      { '1', '1', '1', '1', '1', '1', '0', '1', '1', '1', '0', '1'},
      { '1', '0', '0', '0', '0', '0', '0', '1', '0', '0', '0', '1'},
      { '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1' } };

   mazeTraversal( maze, X_START, Y_START, RIGHT );

   return 0; /* indicate successful termination */

} /* end main */

/* Assume that there is exactly 1 entrance and
   exactly 1 exit to the maze. */
void mazeTraversal( char maze[ 12 ][ 12 ], int xCoord, int yCoord,
                    int direction )
{
   static int flag = 0; /* starting position flag */

   maze[ xCoord ][ yCoord ] = 'X'; /* mark current point */
   printMaze( maze );

   /* if maze completed */
   if ( coordsAreEdge( xCoord, yCoord ) && xCoord != X_START &&
        yCoord != Y_START ) {
      printf( "\nMaze successfully exited!\n\n" );
      return;
   } /* end if */
   else if ( xCoord == X_START && yCoord == Y_START && flag == 1 ) {
      printf( "\nArrived back at the starting location.\n\n" );
      return;
   } /* end else if */
   else { /* make next move */
      int move;  /* next move */
      int count; /* counter */

      flag = 1;

      /* loop 4 times and find first valid move */
      for ( move = direction, count = 0; count < 4; ++count,
         ++move, move %= 4 ) {

         /* choose valid move */
         switch( move ) {

            case DOWN: /* move down */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, xCoord + 1, yCoord ) ) {
                  mazeTraversal( maze, xCoord + 1, yCoord, LEFT );
                  return;
               } /* end if */

               break; /* exit switch */

            case RIGHT: /* move right */

```

```
                  /* if move is valid, call mazeTraversal */
                  if ( validMove( maze, xCoord, yCoord + 1 ) ) {
                     mazeTraversal( maze, xCoord, yCoord + 1, DOWN );
                     return;
                  } /* end if */

                  break; /* exit switch */

               case UP: /* move up */

                  /* if move is valid, call mazeTraversal */
                  if ( validMove( maze, xCoord - 1, yCoord ) ) {
                     mazeTraversal( maze, xCoord - 1, yCoord, RIGHT );
                     return;
                  } /* end if */

                  break; /* exit switch */

               case LEFT: /* move left */

                  /* if move is valid, call mazeTraversal */
                  if ( validMove( maze, xCoord, yCoord - 1 ) ) {  /* move left  */
                     mazeTraversal( maze, xCoord, yCoord - 1, UP );
                     return;
                  } /* end if */

                  break; /* exit switch */
            } /* end switch */

         } /* end for */

      } /* end else */

} /* end function mazeTraversal */

/* validate move */
int validMove( const char maze[][ 12 ], int r, int c )
{
   return ( r >= 0 && r <= 11 && c >= 0 && c <= 11 &&
      maze[ r ][ c ] != '1' );

} /* end function validMove */

/* function to check coordinates */
int coordsAreEdge( int x, int y )
{

   /* if coordinate is not valid */
   if ( ( x == 0 || x == 11 ) && ( y >= 0 && y <= 11 ) ) {
      return 1;
   } /* end if */
   else if ( ( y == 0 || y == 11 ) && ( x >= 0 && x <= 11 ) ) {
      return 1;
   } /* end else if */
   else { /* coordinate is valid */
      return 0;
   } /* end else */

} /* end function coordsAreEdge */
```

```
/* print the current state of the maze */
void printMaze( const char maze[][ 12 ] )
{
   int x; /* row counter */
   int y; /* column counter */

   /* iterate through the maze */
   for ( x = 0; x < 12; x++ ) {

      for ( y = 0; y < 12; y++ ) {
         printf( "%c ", maze[ x ][ y ] );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\nHit return to see next move" );
   getchar();
} /* end function printMaze */
```

```
Hit return to see next move
1 1 1 1 1 1 1 1 1 1 1 1
1 X X X 1 X X X X X X 1
X X 1 X 1 X 1 1 1 1 0 1
1 1 1 X 1 X X X X 1 0 1
1 X X X X 1 1 1 X 1 0 0
1 1 1 1 X 1 0 1 X 1 0 1
1 X X 1 X 1 0 1 X 1 0 1
1 1 X 1 X 1 0 1 X 1 0 1
1 X X X X X X X X 1 0 1
1 1 1 1 1 1 X 1 1 1 0 1
1 X X X X X X 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1

Hit return to see next move
1 1 1 1 1 1 1 1 1 1 1 1
1 X X X 1 X X X X X X 1
X X 1 X 1 X 1 1 1 1 X 1
1 1 1 X 1 X X X X 1 0 1
1 X X X X 1 1 1 X 1 0 0
1 1 1 1 X 1 0 1 X 1 0 1
1 X X 1 X 1 0 1 X 1 0 1
1 1 X 1 X 1 0 1 X 1 0 1
1 X X X X X X X X 1 0 1
1 1 1 1 1 1 X 1 1 1 0 1
1 X X X X X X 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1

...

Hit return to see next move
1 1 1 1 1 1 1 1 1 1 1 1
1 X X X 1 X X X X X X 1
X X 1 X 1 X 1 1 1 1 X 1
1 1 1 X 1 X X X X 1 X 1
1 X X X X 1 1 1 X 1 X X
1 1 1 1 X 1 0 1 X 1 X 1
1 X X 1 X 1 0 1 X 1 X 1
1 1 X 1 X 1 0 1 X 1 X 1
1 X X X X X X X X 1 X 1
1 1 1 1 1 1 X 1 1 1 X 1
1 X X X X X X 1 X X X 1
1 1 1 1 1 1 1 1 1 1 1 1

Hit return to see next move

Maze successfully exited!
```

7.26 (*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 7.25 using several randomly generated mazes.

ANS:

```
/* Exercise 7.26 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define DOWN  0 /* move down */
#define RIGHT 1 /* move right */
#define UP    2 /* move up */
#define LEFT  3 /* move left */
#define POSSIBLE_ZEROS 100 /* maximum possible zeroes */

/* function prototypes */
void mazeTraversal( char maze[ 12 ][ 12 ], const int xCoord,
                    const int yCoord, int row, int col, int direction );
void mazeGenerator( char maze[][ 12 ], int *xPtr, int *yPtr );
void printMaze( const char maze[][ 12 ] );
int validMove( const char maze[][ 12 ], int r, int c );
int coordsAreEdge( int x, int y );

int main()
{
   char maze[ 12 ][ 12 ]; /* maze grid */
   int loop;              /* row counter */
   int loop2;             /* column counter */
   int xStart;            /* starting x coordinate */
   int yStart;            /* starting y coordinate */
   int x;                 /* current x coordinate */
   int y;                 /* current y coordinate */

   /* initialize maze grid to 1's */
   for ( loop = 0; loop < 12; loop++ ) {

      for ( loop2 = 0; loop2 < 12; loop2++ ) {
         maze[ loop ][ loop2 ] = '1';
      } /* end for */

   } /* end for */

   /* generate the maze */
   mazeGenerator( maze, &xStart, &yStart );

   x = xStart; /* starting row */
   y = yStart; /* starting col */

   mazeTraversal( maze, xStart, yStart, x, y, RIGHT );

   return 0; /* indicate successful termination */

} /* end main */

/* Assume that there is exactly 1 entrance and
   exactly 1 exit to the maze. */
void mazeTraversal( char maze[ 12 ][ 12 ], const int xCoord,
                    const int yCoord, int row, int col, int direction )
{
   static int flag = 0; /* starting position flag */
```

```
   maze[ row ][ col ] = 'X'; /* insert X at current location */
   printMaze( maze );

   /* if maze completed */
   if ( coordsAreEdge( row, col ) && row != xCoord && col != yCoord ) {
      printf( "\nMaze successfully exited!\n\n" );
      return;
   } /* end if */
   else if ( row == xCoord && col == yCoord && flag == 1 ) {
      printf( "\nArrived back at the starting location.\n\n" );
      return;
   } /* end else if */
   else { /* make next move */
      int move;  /* next move */
      int count; /* counter */

      flag = 1;

      /* loop 4 times and find first valid move */
      for ( move = direction, count = 0; count < 4; ++count,
         ++move, move %= 4 ) {

         /* choose valid move */
         switch( move ) {

            case DOWN: /* move down */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, row + 1, col ) ) {
                  mazeTraversal( maze, xCoord, yCoord, row + 1,
                     col, LEFT );
                  return;
               } /* end if */

               break; /* exit switch */

            case RIGHT: /* move right */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, row, col + 1 ) ) {
                  mazeTraversal( maze, xCoord, yCoord, row,
                     col + 1, DOWN );
                  return;
               } /* end if */

               break; /* exit switch */

            case UP: /* move up */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, row - 1, col ) ) {
                  mazeTraversal( maze, xCoord, yCoord, row - 1,
                     col, RIGHT );
                  return;
               } /* end if */

               break; /* exit switch */

            case LEFT: /* move left */

```

```
                  /* if move is valid, call mazeTraversal */
                  if ( validMove( maze, row, col - 1 ) ) {
                     mazeTraversal( maze, xCoord, yCoord, row,
                        col - 1, UP );
                     return;
                  } /* end if */

                  break; /* exit switch */
            } /* end switch */

         } /* end for */

   } /* end else */

} /* end function mazeTraversal */

/* validate move */
int validMove( const char maze[][ 12 ], int r, int c )
{
   return ( r >= 0 && r <= 11 && c >= 0 && c <= 11 &&
      maze[ r ][ c ] != '1' );

} /* end function validMove */

/* check boundaries of coordinates */
int coordsAreEdge( int x, int y )
{

   /* if coordinates not valid */
   if ( ( x == 0 || x == 11 ) && ( y >= 0 && y <= 11 ) ) {
      return 1;
   } /* end if */
   else if ( ( y == 0 || y == 11 ) && ( x >= 0 && x <= 11 ) ) {
      return 1;
   } /* end else if */
   else { /* coordinates valid */
      return 0;
   } /* end else */

} /* end function coordsAreEdge */

/* print the maze */
void printMaze( const char maze[][ 12 ] )
{
   int x; /* row counter */
   int y; /* column counter */

   /* loop through maze grid */
   for ( x = 0; x < 12; x++ ) {

      for ( y = 0; y < 12; y++ ) {
         printf( "%c ", maze[ x ][ y ] );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\nHit return to see next move" );
   getchar();
} /* end function printMaze */

```

```
/* random maze generator */
void mazeGenerator( char maze[][ 12 ], int *xPtr, int *yPtr )
{
   int a;     /* random number */
   int x;     /* random number */
   int y;     /* random number */
   int entry; /* random entry */
   int exit;  /* random exit */
   int loop;  /* loop counter */

   srand( time( NULL ) );

   /* generate random entry and exit positions */
   do {
      entry = rand() % 4;
      exit = rand() % 4;
   } while ( entry == exit ); /* end do...while */

   /* Determine entry position while avoiding corners */
   if ( entry == 0 ) {
      *xPtr = 1 + rand() % 10;
      *yPtr = 0;
      maze[ *xPtr ][ 0 ] = '0';
   } /* end if */
   else if ( entry == 1 ) {
      *xPtr = 0;
      *yPtr = 1 + rand() % 10;
      maze[ 0 ][ *yPtr ] = '0';
   } /* end else if */
   else if ( entry == 2 ) {
      *xPtr = 1 + rand() % 10;
      *yPtr = 11;
      maze[ *xPtr ][ 11 ] = '0';
   } /* end else if */
   else {
      *xPtr = 11;
      *yPtr = 1 + rand() % 10;
      maze[ 11 ][ *yPtr ] = '0';
   } /* end else */

   /* Determine exit location */
   if ( exit == 0 ) {
      a = 1 + rand() % 10;
      maze[ a ][ 0 ] = '0';
   } /* end if */
   else if ( exit == 1 ) {
      a = 1 + rand() % 10;
      maze[ 0 ][ a ] = '0';
   } /* end else if */
   else if ( exit == 2 ) {
      a = 1 + rand() % 10;
      maze[ a ][ 11 ] = '0';
   } /* end else if */
   else {
      a = 1 + rand() % 10;
      maze[ 11 ][ a ] = '0';
   } /* end else */

   /* randomly add zeroes to maze grid */
   for ( loop = 1; loop < POSSIBLE_ZEROS; loop++ ) {
      x = 1 + rand() % 10;
```

```
      y = 1 + rand() % 10;
      maze[ x ][ y ] = '0';
   } /* end for */

} /* end function mazeGenerator */
```

```
Hit return to see next move
1 1 1 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 0 1 X X X 1
1 1 1 0 0 0 1 1 1 1 X 1
X X 1 0 0 1 X X X X X 1
1 X X X 1 X X 1 X X 1 1
1 X 1 X X X 1 0 1 X 1 1
1 1 X 1 X X X 1 X X 1 1
1 1 X X X X 1 1 X 1 0 1
1 X X 0 0 X 1 X X X 1 1
1 1 X 0 1 X 1 X X 1 0 1
1 1 X X X X X 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1

Hit return to see next move
1 1 1 1 0 1 1 1 1 1 1 1
1 0 1 1 0 1 0 1 X X X 1
1 1 1 0 0 0 1 1 1 1 X 1
X X 1 0 0 1 X X X X X 1
1 X X X 1 X X 1 X X 1 1
1 X 1 X X X 1 0 1 X 1 1
1 1 X 1 X X X 1 X X 1 1
1 1 X X X X 1 1 X 1 0 1
1 X X 0 0 X 1 X X X 1 1
1 1 X 0 1 X 1 X X 1 0 1
1 1 X X X X X 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1

...

Hit return to see next move
1 1 1 1 X 1 1 1 1 1 1 1
1 0 1 1 X 1 0 1 X X X 1
1 1 1 0 X X 1 1 1 1 X 1
X X 1 X X 1 X X X X X 1
1 X X X 1 X X 1 X X 1 1
1 X 1 X X X 1 0 1 X 1 1
1 1 X 1 X X X 1 X X 1 1
1 1 X X X X 1 1 X 1 0 1
1 X X 0 0 X 1 X X X 1 1
1 1 X 0 1 X 1 X X 1 0 1
1 1 X X X X X 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1

Hit return to see next move

Maze successfully exited!
```

7.27 (*Mazes of Any Size*) Generalize functions `mazeTraverse` and `mazeGenerator` of Exercise 7.25 and Exercise 7.26 to process mazes of any width and height.

ANS:

```
/* Exercise 7.27 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ROW 10  /* height */
#define COL 10  /* width  */
#define DOWN  0 /* move down */
#define RIGHT 1 /* move right */
#define UP    2 /* move up */
#define LEFT  3 /* move left */

/* function prototypes */
void mazeTraversal( char maze[ ROW ][ COL ], const int xCoord,
                    const int yCoord, int row, int col, int direction );
void mazeGenerator( char maze[][ COL ], int *xPtr, int *yPtr );
void printMaze( const char maze[][ COL ] );
int validMove( const char maze[][ COL ], int r, int c );
int coordsAreEdge( int x, int y );

int main()
{
   char maze[ ROW ][ COL ]; /* maze grid */
   int loop;                /* row counter */
   int loop2;               /* column counter */
   int xStart;              /* starting x coordinate */
   int yStart;              /* starting y coordinate */
   int x;                   /* current x coordinate */
   int y;                   /* current y coordinate */

   /* initialize maze grid to 1's */
   for ( loop = 0; loop < ROW; loop++ ) {

      for ( loop2 = 0; loop2 < COL; loop2++ ) {
         maze[ loop ][ loop2 ] = '1';
      } /* end for */

   } /* end for */

   /* generate the maze */
   mazeGenerator( maze, &xStart, &yStart );

   x = xStart; /* starting row */
   y = yStart; /* starting col */

   mazeTraversal( maze, xStart, yStart, x, y, RIGHT );

   return 0; /* indicate successful termination */

} /* end main */

/* Assume that there is exactly 1 entrance and
   exactly 1 exit to the maze. */
void mazeTraversal( char maze[ ROW ][ COL ], const int xCoord,
                    const int yCoord, int row, int col, int direction )
{
   static int flag = 0; /* starting position flag */
```

```
   maze[ row ][ col ] = 'X'; /* insert X at current location */
   printMaze( maze );

   /* if maze completed */
   if ( coordsAreEdge( row, col ) && row != xCoord && col != yCoord ) {
      printf( "\nMaze successfully exited!\n\n" );
      return;
   } /* end if */
   else if ( row == xCoord && col == yCoord && flag == 1 ) {
      printf( "\nArrived back at the starting location.\n\n" );
      return;
   } /* end else if */
   else { /* make next move */
      int move;  /* next move */
      int count; /* counter */

      flag = 1;

      /* loop 4 times and find first valid move */
      for ( move = direction, count = 0; count < 4; ++count,
         ++move, move %= 4 ) {

         /* choose valid move */
         switch( move ) {

            case DOWN: /* move down */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, row + 1, col ) ) {
                  mazeTraversal( maze, xCoord, yCoord, row + 1,
                     col, LEFT );
                  return;
               } /* end if */

               break; /* exit switch */

            case RIGHT: /* move right */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, row, col + 1 ) ) {
                  mazeTraversal( maze, xCoord, yCoord, row,
                     col + 1, DOWN );
                  return;
               } /* end if */

               break; /* exit switch */

            case UP: /* move up */

               /* if move is valid, call mazeTraversal */
               if ( validMove( maze, row - 1, col ) ) {
                  mazeTraversal( maze, xCoord, yCoord, row - 1,
                     col, RIGHT );
                  return;
               } /* end if */

               break; /* exit switch */

            case LEFT: /* move left */

```

```
                  /* if move is valid, call mazeTraversal */
                  if ( validMove( maze, row, col - 1 ) ) {
                     mazeTraversal( maze, xCoord, yCoord, row,
                        col - 1, UP );
                     return;
                  } /* end if */

                  break; /* exit switch */
            } /* end switch */

         } /* end for */

      } /* end else */

} /* end function mazeTraversal */

/* validate move */
int validMove( const char maze[][ COL ], int r, int c )
{
   return ( r >= 0 && r <= ROW - 1 && c >= 0 && c <= COL - 1 &&
      maze[ r ][ c ] != '1' );  /* a valid move */

} /* end function validMove */

/* check boundaries of coordinates */
int coordsAreEdge( int x, int y )
{

   /* if coordinates not valid */
   if ( ( x == 0 || x == ROW - 1 ) && ( y >= 0 && y <= COL - 1 ) ) {
      return 1;
   } /* end if */
   else if ( ( y == 0 || y == COL - 1 ) && ( x >= 0 &&
      x <= ROW - 1 ) ) {
      return 1;
   } /* end else if */
   else { /* coordinates valid */
      return 0;
   } /* end else */

} /* end function coordsAreEdge */

/* print the maze */
void printMaze( const char maze[][ COL ] )
{
   int x; /* row counter */
   int y; /* column counter */

   /* loop through maze grid */
   for ( x = 0; x < ROW; x++ ) {

      for ( y = 0; y < COL; y++ ) {
         printf( "%c ", maze[ x ][ y ] );
      } /* end for */

      printf( "\n" );
   } /* end for */

   printf( "\nHit return to see next move" );
   getchar();
} /* end function printMaze */
```

```c

/* random maze generator */
void mazeGenerator( char maze[][ COL ], int *xPtr, int *yPtr )
{
   int a;     /* random number */
   int x;     /* random number */
   int y;     /* random number */
   int entry; /* random entry */
   int exit;  /* random exit */
   int loop;  /* loop counter */

   srand( time( NULL ) );

   /* generate random entry and exit positions */
   do {
      entry = rand() % 4;
      exit = rand() % 4;
   } while ( entry == exit ); /* end do...while */

   /* Determine entry position while avoiding corners */
   if ( entry == 0 ) {
      *xPtr = 1 + rand() % ( ROW - 2 );
      *yPtr = 0;
      maze[ *xPtr ][ *yPtr ] = '0';
   } /* end if */
   else if ( entry == 1 ) {
      *xPtr = 0;
      *yPtr = 1 + rand() % ( COL - 2 );
      maze[ *xPtr ][ *yPtr ] = '0';
   } /* end else if */
   else if ( entry == 2 ) {
      *xPtr = 1 + rand() % ( ROW - 2 );
      *yPtr = COL - 1;
      maze[ *xPtr ][ *yPtr ] = '0';
   } /* end else if */
   else {
      *xPtr = ROW - 1;
      *yPtr = 1 + rand() % ( COL - 2 );
      maze[ *xPtr ][ *yPtr ] = '0';
   } /* end else */

   /* Determine exit location */
   if ( exit == 0 ) {
      a = 1 + rand() % ( ROW - 2 );
      maze[ a ][ 0 ] = '0';
   } /* end if */
   else if ( exit == 1 ) {
      a = 1 + rand() % ( COL - 2 );
      maze[ 0 ][ a ] = '0';
   } /* end else if */
   else if ( exit == 2 ) {
      a = 1 + rand() % ( ROW - 2 );
      maze[ a ][ COL - 1 ] = '0';
   } /* end else if */
   else {
      a = 1 + rand() % ( COL - 2 );
      maze[ ROW - 1 ][ a ] = '0';
   } /* end else */

   /* randomly add zeroes to maze grid */
   for ( loop = 1; loop < ( ROW - 2 ) * ( COL - 2 ); loop++ ) {
```

```
        x = 1 + rand() % ( ROW - 2 );
        y = 1 + rand() % ( COL - 2 );
        maze[ x ][ y ] = '0';
     } /* end for */

  } /* end function mazeGenerator */
```

```
1 1 X 1 1 1 1 1 1 1
1 0 0 1 1 0 1 0 0 1
0 0 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1
1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 1 0 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1

Hit return to see next move
1 1 X 1 1 1 1 1 1 1
1 0 X 1 1 0 1 0 0 1
0 0 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1
1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 1 0 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1

...

Hit return to see next move
1 1 X 1 1 1 1 1 1 1
1 X X 1 1 0 1 0 0 1
X X 0 1 1 1 1 1 0 1
1 0 0 0 0 0 1 0 1 1
1 1 1 0 0 0 1 0 0 1
1 1 1 0 1 0 1 1 0 1
1 0 0 0 1 0 0 1 0 1
1 0 0 0 1 1 0 0 0 1
1 0 0 0 0 1 1 0 0 1
1 1 1 1 1 1 1 1 1 1

Hit return to see next move

Maze successfully exited!
```

7.28 (*Arrays of Pointers to Functions*) Rewrite the program of Fig. 6.22 to use a menu driven interface. The program should offer the user four options as follows:

```
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. 6.22 must be modified so that 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. 6.22 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.

ANS:

```c
/* Exercise 7.28 Solution */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

/* function prototypes */
void minimum( int grades[][ EXAMS ], int pupils, int tests );
void maximum( int grades[][ EXAMS ], int pupils, int tests );
void average( int grades[][ EXAMS ], int pupils, int tests );
void printArray( int grades[][ EXAMS ], int pupils, int tests );
void printMenu( void );

int main()
{

   /* pointer to a function that takes as parameters a
      two-dimensional array and two integer values */
   void ( *processGrades[ 4 ] )( int [][ EXAMS ], int, int )
                       = { printArray, minimum, maximum, average};

   int choice = 0; /* menu choice */

   /* array of student grades */
   int studentGrades[ STUDENTS ][ EXAMS ] = { { 77, 68, 86, 73 },
                                              { 96, 87, 89, 78 },
                                              { 70, 90, 86, 81 } };

   /* loop while user does not choose option 4 */
   while ( choice != 4 ) {

      /* display menu and read user's choice */
      do {
         printMenu();
         scanf( "%d", &choice );
      } while ( choice < 0 || choice > 4 ); /* end do...while */

      /* pass choice into the array */
      if ( choice != 4 ) {
         ( *processGrades[ choice ] )( studentGrades, STUDENTS, EXAMS );
      } /* end if */
```

```
      else {
         printf( "Program Ended.\n" );
      } /* end else */

   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */

/* search for the minimum value */
void minimum( int grades[][ EXAMS ], int pupils, int tests )
{
   int i;               /* loop counter */
   int j;               /* loop counter */
   int lowGrade = 100; /* set lowGrade to highest possible score */

   /* loop through rows */
   for ( i = 0; i <= pupils - 1; i++ ) {

      /* loop through columns */
      for ( j = 0; j <= tests - 1; j++ ) {

         /* if current grade is lower than lowGrade */
         if ( grades[ i ][ j ] < lowGrade ) {
            lowGrade = grades[ i ][ j ];
         } /* end if */

      } /* end for */

   } /* end for */

   printf( "\n\tThe lowest grade is %d\n", lowGrade );
} /* end function minimum */

/* search for maximum value */
void maximum( int grades[][ EXAMS ], int pupils, int tests )
{
   int i;               /* loop counter */
   int j;               /* loop counter */
   int highGrade = 0;  /* set highGrade to lowest possible score */

   /* loop through rows */
   for ( i = 0; i <= pupils - 1; i++ ) {

      /* loop through columns */
      for ( j = 0; j <= tests - 1; j++ ) {

         /* if current grade is higher than highGrade */
         if ( grades[ i ][ j ] > highGrade ) {
            highGrade = grades[ i ][ j ];
         } /* end if */

      } /* end for */

   } /* end for */

   printf( "\n\tThe highest grade is %d\n", highGrade );
} /* end function maximum */

```

```c
/* calculate average */
void average( int grades[][ EXAMS ], int pupils, int tests )
{
   int i;     /* loop counter */
   int j;     /* loop counter */
   int total; /* sum of all grades */

   printf( "\n" );

   /* loop through rows */
   for ( i = 0; i <= pupils - 1; i++ ) {
      total = 0;

      /* loop through columns */
      for ( j = 0; j <= tests - 1; j++ ) {
         total += grades[ i ][ j ];
      } /* end for */

      printf( "\tThe average for student %d is %.1f\n",
             i + 1, ( double ) total / tests );
   } /* end for */

} /* end function average */

/* print the contents of the array */
void printArray( int grades[][ EXAMS ], int pupils, int tests )
{
   int i; /* loop counter */
   int j; /* loop counter */

   printf( "\n\t                 [ 0 ]  [ 1 ]  [ 2 ]  [ 3 ]" );

   /* loop through rows */
   for ( i = 0; i <= pupils - 1; i++ ) {
      printf( "\n\tstudentGrades[ %d ] ", i );

      /* loop through columns */
      for ( j = 0; j <= tests - 1; j++ ) {
         printf( "%-7d", grades[ i ][ j ] );
      } /* end for */

   } /* end for */

   printf( "\n" );
} /* end function printArray */

/* display the menu */
void printMenu( void )
{
   printf( "\n\tEnter a choice:\n"
          "\t  0  Print the array of grades\n"
          "\t  1  Find the minimum grade\n"
          "\t  2  Find the maximum grade\n"
          "\t  3  Print the average on all"
          " tests for each student\n"
          "\t  4  End program\n"
          "\t? " );
} /* end function printMenu */
```

```
        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 1 is 76.0
        The average for student 2 is 87.5
        The average for student 3 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.
```

7.29 (*Modifications to the Simpletron Simulator*) In Exercise 7.19, 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 12.26 and 12.27, we propose building a compiler that converts programs written in a high-level programming language (a variation of BASIC) to Simpletron Machine Language. Some of the following modifications and enhancements may be required to execute the programs produced by the compiler.

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 remainder 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 prints a string beginning at a specified Simpletron memory location. The first half of the word at that location is the length of the string in characters. 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.]

7.30 What does this program do?

```c
/* ex07_30.c */
/* What does this program do? */
#include <stdio.h>

int mystery3( const char *s1, const char *s2 ); /* prototype */

int main()
{
   char string1[ 80 ]; /* create char array */
   char string2[ 80 ]; /* create char array */

   printf( "Enter two strings: " );
   scanf( "%s%s", string1 , string2 );

   printf( "The result is %d\n", mystery3( string1, string2 ) );

   return 0; /* indicates successful termination */

} /* end main */

int mystery3( const char *s1, const char *s2 )
{
   for ( ; *s1 != '\0' && *s2 != '\0'; s1++, s2++ ) {

      if ( *s1 != *s2 ) {
         return 0;
      } /* end if */

   } /* end for */

   return 1;

} /* end function mystery3 */
```

ANS: The Program compares two strings, element by element, for equality.

```
Enter two strings:  string1 string2
The result is 0
```

```
Enter two strings:  string2 string2
The result is 1
```

8

C Characters and Strings: Solutions

SOLUTIONS

8.5 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.

ANS:

```
/* Exercise 8.5 Solution */
#include <stdio.h>
#include <ctype.h>

int main()
{
   int c; /* character input by user */

   printf( "Enter a character: " );
   c = getchar();

   /* test each function of the character handling library */
   printf( "isdigit( \'%c\' )  = %d\n", c, isdigit( c ) );
   printf( "isalpha( \'%c\' )  = %d\n", c, isalpha( c ) );
   printf( "isalnum( \'%c\' )  = %d\n", c, isalnum( c ) );
   printf( "isxdigit( \'%c\' ) = %d\n", c, isxdigit( c ) );
   printf( "islower( \'%c\' )  = %d\n", c, islower( c ) );
   printf( "isupper( \'%c\' )  = %d\n", c, isupper( c ) );
   printf( "tolower( \'%c\' )  = %d\n", c, tolower( c ) );
   printf( "toupper( \'%c\' )  = %d\n", c, toupper( c ) );
   printf( "isspace( \'%c\' )  = %d\n", c, isspace( c ) );
   printf( "iscntrl( \'%c\' )  = %d\n", c, iscntrl( c ) );
   printf( "ispunct( \'%c\' )  = %d\n", c, ispunct( c ) );
   printf( "isprint( \'%c\' )  = %d\n", c, isprint( c ) );
   printf( "isgraph( \'%c\' )  = %d\n", c, isgraph( c ) );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a character: h
isdigit( 'h' )  = 0
isalpha( 'h' )  = 2
isalnum( 'h' )  = 2
isxdigit( 'h' ) = 0
islower( 'h' )  = 2
isupper( 'h' )  = 0
tolower( 'h' )  = 104
toupper( 'h' )  = 72
isspace( 'h' )  = 0
iscntrl( 'h' )  = 0
ispunct( 'h' )  = 0
isprint( 'h' )  = 2
isgraph( 'h' )  = 2
```

8.6 Write a program that inputs a line of text with function `gets` into `char` array `s[ 100 ]`. Output the line in uppercase letters and in lowercase letters.

ANS:

```
/* Exercise 8.6 Solution */
#include <stdio.h>
#include <ctype.h>

int main()
{
   char s[ 100 ]; /* define character array of size 100 */
   int i; /* loop counter */

   /* use gets to get text from user */
   printf( "Enter a line of text:\n" );
   gets( s );
   printf( "\nThe line in uppercase is:\n" );

   /* convert each character to uppercase and output */
   for ( i = 0; s[ i ] != '\0'; i++ ) {
      printf( "%c", toupper( s[ i ] ) );
   } /* end for */

   printf( "\n\nThe line in lowercase is:\n" );

   /* convert each character to lowercase and output */
   for ( i = 0; s[ i ] != '\0'; i++ ) {
      printf( "%c", tolower( s[ i ] ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a line of text:
A line with UPPER- and lowercase LeTters

The line in uppercase is:
A LINE WITH UPPER- AND LOWERCASE LETTERS

The line in lowercase is:
a line with upper- and lowercase letters
```

8.7 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.

ANS:

```
/* Exercise 8.7 Solution */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char stringValue[ 6 ]; /* integer string input by user */
   int sum = 0;           /* result of four integers */
   int i;                 /* loop counter */

   /* loop 4 times */
   for ( i = 1; i <= 4; i++ ) {
      printf( "Enter an integer string: " );
      scanf( "%s", stringValue );

      /* atoi converts stringValue to integer */
      sum += atoi( stringValue );
   } /* end for */

   printf( "\nThe total of the values is %d\n", sum );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter an integer string: 43
Enter an integer string: 77
Enter an integer string: 120
Enter an integer string: 9999

The total of the values is 10239
```

8.8 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.

ANS:

```
/* Exercise 8.8 Solution */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char stringValue[ 15 ]; /* string input by user */
   double sum = 0.0;       /* sum of all four values */
   int i;                  /* loop counter */

   /* loop 4 times */
   for ( i = 1; i <= 4; i++ ) {
      printf( "Enter a doubleing point string: " );
      gets( stringValue );

      /* atof converts stringValue to a floating-point value */
      sum += atof( stringValue );
   } /* end for */

   printf( "\nThe total of the values is %f\n", sum );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a doubleing point string: 1.2
Enter a doubleing point string: 2.3
Enter a doubleing point string: 3.4
Enter a doubleing point string: 4.5

The total of the values is 11.400000
```

8.9 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.

ANS:

```c
/* Exercise 8.9 Solution */
#include <stdio.h>
#include <string.h>

int main()
{
   char string1[ 20 ]; /* first string input by user */
   char string2[ 20 ]; /* second string input by user */
   int result;         /* result of comparing two strings */

   printf( "Enter two strings: " );
   scanf( "%s%s", string1, string2 ); /* read two strings */

   result = strcmp( string1, string2 );

   /* display appropriate message for result */
   if ( result > 0 ) {
      printf( "\"%s\" is greater than \"%s\"\n", string1, string2 );
   } /* end if */
   else if ( result == 0 ) {
      printf( "\"%s\" is equal to \"%s\"\n", string1, string2 );
   } /* end else if */
   else {
      printf( "\"%s\" is less than \"%s\"\n", string1, string2 );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter two strings: Greg Dave
"Greg" is greater than "Dave"
```

```
Enter two strings: Bill Bill
"Bill" is equal to "Bill"
```

```
Enter two strings: Pete Tim
"Pete" is less than "Tim"
```

8.10 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 second string.

ANS:

```
/* Exercise 8.10 Solution */
#include <stdio.h>
#include <string.h>

int main()
{
   char string1[ 20 ]; /* first string input by user */
   char string2[ 20 ]; /* second string input by user */
   int result;         /* result of using strncmp */
   int compareCount;   /* how many characters to be compared */

   /* get two strings from user */
   printf( "Enter two strings: " );
   scanf( "%s%s", string1, string2 );

   /* get number of characters to compare */
   printf( "How many characters should be compared: " );
   scanf( "%d", &compareCount );

   result = strncmp( string1, string2, compareCount );

   /* display appropriate message for result */
   if ( result > 0 ) {
      printf( "\"%s\" is greater than \"%s\" up to %d characters\n",
              string1, string2, compareCount );
   } /* end if */
   else if ( result == 0 ) {
      printf( "\"%s\" is equal to \"%s\" up to %d characters\n",
              string1, string2, compareCount );
   } /* end else if */
   else {
      printf( "\"%s\" is less than \"%s\" up to %d characters\n",
              string1, string2, compareCount );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter two strings: ABCDEFG ABCDEFH
How many characters should be compared: 6
"ABCDEFG" is less than "ABCDEFH" up to 6 characters
```

```
Enter two strings: ABCDEFG ABCDEFH
How many characters should be compared: 7
"ABCDEFG" is less than "ABCDEFH" up to 7 characters
```

```
Enter two strings: ABCEFG ABCDFG
How many characters should be compared: 4
"ABCEFG" is greater than "ABCDFG" up to 4 characters
```

8.11 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 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?)

ANS:

```c
/* Exercise 8.11 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <ctype.h>

int main()
{
   /* initialize 4 arrays of char pointers */
   char *article[] = { "the", "a", "one", "some", "any" };
   char *noun[] = { "boy", "girl", "dog", "town", "car" };
   char *verb[] = { "drove", "jumped", "ran", "walked", "skipped" };
   char *preposition[] = { "to", "from", "over", "under", "on" };
   char sentence[ 100 ] = ""; /* completed sentence */
   int i; /* loop counter */

   /* create 20 sentences */
   for ( i = 1; i <= 20; i++ ) {

      /* randomly choose pieces of sentence */
      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 ] );

      /* capitalize first letter and print sentence */
      putchar( toupper( sentence[ 0 ] ) );
      printf( "%s.\n", &sentence[ 1 ] );
      sentence[ 0 ] = '\0';
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
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.
```

8.12 *(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 8.11, write a program that produces random limericks. Polishing this program to produce good limericks is a challenging problem, but the result will be worth the effort!

8.13 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 the following: 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.]

ANS:

```c
/* Exercise 8.13 Solution */
#include <stdio.h>
#include <string.h>

void printLatinWord( char *word ); /* function prototype */

int main()
{
   char sentence[ 80 ]; /* sentence input by user */
   char *tokenPtr;      /* pointer to current token */

   printf( "Enter a sentence:\n" );
   gets( sentence );
   printf( "\nThe sentence in Pig Latin is:\n" );

   /* call function strtok to alter the sentence */
   tokenPtr = strtok( sentence, " .,;" );

   /* if tokenPtr does not equal NULL */
   while ( tokenPtr ) {

      /* pass the token to printLatinWord and get next token */
      printLatinWord( tokenPtr );
      tokenPtr = strtok( NULL, " .,;" );

      /* if tokenPtr not NULL, print space */
      if ( tokenPtr ) {
         printf( " " );
      } /* end if */

   } /* end while */

   printf( "." );

   return 0; /* indicates successful termination */

} /* end main */

/* print out the English word in pig Latin form */
void printLatinWord( char *word )
{
   unsigned int i; /* loop counter */

   /* loop through the word */
   for ( i = 1; i < strlen( word ); i++ ) {
      printf( "%c", word[ i ] );
   } /* end for */

   printf( "%c%s", word[ 0 ], "ay" );
} /* end function printLatinWord */
```

```
Enter a sentence:
characters and strings

The sentence in Pig Latin is:
haracterscay ndaay tringssay.
```

8.14 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.

ANS:

```
/* Exercise 8.14 Solution */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
   char p[ 20 ];                       /* complete phone number */
   char phoneNumber[ 10 ] = { '\0' }; /* long integer phone number */
   char *tokenPtr;                     /* store temporary token */
   int  areaCode;                      /* store area code */
   long phone;                         /* store phone number */

   printf( "Enter a phone number in the form ( 555 )"
          " 555-5555:\n" );
   gets( p );

   /* convert area code token to an integer */
   areaCode = atoi( strtok( p, "()" ) );

   /* take next token and copy to phoneNumber */
   tokenPtr = strtok( NULL, " -" );
   strcpy( phoneNumber, tokenPtr );

   /* take last token and concatenate to phoneNumber */
   tokenPtr = strtok( NULL, "" );
   strcat( phoneNumber, tokenPtr );

   /* convert phoneNumber to long integer */
   phone = atol( phoneNumber );

   printf( "\nThe integer area code is %d\n", areaCode );
   printf( "The long integer phone number is %ld\n", phone );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a phone number in the form ( 555 ) 555-5555:
(800) 555-1212

The integer area code is 800
The long integer phone number is 5551212
```

8.15 Write a program that inputs a line of text, tokenizes the line with function `strtok` and outputs the tokens in reverse order..
ANS:

```c
/* Exercise 8.15 solution */
#include <stdio.h>
#include <string.h>

void reverseTokens( char *sentence ); /* function prototype */

int main()
{
   char text[ 80 ]; /* line of text from user */

   printf( "Enter a line of text:\n" );
   gets( text );

   reverseTokens( text ); /* call to function reverseTokens */

   return 0; /* indicate successful termination */

} /* end main */

/* function to reverse the individual tokens */
void reverseTokens( char *sentence )
{
   char *pointers[ 50 ]; /* array to store entire sentence */
   char *temp;           /* pointer to each token */
   int count = 0;        /* token counter */
   int i;                /* loop counter */

   /* function strtok takes first word of sentence */
   temp = strtok( sentence, " " );

   /* while temp does not equal NULL */
   while ( temp ) {

      /* add the word into the array and get next token */
      pointers[ count++ ] = temp;
      temp = strtok( NULL, " " );
   } /* end while */

   printf( "The tokens in reverse order are:\n" );

   /* loop through the array backwards */
   for ( i = count - 1; i >= 0; i-- ) {
      printf( "%s ", pointers[ i ] );
   } /* end for */

} /* end function reverseTokens */
```

```
Enter a line of text:
testing 1 2 3
The tokens in reverse order are:
3 2 1 testing
```

8.16 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 `searchPtr + 1` as its first argument.]

ANS:

```
/* Exercise 8.16 Solution */
#include <stdio.h>
#include <string.h>
int main()
{
   char text[ 80 ];   /* line of text */
   char search[ 15 ]; /* search string */
   char *searchPtr;   /* poiner to search string */

   /* get line of text from user */
   printf( "Enter a line of text:\n" );
   gets( text );

   /* get search string from user */
   printf( "Enter a search string: " );
   scanf( "%s", search );

   /* search for search string in text */
   searchPtr = strstr( text, search );

   /* if searchPtr is not NULL */
   if ( searchPtr ) {
      printf( "\n%s\n%s\"%s\":\n%s\n",
             "The remainder of the line beginning with",
             "the first occurrence of ", search, searchPtr );

      /* search for a second occurrence */
      searchPtr = strstr( searchPtr + 1, search );

      /* if searchPtr is not NULL */
      if ( searchPtr )  {
         printf( "\n%s\n%s\"%s\":\n%s\n",
                "The remainder of the line beginning with",
                "the second occurrence of ", search, searchPtr );
      } /* end if */
      else {
         printf( "The search string appeared only once.\n" );
      } /* end else */

   } /* end if */
   else {
      printf( "\"%s\" not found.\n", search );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a line of text:
To be or not to be; that is the question.
Enter a search string: be

The remainder of the line beginning with
the first occurrence of "be":
be or not to be; that is the question.

The remainder of the line beginning with
the second occurrence of "be":
be; that is the question.
```

8.17 Write a program based on the program of Exercise 8.16 that inputs several lines of text and a search string, and uses function `strstr` to determine the total occurrences of the string in the lines of text. Print the result.

ANS:

```
/* Exercise 8.17 Solution */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main()
{
   char text[ 3 ][ 80 ]; /* array to hold text entered by user */
   char search[ 20 ];    /* search string */
   char *searchPtr;      /* pointer to search string */
   int count = 0;        /* total occurrences of search string */
   int i;                /* loop counter */
   int j;                /* loop counter */

   printf( "Enter three lines of text:\n" );

   /* read in 3 lines of text */
   for ( i = 0; i <= 2; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* make all characters lowercase */
   for ( i = 0; i <= 2; i++ ) {

      /* loop through each character */
      for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
         text[ i ][ j ] = tolower( text[ i ][ j ] );
      } /* end for */

   } /* end for */

   printf( "\nEnter a search string: " ); /* get search string */
   scanf( "%s", search );

   /* loop through all three strings */
   for ( i = 0; i <= 2; i++ ) {

      /* set pointer to first character of string */
      searchPtr = &text[ i ][ 0 ];

      /* loop while strstr does not return NULL */
      while ( searchPtr = strstr( searchPtr, search ) ) {
         ++count;
         searchPtr++;
      } /* end while */

   } /* end for */

```

```
   printf( "\nThe total occurrences of \"%s\" in the text is %d\n",
           search, count );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three lines of text:
This program inputs three lines of text
and counts the number of occurrences of
the search string in the three lines of text.

Enter a search string: th

The total occurrences of "th" in the text is 6
```

8.18 Write a program that inputs several lines of text and a search character, and uses function `strchr` to determine the total occurrences of the character in the lines of text.

ANS:

```
/* Exercise 8.18 Solution */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main()
{
   char text[ 3 ][ 80 ]; /* array to hold text entered by user */
   char search;          /* search character */
   char *searchPtr;      /* pointer to search character */
   int count = 0;        /* total search characters found */
   int i;                /* loop counter */
   int j;                /* loop counter */

   printf( "Enter three lines of text:\n" );

   /* read 3 lines of text */
   for ( i = 0; i <= 2; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* convert all letters to lowercase */
   for ( i = 0; i <= 2; i++ ) {

      /* loop through each character */
      for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
         text[ i ][ j ] = tolower( text[ i ][ j ] );
      } /* end for */

   } /* end for */

   /* get search character */
   printf( "\nEnter a search character: " );
   scanf( "%c", &search );

   /* loop through 3 lines of text */
   for ( i = 0; i <= 2; i++ ) {

      /* set pointer to first character in line */
      searchPtr = &text[ i ][ 0 ];

      /* loop while strchr does not return NULL */
      while ( searchPtr = strchr( searchPtr, search ) ) {
         ++count;
```

```
            searchPtr++;
         } /* end while */

      } /* end for */

      printf( "\nThe total occurrences of '%c' in the text is %d\n",
              search, count );

      return 0; /* indicate successful termination */

   } /* end main */
```

```
Enter three lines of text:
This program inputs three lines of text
and counts the number of occurrences of
the specified search character in the text

Enter a search character: e

The total occurrences of 'e' in the text is 15
```

8.19 Write a program based on the program of Exercise 8.18 that inputs several lines of text and uses function `strchr` to determine the total occurrences of each letter of the alphabet in the lines of 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.

ANS:

```
/* Exercise 8.19 Solution */
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int main()
{
   char text[ 3 ][ 80 ];           /* 3 lines of text */
   char *searchPtr;                /* pointer to search character */
   char characters[ 26 ] = { 0 }; /* totals for each letter */
   int count = 0;                  /* total for current letter */
   int i;                          /* loop counter */
   int j;                          /* loop counter */

   printf( "Enter three lines of text:\n" );

   /* read three lines of text */
   for ( i = 0; i <= 2; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* convert letters to lowercase */
   for ( i = 0; i <= 2; i++ ) {

      /* loop through each character of line */
      for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {
         text[ i ][ j ] = tolower( text[ i ][ j ] );
      } /* end for */

   } /* end for */

   /* loop through alphabet */
   for ( i = 0; i <= 25; i++ ) {

      /* loop through 3 lines of text */
      for ( j = 0, count = 0; j <= 2; j++ ) {
         searchPtr = &text[ j ][ 0 ];

         /* while strchr does not return NULL */
         while ( searchPtr = strchr( searchPtr, 'a' + i ) ) {
            ++count;
            searchPtr++;
         } /* end while */

      } /* end for */

      characters[ i ] = count;
   } /* end for */

   printf( "\nThe total occurrences of each character:\n" );

   /* display totals for each character */
   for ( i = 0; i <= 25; i++ ) {
      printf( "%c:%3d\n", 'a' + i, characters[ i ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three lines of text:
This program inputs three lines of text
and determines the number of occurrences
of each character in the three lines.

The total occurrences of each character:
a:  5
b:  1
c:  6
d:  2
e: 17
f:  3
g:  1
h:  7
i:  6
j:  0
k:  0
l:  2
m:  3
n:  8
o:  5
p:  2
q:  0
r: 10
s:  6
t: 10
u:  3
v:  0
w:  0
x:  1
y:  0
z:  0
```

8.20 Write a program that inputs several lines of text and uses `strtok` to count the total number of words. Assume that the words are separated either by spaces or newline characters.

ANS:

```c
/* Exercise 8.20 Solution */
#include <stdio.h>
#include <string.h>

int main()
{
   char text[ 4 ][ 80 ]; /* text entered by user */
   char *tokenPtr;       /* pointer to current token */
   int i;                /* loop counter */
   int counter = 0;      /* token counter */

   printf( "Enter 4 lines of text: \n" );

   /* read 4 lines of text */
   for ( i = 0; i <= 3; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* loop through 4 lines of text */
   for ( i = 0; i <= 3; i++ ) {

      /* get first token */
      tokenPtr = strtok( &text[ i ][ 0 ], " \n" );

      /* while tokenPtr does not equal NULL */
      while ( tokenPtr ) {
         ++counter;
         tokenPtr = strtok( NULL, " \n" ); /* get next token */
      } /* end while */

   } /* end for */

   printf( "\nThe total number of words is %d\n", counter );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter 4 lines of text:
This line of text has seven words
This line has five words
There are two words on the next line
I am

The total number of words is 22
```

8.21 Use the string comparison functions discussed in Section 8.6 and the techniques for sorting arrays developed in Chapter 6 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.

ANS:

```c
/* Exercise 8.21 solution */
#include <stdio.h>
#include <string.h>

void bubbleSort( char a[][ 50 ] ); /* function prototype */

int main()
{
   char array[ 10 ][ 50 ]; /* 10 lines of text from user */
   int i; /* counter */

   /* read in 10 lines of text */
   for ( i = 0; i <= 9; i++ ) {
      printf( "Enter a string: " );
      scanf( "%s", &array[ i ][ 0 ] );
   } /* end for */

   bubbleSort( array ); /* sort the array of strings */
   printf( "\nThe strings in sorted order are:\n" );

   /* display text in sorted order */
   for ( i = 0; i <= 9; i++ ) {
      printf( "%s\n", &array[ i ][ 0 ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */

/* sort the array */
void bubbleSort( char a[][ 50 ] )
{
   int i;           /* loop counter */
   int j;           /* loop counter */
   char temp[ 50 ]; /* temporary array */

   /* make 9 passes */
   for ( i = 0; i <= 8; i++ ) {

      for ( j = 0; j <= 8; j++ ) {

         /* swap strings if necessary */
         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 );
         } /* end if */

      } /* end for */

   } /* end for */

} /* end function bubbleSort */
```

```
Enter a string: Westborough
Enter a string: Wellesley
Enter a string: Natick
Enter a string: Waltham
Enter a string: Framingham
Enter a string: Marlborough
Enter a string: Boston
Enter a string: Ashland
Enter a string: Hopkington
Enter a string: Shrewsbury

The strings in sorted order are:
Ashland
Boston
Framingham
Hopkington
Marlborough
Natick
Shrewsbury
Waltham
Wellesley
Westborough
```

8.22 The chart in Appendix D 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."

ANS: True.

b) The digit "9" comes before the digit "0."

ANS: False.

c) The commonly used symbols for addition, subtraction, multiplication and division all come before any of the digits.

ANS: True.

d) The digits come before the letters.

ANS: True.

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.

ANS: False.

8.23 Write a program that reads a series of strings and prints only those strings beginning with the letter “b.”
ANS:

```c
/* Exercise 8.23 solution */
#include <stdio.h>

int main()
{
   int i; /* loop counter */
   char array[ 5 ][ 20 ]; /* 5 strings from user */

   /* read 5 strings from user */
   for ( i = 0; i <= 4; i++ ) {
      printf( "Enter a string: " );
      scanf( "%s", &array[ i ][ 0 ] );
   } /* end for */

   printf( "\nThe strings starting with 'b' are:\n" );

   /* loop through strings */
   for ( i = 0; i <= 4; i++ ) {

      /* print if first character is 'b' */
      if ( array[ i ][ 0 ] == 'b' ) {
         printf( "%s\n", &array[ i ][ 0 ] );
      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a string: the
Enter a string: big
Enter a string: bad
Enter a string: boy
Enter a string: sings

The strings starting with 'b' are:
big
bad
boy
```

8.24 Write a program that reads a series of strings and prints only those strings that end with the letters "ed."
ANS:

```c
/* Exercise 8.24 solution */
#include <stdio.h>
#include <string.h>

int main()
{
   int i;                   /* loop counter */
   int length;              /* length of current string */
   char array[ 5 ][ 20 ]; /* 5 strings from user */

   /* read in 5 strings from user */
   for ( i = 0; i <= 4; i++ ) {
      printf( "Enter a string: " );
      scanf( "%s", &array[ i ][ 0 ] );
   } /* end for */

   printf( "\nThe strings ending with \"ED\" are:\n" );

   /* loop through 5 strings */
   for ( i = 0; i <= 4; i++ ) {

      /* find length of current string */
      length = strlen( &array[ i ][ 0 ] );

      /* print string if it ends with "ED" */
      if ( strcmp( &array[ i ][ length - 2 ], "ED" ) == 0 ) {
         printf( "%s\n", &array[ i ][ 0 ] );
      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a string: WALKED
Enter a string: SKIPPED
Enter a string: JUMPED
Enter a string: FLEW
Enter a string: DROVE

The strings ending with "ED" are:
WALKED
SKIPPED
JUMPED
```

8.25 Write a program that inputs an ASCII code and prints the corresponding character. Modify this program so that it generates all possible three-digit codes in the range 000 to 255 and attempts to print the corresponding characters. What happens when this program is run?

ANS:

```c
/* Exercise 8.25 solution */
#include <stdio.h>

int main()
{
   int c; /* ASCII character */

   printf( "Enter an ASCII character code ( EOF to end ): " );

   /* while user does not enter EOF */
   while ( scanf( "%d", &c ) != EOF ) {

      /* check if character code is valid */
      if ( c >= 0 && c <= 255 ) {
         printf( "The corresponding character is '%c'\n", c );
      } /* end if */
      else {
         printf( "Invalid character code\n" );
      } /* end else */

      printf( "\nEnter an ASCII character code ( EOF to end ): " );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter an ASCII character code ( EOF to end ): 90
The corresponding character is 'Z'

Enter an ASCII character code ( EOF to end ): 116
The corresponding character is 't'

Enter an ASCII character code ( EOF to end ): 130
The corresponding character is 'é'

Enter an ASCII character code ( EOF to end ): 45
The corresponding character is '-'

Enter an ASCII character code ( EOF to end ): 40
The corresponding character is '('

Enter an ASCII character code ( EOF to end ): ^Z
```

8.26 Using the ASCII character chart in Appendix D as a guide, write your own versions of the character handling functions in Fig. 8.1.

ANS:

```
/* Exercise 8.26 Solution */
#include <stdio.h>

/* function prototypes */
int isDigit( int c );
int isAlpha( int c );
int isAlNum( int c );
int isLower( int c );
int isUpper( int c );
int toLower( int c );
int isSpace( int c );
int isPunct( int c );
int isPrint( int c );
int isGraph( int c );
int toLower( int c );
int toUpper( int c );

int main()
{
   int v; /* function result */
   char array[ 2 ] = { '\0' }; /* character from user */

   /* read a character from the user */
   printf( "Enter a character: " );
   scanf( "%c", &array[ 0 ] );

   /* test isDigit function */
   v = isDigit( ( int ) array[ 0 ] );
   printf( "According to isDigit" );
   v == 0 ? printf( " %c is not a digit\n", array[ 0 ] ):
            printf( " %c is a digit\n", array[ 0 ] );

   /* test isAlpha function */
   v = isAlpha( ( int ) array[ 0 ] );
   printf( "According to isAlpha" );
   v == 0 ? printf( " %c is not a letter\n", array[ 0 ] ):
            printf( " %c is a letter\n", array[ 0 ] );

   /* test isAlNum function */
   v = isAlNum( ( int ) array[ 0 ] );
   printf( "According to isAlNum" );
   v == 0 ? printf( " %c is not a letter or digit\n", array[ 0 ] ):
            printf( " %c is a letter or digit\n", array[ 0 ] );

   /* test isLower function */
   v = isLower( ( int ) array[ 0 ] );
   printf( "According to isLower" );
   v == 0 ? printf( " %c is not a lowercase letter\n", array[ 0 ] ):
            printf( " %c is a lowercase letter\n", array[ 0 ] );

   /* test isUpper function */
   v = isUpper( ( int ) array[ 0 ] );
   printf( "According to isUpper" );
   v == 0 ? printf( " %c is not an uppercase letter\n", array[ 0 ] ):
            printf( " %c is an uppercase letter\n", array[ 0 ] );

   /* test isSpace function */
   v = isSpace( ( int ) array[ 0 ] );
   printf( "According to isSpace" );
   v == 0 ? printf( " %c is not a white-space character\n", array[ 0 ] ):
            printf( " character is a white-space character\n" );

```

```
   /* test isPunct function */
   v = isPunct( ( int ) array[ 0 ] );
   printf( "According to isPunct" );
   v == 0 ? printf( " %c is not a punctuation character\n", array[ 0 ] ):
            printf( " %c is a punctuation character\n", array[ 0 ] );

   /* test isPrint function */
   v = isPrint( ( int ) array[ 0 ] );
   printf( "According to isPrint" );
   v == 0 ? printf( " %c is not a printing character\n", array[ 0 ] ):
            printf( " %c is a printing character\n", array[ 0 ] );

   /* test isGraph function */
   v = isGraph( ( int ) array[ 0 ] );
   printf( "According to isGraph" );
   v == 0 ? printf( " %c is not a printing character\n", array[ 0 ] ):
            printf( " %c is a printing character other than space\n", array[ 0 ] );

   /* test toLower function */
   v = toLower( ( int ) array[ 0 ] );
   printf( "According to toLower" );
   v == 0 ? printf( " %c is unchanged\n", array[ 0 ] ):
            printf( " %c has been converted to lowercase\n", v );

   /* test toUpper function */
   v = toUpper( ( int ) array[ 0 ] );
   printf( "According to toUpper" );
   v == 0 ? printf( " %c is unchanged\n", array[ 0 ] ):
            printf( " %c has been converted to uppercase\n", v );

   return 0; /* indicate successful termination */

} /* end main */

/* determines whether argument is a digit */
int isDigit( int c )
{
   return ( c >= 48 && c <= 57 ) ? 1 : 0;

} /* end function isDigit */

/* determines whether argument is a letter */
int isAlpha( int c )
{
   return ( ( c >= 65 && c <= 90 ) || ( c >= 97 && c <= 122 ) ) ? 1 : 0;

} /* end function isAlpha */

/* determines whether argument is a letter or digit */
int isAlNum( int c )
{
   return ( isDigit( c ) == 1 || isAlpha( c ) == 1 ) ? 1 : 0;

} /* end function isAlNum */

/* determines whether argument is a lowercase letter */
int isLower( int c )
{
   return ( c >= 97 && c <= 122 ) ? 1 : 0;

} /* end function isLower */

/* determines whether argument is an uppercase letter */
int isUpper( int c )
{
   return ( c >= 65 && c <= 90 ) ? 1 : 0;

} /* end function isUpper */
```

```

/* determines whether argument is a whitespace character */
int isSpace( int c )
{
   return ( ( c == 32 ) || ( c >= 9 && c <= 13 ) ) ? 1 : 0;

} /* end function isSpace */

/* determines whether argument is a printing character
   other than a space, a digit or a letter */
int isPunct( int c )
{
   return ( isAlNum( c ) == 0 && isSpace( c ) == 0 ) ? 1 : 0;

} /* end function isPunct */

/* determines whether argument is a printing character
   including the space character */
int isPrint( int c )
{
   return ( c >= 32 && c <= 126 ) ? 1 : 0;

} /* end function isPrint */

/* determines whether argument is a printing character
   other than the space character */
int isGraph( int c )
{
   return ( c >= 33 && c <= 126 ) ? 1 : 0;

} /* end function isGraph */

/* converts and uppercase letter to lowercase */
int toLower( int c )
{
   return ( isUpper( c ) == 1 ) ? c + 32 : c;

} /* end function toLower */

/* converts a lowercase letter to uppercase */
int toUpper( int c )
{
   return ( isLower( c ) == 1 ) ? c - 32 : c;

} /* end function toUpper */

```

```
Enter a character: m
According to isDigit m is not a digit
According to isAlpha m is a letter
According to isAlNum m is a letter or digit
According to isLower m is a lowercase letter
According to isUpper m is not an uppercase letter
According to isSpace m is not a white-space character
According to isPunct m is not a punctuation character
According to isPrint m is a printing character
According to isGraph m is a printing character other than space
According to toLower m has been converted to lowercase
According to toUpper M has been converted to uppercase
```

```
Enter a character: *
According to isDigit * is not a digit
According to isAlpha * is not a letter
According to isAlNum * is not a letter or digit
According to isLower * is not a lowercase letter
According to isUpper * is not an uppercase letter
According to isSpace * is not a white-space character
According to isPunct * is a punctuation character
According to isPrint * is a printing character
According to isGraph * is a printing character other than space
According to toLower * has been converted to lowercase
According to toUpper * has been converted to uppercase
```

8.27 Write your own versions of the functions in Fig. 8.5 for converting strings to numbers.

8.28 Write two versions of each of the string copy and string concatenation functions in Fig. 8.17. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

8.29 Write your own versions of the functions `getchar`, `gets`, `putchar` and `puts` described in Fig. 8.12.

8.30 Write two versions of each string comparison function in Fig. 8.20. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

8.31 Write your own versions of the functions in Fig. 8.22 for searching strings.

8.32 Write your own versions of the functions in Fig. 8.30 for manipulating blocks of memory.

8.33 Write two versions of function `strlen` in Fig. 8.36. The first version should use array subscripting, and the second version should use pointers and pointer arithmetic.

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 problems. 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.

8.34 *(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 that there is substantial evidence indicating that Christopher Marlowe 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 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.

ANS:

```c
/* Exercise 8.34 Part A Solution */
#include <stdio.h>
#include <ctype.h>

int main()
{
   char letters[ 26 ] = { 0 }; /* letters of the alphabet */
   char text[ 3 ][ 80 ];       /* three lines of text */
   int i;                      /* loop counter */
   int j;                      /* loop counter */

   printf( "Enter three lines of text:\n" );
```

```
   /* read 3 lines of text */
   for ( i = 0; i <= 2; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* loop through 3 strings */
   for ( i = 0; i <= 2; i++ ) {

      /* loop through each character */
      for ( j = 0; text[ i ][ j ] != '\0'; j++ ) {

         /* if letter, update corresponding array element */
         if ( isalpha( text[ i ][ j ] ) ) {
            ++letters[ tolower( text[ i ][ j ] ) - 'a' ];
         } /* end if */

      } /* end for */

   } /* end for */

   printf( "\nTotal letter counts:\n" );

   /* print letter totals */
   for ( i = 0; i <= 25; i++ ) {
      printf( "%c:%3d\n", 'a' + i, letters[ i ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three lines of text:
This program counts the occurrences of each
letter of the alphabet in the input text. Then,
it prints a summary of the occurrences.

Total letter counts:
a:  6
b:  1
c:  8
d:  0
e: 14
f:  3
g:  1
h:  8
i:  5
j:  0
k:  0
l:  2
m:  3
n:  7
o:  7
p:  4
q:  0
r:  9
s:  6
t: 15
u:  5
v:  0
w:  0
x:  1
y:  1
z:  0
```

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

ANS:

```c
/* Exercise 8.34 Part B solution */
#include <stdio.h>
#include <string.h>

int main()
{
   char text[ 3 ][ 80 ];        /* 3 strings from user */
   char *temp;                  /* token pointer */
   int lengths[ 20 ] = { 0 }; /* array of length counts */
   int i;                       /* loop counter */

   printf( "Enter three lines of text:\n" );

   /* read 3 lines of text */
   for ( i = 0; i <= 2; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* loop through each string */
   for ( i = 0; i <= 2; i++ ) {

      /* get first token */
      temp = strtok( &text[ i ][ 0 ], ". \n" );

      /* while temp does not equal NULL */
      while ( temp ) {

         /* increment corresponding array element */
         ++lengths[ strlen( temp ) ];
         temp = strtok( NULL, ". \n" );
      } /* end while */

   } /* end for */

   putchar( '\n' );

   /* display results in array */
   for ( i = 1; i <= 19; i++ ) {

      /* if length is not zero */
      if ( lengths[ i ] ) {
         printf( "%d word%s of length %d\n",
                 lengths[ i ], lengths[ i ] == 1 ? "" : "s", i );
```

```
      } /* end if */

   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three lines of text:
This program determines the length of each word
in the input text.  The input text here has words
of several different lengths.

3 words of length 2
4 words of length 3
6 words of length 4
3 words of length 5
1 word of length 6
3 words of length 7
1 word of length 9
1 word of length 10
```

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. A more interesting (and useful) printout should then be attempted in which the words are sorted alphabetically. 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.

ANS:

```
/* Exercise 8.34 Part C solution */
#include <stdio.h>
#include <string.h>

int main()
{
   char text[ 3 ][ 80 ];              /* 3 string from user */
   char *temp;                        /* token pointer */
   char words[ 100 ][ 20 ] = { "" }; /* array of words */
   int i;                             /* loop counter */
   int j;                             /* loop counter */
   int count[ 100 ] = { 0 };          /* array of word counts */

   printf( "Enter three lines of text:\n" );

   /* read three lines of text */
   for ( i = 0; i <= 2; i++ ) {
      gets( &text[ i ][ 0 ] );
   } /* end for */

   /* loop through 3 strings */
   for ( i = 0; i <= 2; i++ ) {

      /* get first token */
      temp = strtok( &text[ i ][ 0 ], ". \n" );

      /* while temp does not equal NULL */
      while ( temp ) {

```

```
         /* loop through words for match */
         for ( j = 0; words[ j ][ 0 ] && strcmp( temp,
            &words[ j ][ 0 ] ) != 0; j++ ) {
            ; /* empty body */
         } /* end for */

         ++count[ j ]; /* increment count */

         /* if temp could not be found in words array */
         if ( !words[ j ][ 0 ] ) {
            strcpy( &words[ j ][ 0 ], temp );
         } /* end if */

         temp = strtok( NULL, ". \n" );
      } /* end while */

   } /* end for */

   putchar( '\n' );

   /* loop through words array */
   for ( j = 0; words[ j ][ 0 ] != '\0' && j <= 99; j++ ) {
      printf( "\"%s\" appeared %d time%s\n",
            &words[ j ][ 0 ], count[ j ], count[ j ] == 1 ? "" : "s" );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three lines of text:
This program counts the number
of occurrences of each word in
the input text.

"This" appeared 1 time
"program" appeared 1 time
"counts" appeared 1 time
"the" appeared 2 times
"number" appeared 1 time
"of" appeared 2 times
"occurrences" appeared 1 time
"each" appeared 1 time
"word" appeared 1 time
"in" appeared 1 time
"input" appeared 1 time
"text" appeared 1 time
```

8.35 *(Word Processing)* The detailed treatment of string manipulation in this text is greatly attributable to the exciting growth in word processing in recent years. 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 one or more 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.

8.36 *(Printing Dates in Various Formats)* Dates are commonly printed in several different formats in business correspondence. Two of the more common formats are

`07/21/2003` and `July 21, 2003`

Write a program that reads a date in the first format and prints that date in the second format.

ANS:

```
/* Exercise 8.36 solution */
#include <stdio.h>

int main()
{

   /* array of month names */
   char *months[ 13 ] = { "", "January", "February", "March",
                          "April", "May", "June", "July",
                          "August", "September", "October",
                          "November", "December"};
   int m; /* integer month */
   int d; /* integer day */
   int y; /* integer year */

   /* read a date from user */
   printf( "Enter a date in the form mm/dd/yyyy: " );
   scanf( "%d/%d/%d", &m, &d, &y );

   /* output date in new format */
   printf( "The date is: %s %d, %d\n", months[ m ], d, y );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a date in the form mm/dd/yyyy: 06/18/2003
The date is: June 18, 2003
```

8.37 *(Check Protection)* Computers are frequently used 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 and/or machine failure. Systems designers, of course, make every effort to build controls into their systems to prevent 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 nine blank spaces in which the computer is supposed to print the amount of a weekly paycheck. If the amount is large, then all nine of those spaces will be filled, for example:

```
11,230.60  (check amount)
---------
123456789  (position numbers)
```

```
    99.87
---------
123456789
```

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
---------
    123456789
```

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.

ANS:

```c
/* Exercise 8.37 solution */
#include <stdio.h>

int main()
{
   double amount;           /* check amount */
   double base = 100000.0; /* base to check number of digits */
   int i;                   /* loop counter */
   int j;                   /* loop counter */

   /* get check amount */
   printf( "Enter check amount: " );
   scanf( "%lf", &amount );

   printf( "The protected amount is $" );

   /* loop until amount is less than base */
   for ( i = 0; amount < base; i++ ) {
      base /= 10;
   } /* end for */

   /* print i leading asterisks */
   for ( j = 1; j <= i; j++ ) {
      printf( "*" );
   } /* end for */

   printf( "%*.2f\n", 9 - i, amount );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter check amount: 234.83
The protected amount is $***234.83
```

```
Enter check amount: 14892.98
The protected amount is $*14892.98
```

```
Enter check amount: 1.54
The protected amount is $*****1.54
```

8.38 *(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 both written in numbers and "spelled out" in words. 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
```

ANS:

```
/* Exercise 8.38 solution */
/* NOTE THAT THIS PROGRAM ONLY HANDLES VALUES UP TO $99.99 */
/* The program is easily modified to process larger values */
#include <stdio.h>

int main()
{

   /* word equivalents of single digits */
   char *digits[ 10 ] = { "", "ONE", "TWO", "THREE", "FOUR",
                          "FIVE", "SIX", "SEVEN", "EIGHT", "NINE"};

   /* word equivalents of 10-19 */
   char *teens[ 10 ] = { "TEN", "ELEVEN", "TWELVE", "THIRTEEN",
                         "FOURTEEN", "FIFTEEN", "SIXTEEN",
                         "SEVENTEEN", "EIGHTEEN", "NINETEEN"};

   /* word equivalents of tens digits */
   char *tens[ 10 ] = { "", "TEN", "TWENTY", "THIRTY", "FORTY",
                        "FIFTY", "SIXTY", "SEVENTY", "EIGHTY",
                        "NINETY"};

   int dollars; /* check dollar amount */
   int cents;   /* check cents amount */
   int digit1;  /* ones digit */
   int digit2;  /* tens digit */

   /* get check amount */
   printf( "Enter the check amount ( 0.00 to 99.99 ): " );
   scanf( "%d.%d", &dollars, &cents );
   printf( "\nThe check amount in words is:\n" );

   /* print equivalent words */
   if ( dollars < 10 ) {
      printf( "%s ", digits[ dollars ] );
   } /* end if */
   else if ( dollars < 20 ) {
      printf( "%s ", teens[ dollars - 10 ] );
   } /* end else if */
   else {
      digit1 = dollars / 10; /* ones digit */
      digit2 = dollars % 10; /* tens digit */

      /* if ones digit is zero */
      if ( digit2 == 0 ) {
         printf( "%s ", tens[ digit1 ] );
      } /* end if */
      else {
         printf( "%s-%s ", tens[ digit1 ], digits[ digit2 ] );
      } /* end else */

   } /* end else */

   printf( "and %d/100\n", cents );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the check amount ( 0.00 to 99.99 ): 72.63

The check amount in words is:
SEVENTY-TWO and 63/100
```

```
Enter the check amount ( 0.00 to 99.99 ): 13.22

The check amount in words is:
THIRTEEN and 22/100
```

```
Enter the check amount ( 0.00 to 99.99 ): 5.75

The check amount in words is:
FIVE and 75/100
```

ANS:

8.39 *(Morse Code)* Perhaps the most famous of all coding schemes is Morse code, developed by Samuel Morse in 1832 for use with the telegraph system. 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,—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 Morse code appears in Fig. 8.39.

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. 8.1 The letters of the alphabet as expressed in international Morse code.

8.40 *(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 mass.

8.41 *(Dunning Letters)* Many businesses spend a great deal of time and money collecting overdue debts. *Dunning* is the process of making repeated and insistent demands upon a debtor in an attempt to collect a debt.

Computers are often used to generate dunning letters automatically and in increasing degrees of severity as a debt ages. The theory is that as a debt becomes older, it becomes more difficult to collect, and therefore the dunning letters must become more threatening.

Write a program that contains the texts of five dunning letters of increasing severity. Your program should accept as input the following:

a) Debtor's name
b) Debtor's address
c) Debtor's account
d) Amount owed
e) Age of the amount owed (i.e., one month overdue, two months overdue, etc.).

Use the age of the amount owed to select one of the five message texts, and then print the dunning letter inserting the other user-supplied information where appropriate.

A CHALLENGING STRING MANIPULATION PROJECT

8.42 *(A Crossword-Puzzle Generator)* Most people have worked a crossword puzzle at one time or another, 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.

9

C Formatted Input/Output: Solutions

SOLUTIONS

9.4 Write a `printf` or `scanf` statement for each of the following:

a) Print unsigned integer 40000 left justified in a 15-digit field with 8 digits.

ANS: `printf( "%-15.8u", ( unsigned ) 40000 );`

b) Read a hexadecimal value into variable `hex`.

ANS: `scanf( "%x", hex );`

c) Print 200 with and without a sign.

ANS: `printf( "%+d %d\n", 200, 200 );`

d) Print 100 in hexadecimal form preceded by `0x`.

ANS: `printf( %#x\n", 100 );`

e) Read characters into array `s` until the letter `p` is encountered.

ANS: `scanf( "%[^p]", s );`

f) Print `1.234` in a 9-digit field with preceding zeros.

ANS: `printf( "%09.3f\n", 1.234 );`

g) Read a time of the form `hh:mm:ss`, storing the parts of the time in the integer variables `hour`, `minute` and `second`. Skip the colons (`:`) in the input stream. Use the assignment suppression character.

ANS: `scanf( "%d%*c%d%*c%d", &hour, &minute, &second );`

h) 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.

ANS: `scanf( "\"%[^\"]", s );`

i) Read a time of the form `hh:mm:ss`, storing the parts of the time in the integer variables `hour`, `minute` and `second`. Skip the colons (`:`) in the input stream. Do not use the assignment-suppression character.

ANS: `scanf( "%d:%d:%d:", &hour, &minute, &second );`

9.5 Show what is printed by each of the following statements. If a statement is incorrect, indicate why.

a) `printf( "%-10d\n", 10000 );`

ANS: `10000`

b) `printf( "%c\n", "This is a string" );`

ANS: A string cannot be printed with the `%c` specifier.

c) `printf( "%*.*lf\n", 8, 3, 1024.987654 );`

ANS: `1024.988`

d) `printf( "%#o\n%#X\n%#e\n", 17, 17, 1008.83689 );`

ANS:

```
021
```

```
 0X11
1.008837e+03
```

e) `printf( "% ld\n%+ld\n", 1000000, 1000000 );`

ANS:

```
 1000000
+1000000
```

f) `printf( "%10.2E\n", 444.93738 );`

ANS: `4.45E+02` *preceded by two spaces*

g) `printf( "%10.2g\n", 444.93738 );`

ANS: `4.4e+02` *preceded by three spaces*

h) `printf( "%d\n", 10.987 );`

ANS: A floating point value cannot be printed with the `%d` conversion specifier.

9.6 Find the error(s) in each of the following program segments. Explain how each error can be corrected.

a) `printf( "%s\n", 'Happy Birthday' );`

ANS: `printf( “%s\n”, “Happy Birthday” );`

b) `printf( "%c\n", 'Hello' );`

ANS: `printf( “%s\n”, “Hello” );`

c) `printf( "%c\n", "This is a string" );`

ANS: `printf( “%s\n”, “This is a string” );`

d) The following statement should print `"Bon Voyage"`:

`printf( ""%s"", "Bon Voyage" );`

ANS: `printf( “\“%s\””, “Bon Voyage” );`

e)
```
char day[] = "Sunday";
printf( "%s\n", day[ 3 ] );
```

ANS: `printf( “%s\n”, day );`

f) `printf( 'Enter your name: ' );`

ANS: `printf( “Enter your name: “ );`

g) `printf( %f, 123.456 );`

ANS: `printf( “%f”, 123.456 );`

h) The following statement should print the characters `'O'` and `'K'`:

`printf( "%s%s\n", 'O', 'K' );`

ANS: `printf( “%c%c\n”, ‘O’, ‘K’ );`

i)
```
char s[ 10 ];
scanf( "%c", s[ 7 ] );
```

ANS: `scanf( “%c”, &s[ 7 ] );`

9.7 Write a program that loads 10-element array `number` with random integers from 1 to 1000. For each value, print the value and a running total of the number of characters printed. Use the `%n` conversion specifier to determine the number of characters output for each value. Print the total number of characters output for all values up to and including the current value each time the current value is printed. The output should have the following format:

```
Value     Total characters
342       3
1000      7
963       10
6         11
etc.
```

ANS:

```
/* Exercise 9.7 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main()
{
```

```
    int a[ 10 ] = { 0 }; /* random integers from 1 to 1000 */
    int i;               /* loop counter */
    int count;           /* number of characters in current value */
    int totalCount = 0;  /* total characters in array */

    srand( time( NULL ) );

    /* fill the array with random numbers */
    for ( i = 0; i <= 9; i++ ) {
       a[ i ] = 1 + rand() % 1000;
    } /* end for */

    /* print table headers */
    printf( "%s\t%s\n", "Value", "Total characters" );

    /* loop through 10 elements */
    for ( i = 0; i <= 9; i++ ) {
       printf( "%d%n", a[ i ], &count );
       totalCount+= count; /* update totalCount */
       printf( "\t%d\n", totalCount );
    } /* end for */

    return 0; /* indicate successful termination */

 } /* end main */
```

```
Value   Total characters
842     3
18      5
220     8
658     11
275     14
647     17
657     20
623     23
242     26
471     29
```

9.8 Write a program to test the difference between the %d and %i conversion specifiers when used in scanf statements. Use the statements

```
scanf( "%i%d", &x, &y );
printf( "%d %d\n", x, y );
```

to input and print the values. Test the program with the following sets of input data:

```
  10     10
 -10    -10
 010    010
0x10   0x10
```

ANS:

```
/* Exercise 9.8 Solution */
#include <stdio.h>

int main()
{
   int i; /* loop counter */
   int x; /* first integer from user */
   int y; /* second integer from user */

```

```
   /* loop four times */
   for ( i = 1; i <= 4; i++ ) {
      printf( "\nEnter two integers: " );
      scanf( "%i%d", &x, &y );
      printf( "%d %d\n", x, y );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter two integers: 10 10
10 10

Enter two integers: -10 -10
-10 -10

Enter two integers: 010 010
8 10

Enter two integers: 0x10 0x10
16 0
```

9.9 Write a program that prints pointer values using all the integer conversion specifiers and the `%p` conversion specifier. Which ones print strange values? Which ones cause errors? In which format does the `%p` conversion specifier display the address on your system?

ANS:

```
/* Exercise 9.9 Solution */
#include <stdio.h>

int main()
{
   int x; /* define x for testing */

   printf( "%o\n", &x );
   printf( "%lo\n", &x );
   printf( "%d\n", &x );
   printf( "%ld\n", &x );
   printf( "%x\n", &x );
   printf( "%lx\n", &x );
   printf( "%p\n", &x );

   return 0; /* indicate successful termination */

} /* end main */
```

```
4577574
4577574
1245052
1245052
12ff7c
12ff7c
0012FF7C
```

9.10 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?

ANS:

```
/* Exercise 9.10 Solution */
#include <stdio.h>

int main()
{

   /* print the integer 12345 */
   printf( "%10d\n", 12345 );
   printf( "%5d\n", 12345 );
   printf( "%2d\n\n", 12345 );

   /* print the floating-point value 1.2345 */
   printf( "%10f\n", 1.2345 );
   printf( "%6f\n", 1.2345 );
   printf( "%2f\n", 1.2345 );

   return 0; /* indicate successful termination */

} /* end main */
```

```
     12345
12345
12345

  1.234500
1.234500
1.234500
```

9.11 Write a program that prints the value `100.453627` rounded to the nearest digit, tenth, hundredth, thousandth and ten thousandth.

ANS:

```
/* Exercise 9.11 Solution */
#include <stdio.h>

int main()
{
   printf( "%.0f\n", 100.453627 );
   printf( "%.1f\n", 100.453627 );
   printf( "%.2f\n", 100.453627 );
   printf( "%.3f\n", 100.453627 );
   printf( "%.4f\n", 100.453627 );

   return 0; /* indicate successful termination */

} /* end main */
```

```
100
100.5
100.45
100.454
100.4536
```

9.12 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.

ANS:

```
/* Exercise 9.12 Solution */
#include <stdio.h>

int main()
{
   int count;         /* length of string */
   char string[ 20 ]; /* string entered by user */

   /* read string from user and find length */
   printf( "Enter a string:\n" );
   scanf( "%s%n", string, &count );

   printf( "%*s\n", 2 * count, string ); /* print the string */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a string:
hello
     hello
```

9.13 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 of 10 characters each, and the Celsius temperatures should be preceded by a sign for both positive and negative values.

ANS:

```
/* Exercise 9.13 Solution */
#include <stdio.h>

int main()
{
   int fahrenheit; /* holds fahrenheit temperature */
   double celcius; /* holds celcius temperature */

   printf( "%10s%12s\n", "Fahrenheit", "Celcius" );

   /* convert fahrenheit to celsius and display temperatures
      showing the sign for celsius temperatures */
   for ( fahrenheit = 0; fahrenheit <= 212; fahrenheit++ ) {
      celcius = 5.0 / 9.0 * ( fahrenheit - 32 );
      printf( "%10d%+12.3f\n", fahrenheit, celcius );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Fahrenheit     Celcius
         0     -17.778
         1     -17.222
         2     -16.667
         3     -16.111
         4     -15.556
         5     -15.000
         6     -14.444
         7     -13.889
         .
         .
         .
       204     +95.556
       205     +96.111
       206     +96.667
       207     +97.222
       208     +97.778
       209     +98.333
       210     +98.889
       211     +99.444
       212    +100.000
```

9.14 Write a program to test all the escape sequences in Figure 9.16. For the escape sequences that move the cursor, print a character before and after printing the escape sequence so it is clear where the cursor has moved.

ANS:

```
/* Exercise 9.14 Solution */
#include <stdio.h>

int main()
{

   /* test all escape sequences */
   printf( "The single quote : \'\n" );
   printf( "The double quote : \"\n" );
   printf( "The question mark: \?\n" );
   printf( "The backslash    : \\\n" );

   printf( "The bell. \a\n\n" );

   printf( "Move cursor back one position on current line. *\b*\n" );
   printf( "Move cursor to start of next logical page. *\f*\n" );

   printf( "Move cursor to the beginning of next line. *\n*\n" );
   printf( "Move cursor to the beginning of current line. *\r*\n" );

   printf( "Move cursor to the next horizontal tab position. *\t*\n" );
   printf( "Move cursor to the next vertical tab position. *\v*\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

```
The single quote : '
The double quote : "
The question mark: ?
The backslash    : \
The bell.

Move cursor back one position on current line. *
Move cursor to start of next logical page. *?*
Move cursor to the beginning of next line. *
*
*ove cursor to the beginning of current line. *
Move cursor to the next horizontal tab position. *      *
Move cursor to the next vertical tab position. *?*
```

9.15 Write a program that determines whether `?` can be printed as part of a `printf` format control string as a literal character rather than using the `\?` escape sequence.

ANS:

```c
/* Exercise 9.15 Solution */
#include <stdio.h>

int main()
{
   printf( "Did the \? print at the end of the sentence?\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Did the ? print at the end of the sentence?
```

9.16 Write a program that inputs the value `437` using each of the `scanf` integer conversion specifiers. Print each input value using all the integer conversion specifiers.

ANS:

```c
/* Exercise 9.16 Solution */
#include <stdio.h>

int main()
{
   int array[ 5 ]; /* holds the value 437 five times */
   int loop;       /* loop counter */

   /* array of table headers */
   char *s[] = { "Read with %d:", "Read with %i:", "Read with %o:",
                 "Read with %u:", "Read with %x:"};

   /* prompt the user and read 5 values */
   printf( "Enter the value 437 five times: " );
   scanf( "%d%i%o%u%x", &array[ 0 ], &array[ 1 ], &array[ 2 ],
      &array[ 3 ], &array[ 4 ] );

   /* loop through all 5 values */
   for ( loop = 0; loop <= 4; loop++ ) {

      /* print each of the 5 values */
      printf( "%s\n%d %i %o %u %x\n\n", s[ loop ], array[ loop ],
         array[ loop ], array[ loop ], array[ loop ], array[ loop ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the value 437 five times: 437 437 437 437 437
Read with %d:
437 437 665 437 1b5

Read with %i:
437 437 665 437 1b5

Read with %o:
287 287 437 287 11f

Read with %u:
437 437 665 437 1b5

Read with %x:
1079 1079 2067 1079 437
```

9.17 Write a program that uses each of the conversion specifiers `e`, `f` and `g` to input the value `1.2345`. Print the values of each variable to prove that each conversion specifier can be used to input this same value.

ANS:

```
/* Exercise 9.17 Solution */
#include <stdio.h>

int main()
{
   float a[ 3 ]; /* holds the value 1.2345 three times */

   /* array of table headers */
   char *s[] = { "Read with %e:", "Read with %f:", "Read with %g:" };

   /* prompt the user and read 3 values */
   printf( "Enter the value 1.2345 three times: " );
   scanf( "%e%f%g", &a[ 0 ], &a[ 1 ], &a[ 2 ] );

   printf( "%s%e\n\n", s[ 0 ], a[ 0 ] );
   printf( "%s%f\n\n", s[ 1 ], a[ 1 ] );
   printf( "%s%g\n\n", s[ 2 ], a[ 2 ] );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the value 1.2345 three times: 1.2345 1.2345 1.2345
Read with %e:1.234500e+000

Read with %f:1.234500

Read with %g:1.2345
```

9.18 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 by C or read as part of the string?

ANS:

```c
/* Exercise 9.18 Solution */
#include <stdio.h>

int main()
{
   char a[ 10 ]; /* first string */
   char b[ 10 ]; /* second string */
   char c[ 10 ]; /* third string */

   /* prompt user and read three strings */
   printf( "Enter the strings suzy, \"suzy\", and 'suzy':\n" );
   scanf( "%s%s%s", a, b, c );

   printf( "%s %s %s\n", a, b, c ); /* display strings */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter the strings suzy, "suzy", and 'suzy':
suzy
"suzy"
'suzy'
suzy "suzy" 'suzy'
```

9.19 Write a program that determines whether `?` can be printed as the character constant `'?'` rather than the character constant escape sequence `'\?'` using conversion specifier `%c` in the format control string of a `printf` statement.

ANS:

```c
/* Exercise 9.19 Solution */
#include <stdio.h>

int main()
{
   const char questionMark = '?'; /* define '?' as a char constant */

   printf( "This %c can be printed without using the \\\?\n",
      questionMark );

   return 0; /* indicate successful termination */

} /* end main */
```

```
This ? can be printed without using the \?
```

9.20 Write a program that uses the conversion specifier `g` to output the value `9876.12345`. Print the value with precisions ranging from 1 to 9.

ANS:

```c
/* Exercise 9.20 Solution */
#include <stdio.h>

int main()
{

   /* output the value 9876.12345 with precisions from 1 to 9 */
   printf( "Precision: %d, value = %.1g\n", 1, 9876.12345 );
   printf( "Precision: %d, value = %.2g\n", 2, 9876.12345 );
   printf( "Precision: %d, value = %.3g\n", 3, 9876.12345 );
   printf( "Precision: %d, value = %.4g\n", 4, 9876.12345 );
   printf( "Precision: %d, value = %.5g\n", 5, 9876.12345 );
   printf( "Precision: %d, value = %.6g\n", 6, 9876.12345 );
   printf( "Precision: %d, value = %.7g\n", 7, 9876.12345 );
   printf( "Precision: %d, value = %.8g\n", 8, 9876.12345 );
   printf( "Precision: %d, value = %.9g\n", 9, 9876.12345 );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Precision: 1, value = 1e+004
Precision: 2, value = 9.9e+003
Precision: 3, value = 9.88e+003
Precision: 4, value = 9876
Precision: 5, value = 9876.1
Precision: 6, value = 9876.12
Precision: 7, value = 9876.123
Precision: 8, value = 9876.1234
Precision: 9, value = 9876.12345
```

10

Structures, Unions, Bit Manipulations and Enumerations: Solutions

Solutions

10.5 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;
   float price;
   int stock;
   int reorder;
};
```

b) Union `data` containing `char c`, `short s`, `long b`, `float f` and `double d`.

ANS:

```
union data {
   char c;
   short s;
   long l;
   float f;
   double d;
};
```

c) 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 ];
};
```

d) Structure `student` that contains arrays `firstName[ 15 ]` and `lastName[ 15 ]` and variable `homeAddress` of type `struct address` from part (c).

ANS:

```
struct student {
   char firstName[ 15 ];
```

```
   char lastName[ 15 ];
   struct address homeAddress;
};
```

e) 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;
};
```

10.6 Given the following structure and variable definitions,

```
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 an expression that can be used to access 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: `customerRecord-> 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: `customerRecord->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: `customerRecord->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: `customerRecord->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: `customerRecord->personal.state`
o) Member `zipCode` of member `personal` of `customerRecord`.
ANS: `customerRecord.personal.zipCode`
p) Member `zipCode` of member `personal` of the structure pointed to by `customerPtr`.
ANS: `customerRecord->personal.zipCode`

10.7 Modify the program of Fig. 10.16 to shuffle the cards using a high performance shuffle (as shown in Fig. 10.3). Print the resulting deck in two column format as in Fig. 10.4. Precede each card with its color.

ANS:

```
/* Exercise 10.7 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* bitCard structure definition */
struct bitCard {
   unsigned face : 4;  /* 4 bits; 0-15 */
   unsigned suit : 2;  /* 2 bits; 0-3 */
   unsigned color : 1; /* 1 bit; 0-1 */
}; /* end structure bitCard */

/* new type name Card */
typedef struct bitCard Card;

/* prototypes */
void fillDeck( Card *wDeck );
void shuffle( Card *wDeck );
void deal( Card *wDeck2 );

int main()
{
   Card deck[ 52 ]; /* create array of Cards */

   srand( time( NULL ) ); /* randomize */

   fillDeck( deck );
   shuffle( deck );
   deal( deck );

   return 0; /* indicate successful termination */

} /* end main */

/* create 52 cards */
void fillDeck( Card *wDeck )
{
   int i; /* loop counter */

   /* loop 52 times and create cards */
   for ( i = 0; i <= 51; i++ ) {
      wDeck[ i ].face = i % 13;
      wDeck[ i ].suit = i / 13;
      wDeck[ i ].color = i / 26;
   } /* end for */

} /* end function fillDeck */

/* shuffle cards */
void shuffle( Card *wDeck )
{
   int i;     /* current card */
   int j;     /* random card to swap with current card */
   Card temp; /* temporary Card */

```

```
   /* loop through deck */
   for ( i = 0; i <= 51; i++ ) {
      j = rand() % 52;

      /* swap cards if not equal */
      if ( i != j ) {
         temp = wDeck[ i ];
         wDeck[ i ] = wDeck[ j ];
         wDeck[ j ] = temp;
      } /* end if */

   } /* end for */

} /* end function shuffle */

/* deal the cards */
void deal( Card *wDeck2 )
{

   /* arrays face, suit and color hold all possible string
      descriptions of the cards */
   char *face[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
      "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King"};
   char *suit[] = { "Hearts", "Diamonds", "Clubs", "Spades"};
   char *color[] = { "Red", "Black"};
   int i; /* loop counter */

   /* loop through deck and print string description of each card */
   for ( i = 0; i <= 51; i++ ) {
      printf( "%5s: %5s of %-8s", color[ wDeck2[ i ].color ],
         face[ wDeck2[ i ].face ], suit[ wDeck2[ i ].suit ] );
      putchar( ( i + 1 ) % 2 ? '\t' : '\n' );
   } /* end for */

} /* end function deal */
```

```
  Red: Eight of Diamonds         Red: Queen of Hearts
  Red:  Jack of Hearts           Red: Seven of Hearts
  Red: Three of Diamonds       Black: Eight of Spades
Black:   Ten of Spades         Black: Three of Clubs
Black:  Jack of Spades         Black: Deuce of Spades
  Red: Deuce of Diamonds         Red:   Ten of Hearts
  Red: Queen of Diamonds         Red:  King of Diamonds
Black:  Nine of Clubs          Black:   Ace of Spades
  Red: Seven of Diamonds         Red: Three of Hearts
Black:  Nine of Spades           Red:  Five of Diamonds
Black:  Jack of Clubs          Black:   Six of Spades
Black:  Five of Clubs          Black: Queen of Clubs
Black:   Ace of Clubs            Red:  Nine of Hearts
  Red:   Ten of Diamonds         Red:   Ace of Diamonds
Black: Deuce of Clubs            Red:  Four of Diamonds
Black: Seven of Clubs            Red:  King of Hearts
  Red:   Six of Hearts           Red: Deuce of Hearts
  Red:  Jack of Diamonds       Black: Three of Spades
  Red:  Four of Hearts         Black:  Four of Clubs
Black:   Ten of Clubs          Black:   Six of Clubs
  Red:  Nine of Diamonds       Black:  King of Spades
  Red:   Ace of Hearts         Black:  Five of Spades
Black:  Four of Spades         Black: Queen of Spades
Black: Seven of Spades           Red:  Five of Hearts
  Red: Eight of Hearts         Black:  King of Clubs
Black: Eight of Clubs            Red:   Six of Diamonds
```

10.8 Create union `integer` with members `char c`, `short s`, `int i` and `long b`. Write a program that inputs value 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?

ANS:

```c
/* Exercise 10.8 Solution */
/* NOTE: The program output is machine dependent */
#include <stdio.h>

/* integer union definition */
union integer {
   char c;  /* character input by user */
   short s; /* short integer input by user */
   int i;   /* integer input by user */
   long l;  /* long integer input by user */
}; /* end union integer */

int main()
{
   union integer a; /* define union a */

   /* read a character from user into the union */
   printf( "Enter a character: " );
   scanf( "%c", &a.c );

   /* print each value of union */
   printf( "\'%c'\ printed as a character is %c\n", a.c, a.c );
   printf( "\'%c'\ printed as a short integer is %hd\n", a.c, a.s );
   printf( "\'%c'\ printed as an integer is %d\n", a.c, a.i );
   printf( "\'%c'\ printed as a long integer is %ld\n", a.c, a.l );

   /* read a short integer from user into the union */
   printf( "\nEnter a short integer: " );
   scanf( "%hd", &a.s );

   /* print each value of union */
   printf( "%hd printed as a character is %c\n", a.s, a.c );
   printf( "%hd printed as a short integer is %hd\n", a.s, a.s );
   printf( "%hd printed as an integer is %d\n", a.s, a.i );
   printf( "%hd printed as a long integer is %ld\n", a.s, a.l );

   /* read an integer from user into the union */
   printf( "\nEnter an integer: " );
   scanf( "%d", &a.i );

   /* print each value of union */
   printf( "%d printed as a character is %c\n", a.i, a.c );
   printf( "%d printed as a short integer is %hd\n", a.i, a.s );
   printf( "%d printed as an integer is %d\n", a.i, a.i );
   printf( "%d printed as a long integer is %ld\n", a.i, a.l );

   /* read a long integer from user into the union */
   printf( "\nEnter a long integer: " );
   scanf( "%ld", &a.l );

   /* print each value of union */
   printf( "%ld printed as a character is %c\n", a.l, a.c );
   printf( "%ld printed as a short integer is %hd\n", a.l, a.s );
   printf( "%ld printed as an integer is %d\n", a.l, a.i );
   printf( "%ld printed as a long integer is %ld\n", a.l, a.l );

```

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

} /* end main */
```

```
Enter a character: A
'A' printed as a character is A
'A' printed as a short integer is -13247
'A' printed as an integer is -858993599
'A' printed as a long integer is -858993599

Enter a short integer: 97
97 printed as a character is a
97 printed as a short integer is 97
97 printed as an integer is -859045791
97 printed as a long integer is -859045791

Enter an integer: 32700
32700 printed as a character is +
32700 printed as a short integer is 32700
32700 printed as an integer is 32700
32700 printed as a long integer is 32700

Enter a long integer: 10000000
10000000 printed as a character is Ç
10000000 printed as a short integer is -27008
10000000 printed as an integer is 10000000
10000000 printed as a long integer is 10000000
```

10.9 Create union `floatingPoint` with members `float f`, `double d` and `long double x`. 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?

ANS:

```
/* Exercise 10.9 Solution */
/* NOTE: The program output is machine dependent */
#include <stdio.h>

/* floatingPoint union definition */
union floatingPoint {
   float f;       /* floating-point value input by user */
   double d;      /* double value input by user */
   long double l; /* long double value input by user */
}; /* end union floatingPoint */

int main()
{
   union floatingPoint a; /* define union a */

   /* read a floating-point value from user into the union */
   printf( "Enter a float: " );
   scanf( "%f", &a.f );

   /* print each value of union */
   printf( "%f printed as a float is %f\n", a.f, a.f );
   printf( "%f printed as a double is %f\n", a.f, a.d );
   printf( "%f printed as a long double is %Lf\n", a.f, a.l );

   /* read a double value from user into the union */
   printf( "\nEnter a double: " );
   scanf( "%lf", &a.d );

   /* print each value of union */
   printf( "%lf printed as a float is %f\n", a.d, a.f );
   printf( "%lf printed as a double is %f\n", a.d, a.d );
   printf( "%lf printed as a long double is %Lf\n", a.d, a.l );

   /* read a long double value from user into the union */
   printf( "\nEnter a long double: " );
   scanf( "%Lf", &a.l );

   /* print each value of union */
   printf( "%Lf printed as a float is %f\n", a.l, a.f );
   printf( "%Lf printed as a double is %f\n", a.l, a.d );
   printf( "%Lf printed as a long double is %Lf\n", a.l, a.l );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a float: 7.2
7.200000 printed as a float is 7.200000
7.200000 printed as a double is -92559604549802064000000000000000000000000000000
000000000000000.000000
7.200000 printed as a long double is -925596045498020640000000000000000000000000
00000000000000000000.000000

Enter a double: 3884.29382387423
3884.293824 printed as a float is 184710340379508000000000000000.000000
3884.293824 printed as a double is 3884.293824
3884.293824 printed as a long double is 3884.293824

Enter a long double: 833738.9384434797
833738.938443 printed as a float is -72537143835359183000.000000
833738.938443 printed as a double is 833738.938443
833738.938443 printed as a long double is 833738.938443
```

10.10 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 0s or 1s in the vacated bits?

ANS:

```c
/* Exercise 10.10 Solution */
#include <stdio.h>

void displayBits( unsigned value ); /* prototype */

int main()
{
   unsigned val; /* value from user */

   /* prompt user and read value */
   printf( "Enter an integer: " );
   scanf( "%u", &val );

   /* display value before shifting */
   printf( "%u before right shifting 4 bits is:\n", val );
   displayBits( val );

   /* display value after shifting */
   printf( "%u after right shifting 4 bits is:\n", val );
   displayBits( val >> 4 );

   return 0; /* indicate successful termination */

} /* end main */

/* function displayBits prints each bit of value */
void displayBits( unsigned value )
{
   unsigned c; /* bit counter */
   unsigned displayMask = 1 << 15; /* bit mask */

   printf( "%7u = ", value );

   /* loop through bits */
   for ( c = 1; c <= 16; c++ ) {
      value & displayMask ? putchar( '1' ) : putchar( '0' );
      value <<= 1; /* shift value 1 bit to the left */

      if ( c % 8 == 0 ) { /* print a space */
         putchar( ' ' );
      } /* end if */

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

```
Enter an integer: 1234
1234 before right shifting 4 bits is:
   1234 = 00000100 11010010
1234 after right shifting 4 bits is:
     77 = 00000000 01001101
```

10.11 If your computer uses 2-byte integers, modify the program of Fig. 10.7 so that it works with 2-byte integers.

10.12 Left shifting an `unsigned` integer by 1 bit is equivalent to multiplying the value 2. Write function `power2` that takes two integer arguments `number` and `pow` and calculates

`number * 2`^{pow}

Use the shift operator to calculate the result. Print the values as integers and as bits.

ANS:

```
/* Exercise 10.12 Solution */
#include <stdio.h>

/* prototypes */
void displayBits( unsigned value );
unsigned power2( unsigned n, unsigned p );

int main()
{
   unsigned number; /* value from user */
   unsigned pow;    /* number of bits to left shift */
   unsigned result; /* result of shift */

   /* prompt user and read two integers */
   printf( "Enter two integers: " );
   scanf( "%u%u", &number, &pow );

   /* display bits of number */
   printf( "number:\n" );
   displayBits( number );

   /* display bits of pow */
   printf( "\npow:\n" );
   displayBits( pow );

   /* perform shift and display results */
   result = power2( number, pow );
   printf( "\n%u * 2^%u = %u\n", number, pow, result );
   displayBits( result );

   return 0; /* indicate successful termination */

} /* end main */

/* function power2 left shifts n by p */
unsigned power2( unsigned n, unsigned p )
{
   return n << p;

} /* end function power2 */

/* display the bits of value */
void displayBits( unsigned value )
{
   unsigned c; /* bit counter */
   unsigned displayMask = 1 << 15; /* bit mask */

   printf( "%7u = ", value );

   /* loop through bits */
   for ( c = 1; c <= 16; c++ ) {
      value & displayMask ? putchar( '1' ) : putchar( '0' );
```

```
      value <<= 1; /* shift value 1 bit to the left */

      if ( c % 8 == 0 ) { /* print a space */
         putchar( ' ' );
      } /* end if */

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

```
Enter two integers: 10 3
number:
     10 = 00000000 00001010

pow:
      3 = 00000000 00000011

10 * 2^3 = 80
     80 = 00000000 01010000
```

10.13 The left-shift operator can be used to pack two character values into an `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.

ANS:

```
/* Exercise 10.13 Solution */
#include <stdio.h>

/* prototypes */
unsigned packCharacters( char x, char y );
void displayBits( unsigned value );

int main()
{
   char a;           /* first character from user */
   char b;           /* second character from user */
   unsigned result; /* result of packing both characters */

   /* prompt user and read two characters */
   printf( "Enter two characters: " );
   scanf( "%c %c", &a, &b );

   /* display first character as bits */
   printf( "\'%c\' in bits as an unsigned integers is:\n", a );
   displayBits( a );

   /* display second character as bits */
   printf( "\n\'%c\' in bits as an unsigned integers is:\n", b );
   displayBits( b );

   /* pack characters and display result */
   result = packCharacters( a, b );
   printf( "\n\'%c\' and \'%c\' packed in an unsigned integer:\n",
      a, b );
```

```
   displayBits( result );

   return 0; /* indicate successful termination */

} /* end main */

/* function packCharacters packs two characters into an unsigned int */
unsigned packCharacters( char x, char y )
{
   unsigned pack = x; /* initialize pack to x */

   pack <<= 8; /* shift pack 8 bits to the left */
   pack |= y; /* pack y using inclusive OR operator */
   return pack;

} /* end function packCharacters */

/* display the bits of value */
void displayBits( unsigned value )
{
   unsigned c; /* bit counter */
   unsigned displayMask = 1 << 15; /* bit mask */

   printf( "%7u = ", value );

   /* loop through bits */
   for ( c = 1; c <= 16; c++ ) {
      value & displayMask ? putchar( '1' ) : putchar( '0' );
      value <<= 1; /* shift value 1 bit to the left */

      if ( c % 8 == 0 ) { /* print a space */
         putchar( ' ' );
      } /* end if */

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

```
Enter two characters: A B
'A' in bits as an unsigned integers is:
     65 = 00000000 01000001

'B' in bits as an unsigned integers is:
     66 = 00000000 01000010

'A' and 'B' packed in an unsigned integer:
  16706 = 01000001 01000010
```

10.14 Using the right-shift operator, the bitwise AND operator and a mask, write function `unpackCharacters` that takes the `unsigned` integer from Exercise 10.13 and unpacks it into two characters. To unpack two characters from an `unsigned` integer, combine the unsigned integer with the mask `65280` (00000000 00000000 11111111 00000000) 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 00000000 00000000 11111111). Assign the result to another `char` variable. The program should print the `unsigned` integer in bits before it is unpacked, then print the characters in bits to confirm that they were unpacked correctly.

ANS:

```c
/* Exercise 10.14 Solution */
#include <stdio.h>

/* prototypes */
void unpackCharacters( char *aPtr, char *bPtr, unsigned pack );
void displayBits( unsigned value );

int main()
{
   char a; /* first character unpacked */
   char b; /* second character unpacked */
   unsigned packed = 16706; /* initialize packed value */

   /* display bits of packed */
   printf( "The packed character representation is:\n" );
   displayBits( packed );

   /* unpack packed and display results */
   unpackCharacters( &a, &b, packed );
   printf( "\nThe unpacked characters are \'%c\' and \'%c\'\n", a, b );
   displayBits( a );
   displayBits( b );

   return 0; /* indicate successful termination */

} /* end main */

/* unpack two characters from pack */
void unpackCharacters( char *aPtr, char *bPtr, unsigned pack )
{
   unsigned mask1 = 65280; /* mask for first character */
   unsigned mask2 = 255;   /* mask for second character */

   *aPtr = ( pack & mask1 ) >> 8; /* separate first character */
   *bPtr = ( pack & mask2 ); /* separate second character */
} /* end function unpackCharacters */

/* display the bits of value */
void displayBits( unsigned value )
{
   unsigned c; /* bit counter */
   unsigned displayMask = 1 << 15; /* bit mask */

   printf( "%7u = ", value );

   /* loop through bits */
   for ( c = 1; c <= 16; c++ ) {
      value & displayMask ? putchar( '1' ) : putchar( '0' );
      value <<= 1; /* shift value 1 bit to the left */

      if ( c % 8 == 0 ) { /* print a space */
         putchar( ' ' );
      } /* end if */
```

```

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

```
The packed character representation is:
  16706 = 01000001 01000010

The unpacked characters are 'A' and 'B'
     65 = 00000000 01000001
     66 = 00000000 01000010
```

10.15 If your system uses 4-byte integers, rewrite the program of Exercise 10.13 to pack 4 characters.

10.16 If your system uses 4-byte integers, rewrite the function `unpackCharacters` of Exercise 10.14 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).

10.17 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.

ANS:

```
/* Exercise 10.17 Solution */
#include <stdio.h>

/* prototypes */
unsigned reverseBits( unsigned value );
void displayBits( unsigned value );

int main()
{
   unsigned a; /* unsigned integer from user */

   /* prompt user and read value */
   printf( "Enter an unsigned integer: " );
   scanf( "%u", &a );

   /* display bits of a before reversed */
   printf( "\nBefore bits are reversed:\n" );
   displayBits( a );

   /* reverse bits and display results */
   a = reverseBits( a );
   printf( "\nAfter bits are reversed:\n" );
   displayBits( a );

   return 0; /* indicate successful termination */

} /* end main */

/* reverseBits reverses the bits of value */
unsigned reverseBits( unsigned value )
{
   unsigned mask = 1; /* bit mask */
   unsigned temp = 0; /* reversed bits */
   int i;             /* loop counter */

   /* loop through bits of value */
   for ( i = 0; i <= 15; i++ ) {
      temp <<= 1; /* right shift 1 bit */
      temp |= ( value & mask ); /* separate bit and place in temp */
      value >>= 1; /* left shift 1 bit */
   } /* end for */

   return temp;

} /* end function reverseBits */

/* display the bits of value */
void displayBits( unsigned value )
{
   unsigned c; /* bit counter */
   unsigned displayMask = 1 << 15; /* bit mask */

   printf( "%7u = ", value );

   /* loop through bits */
   for ( c = 1; c <= 16; c++ ) {
```

```
      value & displayMask ? putchar( '1' ) : putchar( '0' );
      value <<= 1; /* shift value 1 bit to the left */

      if ( c % 8 == 0 ) { /* print a space */
         putchar( ' ' );
      } /* end if */

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

```
Enter an unsigned integer: 2127

Before bits are reversed:
   2127 = 00001000 01001111

After bits are reversed:
  61968 = 11110010 00010000
```

10.18 Modify function `displayBits` of Fig. 10.7 so it is portable between systems using 2-byte integers and systems using 4-byte integers. [*Hint:* Use the `sizeof` operator to determine the size of an integer on a particular machine.]

ANS:

```
/* Exercise 10.18 Solution */
#include <stdio.h>

void displayBits( unsigned value ); /* prototype */

int main()
{
   unsigned x; /* value from user */

   /* prompt user and read value */
   printf( "Enter an unsigned integer: " );
   scanf( "%u", &x );
   displayBits( x );

   return 0; /* indicate successful termination */

} /* end main */

/* display the bits of value */
void displayBits( unsigned value )
{
   unsigned c; /* bit counter */
   unsigned displayMask; /* bit mask */

   /* if system uses 4-byte integers */
   if ( sizeof( int ) == 4 ) {
      displayMask = 1 << 31;
   } /* end if */
   else { /* assume default of 2-byte integers */
      displayMask = 1 << 15;
   } /* end else */

   printf( "%7u = ", value );

   /* loop through bits */
   for ( c = 1; c <= sizeof( int ) * 8; c++ ) {
      putchar( value & displayMask ? '1' : '0' );
      value <<= 1; /* shift value 1 bit to the left */

      if ( c % 8 == 0 ) { /* print a space */
         putchar( ' ' );
      } /* end if */

   } /* end for */

   putchar( '\n' );
} /* end function displayBits */
```

```
Enter an unsigned integer: 2345
   2345 = 00000000 00000000 00001001 00101001
```

10.19 The following program uses function `multiple` to determine if the integer entered from the keyboard is a multiple of some integer X. Examine the function multiple, then determine the value of X.

```c
/* ex10_19.c */
/* This program determines if a value is a multiple of X. */
#include <stdio.h>

int multiple( int num ); /* prototype */

int main()
{
   int y; /* y will hold an integer entered by the user  */

   printf( "Enter an integer between 1 and 32000: " );
   scanf( "%d", &y );

   /* if y is a multiple of X */
   if ( multiple( y ) ) {
      printf( "%d is a multiple of X\n", y );
   } /* end if */
   else {
      printf( "%d is not a multiple of X\n", y );
   } /* end else */

   return 0; /* indicates successful termination */
} /* end main */

/* determine if num is a multiple of X */
int multiple( int num )
{
   int i;        /* counter */
   int mask = 1; /* initialize mask */
   int mult = 1; /* initialize mult */

   for ( i = 1; i <= 10; i++, mask <<= 1 ) {

      if ( ( num & mask ) != 0 ) {
         mult = 0;
         break;
      } /* end if */

   } /* end for */

   return mult;
} /* end function multiple */
```

ANS:

```
Enter an integer between 1 and 32000: 1024
1024 is a multiple of X
```

10.20 What does the following program do?

```
/* ex10_20.c */
#include <stdio.h>

int mystery( unsigned bits ); /* prototype */

int main()
{
   unsigned x; /* x will hold an integer entered by the user */

   printf( "Enter an integer: " );
   scanf( "%u", &x );

   printf( "The result is %d\n", mystery( x ) );

   return 0; /* indicates successful termination */
} /* end main */

/* What does this function do? */
int mystery( unsigned bits )
{
   unsigned i;              /* counter */
   unsigned mask = 1 << 31; /* initialize mask */
   unsigned total = 0;      /* initialize total */

   for ( i = 1; i <= 32; i++, bits <<= 1 ) {

      if ( ( bits & mask ) == mask ) {
         total++;
      } /* end if */

   } /* end for */

   return !( total % 2 ) ? 1 : 0;
} /* end function mystery */
```

ANS:

```
Enter an integer: 5678
The result is 0
```

```
Enter an integer: 65
The result is 1
```

11

C File Processing: Solutions

SOLUTIONS

11.5 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(n) __________ is composed of several fields.

ANS: record.

c) A field that may contain digits, letters and blanks is called a(n) __________ field.

ANS: alphanumeric.

d) To facilitate the retrieval of specific records from a file, one field in each record is chosen as a(n) __________.

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(n) __________.

ANS: field.

g) The file pointers for the three files that are opened automatically when program execution begins are named __________, __________ and __________.

ANS: `stdin, stdout, stderr`.

h) Function __________ writes a character to a specified file.

ANS: `fputc`.

i) Function __________ writes a line to a specified file.

ANS: `fputs`.

j) Function __________ is generally used to write data to a random-access file.

ANS: `fwrite`.

k) Function __________ repositions the file position pointer to the beginning of the file.

ANS: `rewind`.

11.6 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 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 processed by a computer form a data hierarchy in which data items become larger and more complex as we progress from fields to characters to bits etc.

ANS: Data items process 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 their information in a single file to facilitate computer processing.

ANS: False. Most organizations have many files in which they store their information.

i) Files are always referred to by name in C programs.

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.

11.7 Exercise 11.3 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 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 (i.e., sales are made and cash payments arrive in the mail), they are entered into a file. At the end of each business period (i.e., a month for some companies, a week for others and a day in some cases) the file of transactions (called `"trans.dat"` in Exercise 11.3) is applied to the master file (called `"oldmast.dat"` in Exercise 11.3), thus updating each account's record of purchases and payments. After each of these updatings 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 might 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 might 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 written in Exercise 11.3 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 account number order.

When a match occurs (i.e., records with the same account number appear on both the master file and the transaction file), add the dollar amount on the transaction file to the current balance on the master file and write the `"newmast.dat"` record. (Assume that purchases are indicated by positive amounts on the transaction file, and that 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).

ANS:

```c
/* Exercise 11.7 Solution */
/* NOTE: This program was run using the */
/* data in Exercise 11.8  */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int masterAccount;          /* account from old master file */
   int transactionAccount;     /* account from transactions file */
   double masterBalance;       /* balance from old master file */
   double transactionBalance; /* balance from transactions file */
   char masterName[ 30 ];      /* name from master file */
   FILE *ofPtr;                /* old master file pointer */
```

```
   FILE *tfPtr;               /* transactions file pointer */
   FILE *nfPtr;               /* new master file pointer */

   /* terminate application if old master file cannot be opened */
   if ( ( ofPtr = fopen( "oldmast.dat", "r" ) ) == NULL ) {
      printf( "Unable to open oldmast.dat\n" );
      exit( 1 );
   } /* end if */

   /* terminate application if transactions file cannot be opened */
   if ( ( tfPtr = fopen( "trans.dat", "r" ) ) == NULL ) {
      printf( "Unable to open trans.dat\n" );
      exit( 1 );
   } /* end if */

   /* terminate application if new master file cannot be opened */
   if ( ( nfPtr = fopen( "newmast.dat", "w" ) ) == NULL ) {
      printf( "Unable to open newmast.dat\n" );
      exit( 1 );
   } /* end if */

   /* display account currently being processed */
   printf( "Processing....\n" );
   fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );

   /* while not the end of transactions file */
   while ( !feof( tfPtr ) ) {

      /* read next record from old master file */
      fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount, masterName,
         &masterBalance );

      /* display accounts from master file until number of
         new account is reached */
      while ( masterAccount < transactionAccount && !feof( ofPtr ) ) {
         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );
         printf( "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );

         /* read next record from old master file */
         fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount,
                masterName, &masterBalance );
      } /* end while */

      /* if matching account found, update balance and output
         account info */
      if ( masterAccount == transactionAccount ) {
         masterBalance += transactionBalance;
         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );
         printf( "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );
      } /* end if */

      /* tell user if account from transactions file does
         not match account from master file */
      else if ( masterAccount > transactionAccount ) {
         printf( "Unmatched transaction record for account %d\n",
                transactionAccount );
         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );
         printf( "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );
      } /* end else if */
      else {
         printf( "Unmatched transaction record for account %d\n",
                transactionAccount );
      } /* end else */
```

```

      /* get next account and balance from transactions file */
      fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );
   } /* end while */

   /* loop through file and display account number, name and balance */
   while ( !feof( ofPtr ) ) {
      fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount, masterName,
         &masterBalance );
      fprintf( nfPtr, "%d %s %.2f", masterAccount, masterName,
         masterBalance );
      printf( "%d %s %.2f", masterAccount, masterName, masterBalance );
   } /* end while */

   fclose( ofPtr ); /* close all file pointers */
   fclose( tfPtr );
   fclose( nfPtr );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Processing....
100   Alan Jones   375.31
300   Mary Smith   89.30
Unmatched transaction record for account 400
500   Sam Sharp   0.00
700   Suzy Green   -14.22
Unmatched transaction record for account 900
```

11.8 After writing the program of Exercise 11.7, write a simple program to create some test data for checking out the program of Exercise 11.7. 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**	**Dollar amount**
100	27.14
300	62.11
400	100.56
900	82.17

ANS:

```
/* Exercise 11.8 Solution */
#include <stdio.h>

int main()
{
   int account;     /* account number */
```

```
   char name[ 30 ]; /* account name */
   double balance;  /* account balance */
   double amount;   /* transaction amount */
   FILE *ofPtr;     /* old master file pointer */
   FILE *tfPtr;     /* transaction file pointer */

   /* open both files for writing */
   ofPtr = fopen( "oldmast.dat", "w" );
   tfPtr = fopen( "trans.dat", "w" );

   /* prompt user for sample data */
   printf( "Sample data for file oldmast.dat:\n" );
   printf( "Enter account, name, and balance (EOF to end): " );

   /* loop while EOF character not entered by user */
   while ( scanf( "%d%[^0-9-]%lf", &account, name,
      &balance ) != EOF ) {

      /* write data to old master file */
      fprintf( ofPtr, "%d %s %.2f\n", account, name, balance );
      printf( "Enter account, name, and balance (EOF to end): " );
   } /* end while */

   fclose( ofPtr ); /* close file pointer */

   /* prompt user for sample data */
   printf( "\nSample data for file trans.dat:\n" );
   printf( "Enter account and transaction amount (EOF to end): " );

   /* loop while EOF character not entered by user */
   while ( scanf( "%d%lf", &account, &amount ) != EOF ) {

      /* write data to transactions file */
      fprintf( tfPtr, "%d %.2f\n", account, amount );
      printf( "Enter account and transaction amount (EOF to end): " );
   } /* end while */

   fclose( tfPtr ); /* close file pointer */

   return 0; /* indicate successful termination */

} /* end main */
```

```
Sample data for file oldmast.dat:
Enter account, name, and balance (EOF to end): 100 Alan Jones 348.17
Enter account, name, and balance (EOF to end): 300 Mary Smith 27.19
Enter account, name, and balance (EOF to end): 500 Sam Sharp 0.00
Enter account, name, and balance (EOF to end): 700 Suzy Green -14.22
Enter account, name, and balance (EOF to end): ^Z

Sample data for file trans.dat:
Enter account and transaction amount (EOF to end): 100 27.14
Enter account and transaction amount (EOF to end): 300 62.11
Enter account and transaction amount (EOF to end): 400 100.56
Enter account and transaction amount (EOF to end): 900 82.17
Enter account and transaction amount (EOF to end): ^Z
```

11.9 Run the program of Exercise 11.7 using the files of test data created in Exercise 11.8. Use the listing program of Section 11.7 to print the new master file. Check the results carefully.

11.10 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 11.7 to provide for the possibility of handling several transaction records with the same record key. Modify the test data of Exercise 11.8 to include the following additional transaction records:

Account number	Dollar amount
300	83.89
700	80.78
700	1.53

ANS:

```
/* Exercise 11.10 Solution */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int masterAccount;          /* account from old master file */
   int transactionAccount;     /* account from transactions file */
   double masterBalance;       /* balance from old master file */
   double transactionBalance; /* balance from transactions file */
   char masterName[ 30 ];      /* name from master file */
   FILE *ofPtr;                /* old master file pointer */
   FILE *tfPtr;                /* transactions file pointer */
   FILE *nfPtr;                /* new master file pointer */

   /* terminate application if old master file cannot be opened */
   if ( ( ofPtr = fopen( "oldmast.dat", "r" ) ) == NULL ) {
      printf( "Unable to open oldmast.dat\n" );
      exit( 1 );
   } /* end if */

   /* terminate application if transactions file cannot be opened */
   if ( ( tfPtr = fopen( "trans.dat", "r" ) ) == NULL ) {
      printf( "Unable to open trans.dat\n" );
      exit( 1 );
   } /* end if */

   /* terminate application if new master file cannot be opened */
   if ( ( nfPtr = fopen( "newmast.dat", "w" ) ) == NULL ) {
      printf( "Unable to open newmast.dat\n" );
      exit( 1 );
   } /* end if */

   /* display account currently being processed */
   printf( "Processing....\n" );
   fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );

   /* while not the end of transactions file */
   while ( !feof( tfPtr ) ) {

      /* read next record from old master file */
      fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount, masterName,
         &masterBalance );

      /* display accounts from master file until number of
         new account is reached */
      while ( masterAccount < transactionAccount && !feof( ofPtr ) ) {
         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );
```

```
         printf( "%d %s %.2f\n", masterAccount, masterName,
            masterBalance );

         /* read next record from old master file */
         fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount,
  masterName, &masterBalance );
      } /* end while */

      /* if matching account found, update balance and output
         account info */
      if ( masterAccount == transactionAccount ) {

         /* while more transactions exist for current account */
         while ( masterAccount == transactionAccount &&
            !feof( tfPtr ) ) {

            /* update masterBalance and read next record */
            masterBalance += transactionBalance;
            fscanf( tfPtr, "%d%lf", &transactionAccount,
               &transactionBalance );
         } /* end while */

         fprintf( nfPtr, "%d %s %.2f\n",
                  masterAccount, masterName, masterBalance );
         printf( "%d %s %.2f\n", masterAccount, masterName, masterBalance );
      } /* end if */

      /* tell user if account from transactions file does
         not match account from master file */
      else if ( masterAccount > transactionAccount ) {
         printf( "Unmatched transaction record for account %d\n",
                 transactionAccount );
         fprintf( nfPtr, "%d %s %.2f\n", masterAccount, masterName, masterBalance );
         printf( "%d %s %.2f\n", masterAccount, masterName, masterBalance );
         fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );
      } /* end else if */
      else {
         printf( "Unmatched transaction record for account %d\n",
                 transactionAccount );
         fscanf( tfPtr, "%d%lf", &transactionAccount, &transactionBalance );
      } /* end else */

   } /* end while */

   /* loop through file and display account number, name and balance */
   while ( !feof( ofPtr ) ) {
      fscanf( ofPtr, "%d%[^0-9-]%lf", &masterAccount, masterName,
         &masterBalance );
      fprintf( nfPtr, "%d %s %.2f", masterAccount, masterName,
         masterBalance );
      printf( "%d %s %.2f", masterAccount, masterName, masterBalance );
   } /* end while */

   fclose( ofPtr ); /* close all file pointers */
   fclose( tfPtr );
   fclose( nfPtr );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Processing....
100   Alan Jones   375.31
300   Mary Smith   173.19
Unmatched transaction record for account 400
500   Sam Sharp   0.00
700   Suzy Green   68.09
Unmatched transaction record for account 900
```

11.11 Write statements that accomplish each of the following. Assume that the structure

```
struct person {
   char lastName[ 15 ];
   char firstName[ 15 ];
   char age[ 4 ];
};
```

has been defined and that the file is already open for writing.

a) Initialize the file "`nameage.dat`" so that there are 100 records with `lastName` = "`unassigned`", `firstname` = `""` and `age` = "`0`".
b) Input 10 last names, first names and ages, and write them to the file.
c) Update a record; if there is no information in the record, tell the user "`No info`".
d) Delete a record that has information by reinitializing that particular record.

11.12 You are the owner of a hardware store and need to keep an inventory that can tell you what tools you have, how many you have and the cost of each one. Write a program that initializes the file "`hardware.dat`" to 100 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

11.13 *Telephone Number Word Generator.* Standard telephone keypads contain the digits 0 through 9. The numbers 2 through 9 each have three letters associated with them, as is indicated by the following table:

Digit	Letter
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P R S
8	T U V
9	W X Y

Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2377 might use the correspondence indicated in the above table to develop the seven-letter word "NUMBERS."

Businesses frequently attempt to get telephone numbers that are easy for their clients to remember. If a business can advertise a simple word for its customers to dial, then no doubt the business will receive a few more calls.

Each seven-letter word corresponds to exactly one seven-digit telephone number. The restaurant wishing to increase its take-home business could surely do so with the number 825-3688 (i.e., "TAKEOUT").

Each seven-digit phone number corresponds to many separate seven-letter words. Unfortunately, most of these represent unrecognizable juxtapositions of letters. It is possible, however, that the owner of a barber shop would be pleased to know that the shop's telephone number, 424-7288, corresponds to "HAIRCUT." The owner of a liquor store would, no doubt, be delighted to find that the store's telephone number, 233-7226, corresponds to "BEERCAN." A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to the letters "PETCARE."

Write a C program that, given a seven-digit number, writes to a file every possible seven-letter word corresponding to that number. There are 2187 (3 to the seventh power) such words. Avoid phone numbers with the digits 0 and 1.

ANS:

```c
/* Exercise 11.13 Solution */
#include <stdio.h>

void wordGenerator( int number[] ); /* prototype */

int main()
{
   int loop; /* loop counter */
   int phoneNumber[ 7 ] = { 0 }; /* holds phone number */

   /* prompt user to enter phone number */
   printf( "Enter a phone number one digit at a time" );
   printf( " using the digits 2 thru 9:\n" );

   /* loop 7 times to get number */
   for ( loop = 0; loop <= 6; loop++ ) {
      printf( "? " );
      scanf( "%d", &phoneNumber[ loop ] );

      /* test if number is between 0 and 9 */
      while ( phoneNumber[ loop ] < 2 || phoneNumber[ loop ] > 9 ) {
         printf( "\nInvalid number entered. Please enter again: " );
         scanf( "%d", &phoneNumber[ loop ] );
      } /* end while */

   } /* end for */

   wordGenerator( phoneNumber ); /* form words from phone number */

   return 0; /* indicate successful termination */

} /* end main */

/* function to form words based on phone number */
void wordGenerator( int number[] )
{
   int loop;  /* loop counter */
   int loop1; /* loop counter for first digit of phone number */
   int loop2; /* loop counter for second digit of phone number */
   int loop3; /* loop counter for third digit of phone number */
   int loop4; /* loop counter for fourth digit of phone number */
   int loop5; /* loop counter for fifth digit of phone number */
   int loop6; /* loop counter for sixth digit of phone number */
   int loop7; /* loop counter for seventh digit of phone number */
   FILE *foutPtr; /* output file pointer */

```

```
   /* letters corresponding to each number */
   char *phoneLetters[ 10 ] = { "", "", "ABC", "DEF", "GHI", "JKL",
                              "MNO", "PRS", "TUV", "WXY"};

   /* open output file */
   if ( ( foutPtr = fopen( "phone.out", "w" ) ) == NULL ) {
      printf( "Output file was not opened.\n" );
   } /* end if */
   else { /* print all possible combinations */

      for ( loop1 = 0; loop1 <= 2; loop1++ ) {

         for ( loop2 = 0; loop2 <= 2; loop2++ ) {

            for ( loop3 = 0; loop3 <= 2; loop3++ ) {

               for ( loop4 = 0; loop4 <= 2; loop4++ ) {

                  for ( loop5 = 0; loop5 <= 2; loop5++ ) {

                     for ( loop6 = 0; loop6 <= 2; loop6++ ) {

                        for ( loop7 = 0; loop7 <= 2; loop7++ ) {
                           fprintf( foutPtr, "%c%c%c%c%c%c%c\n",
                              phoneLetters[ number[ 0 ] ][ loop1 ],
                              phoneLetters[ number[ 1 ] ][ loop2 ],
                              phoneLetters[ number[ 2 ] ][ loop3 ],
                              phoneLetters[ number[ 3 ] ][ loop4 ],
                              phoneLetters[ number[ 4 ] ][ loop5 ],
                              phoneLetters[ number[ 5 ] ][ loop6 ],
                              phoneLetters[ number[ 6 ] ][ loop7 ] );
                        } /* end for */

                     } /* end for */

                  } /* end for */

               } /* end for */

            } /* end for */

         } /* end for */

      } /* end for */

      /* output phone number */
      fprintf( foutPtr, "\nPhone number is " );

      /* loop through digits */
      for ( loop = 0; loop <= 6; loop++ ) {

         /* insert hyphen */
         if ( loop == 3 ) {
            fprintf( foutPtr, "-" );
         } /* end if */

         fprintf( foutPtr, "%d", number[ loop ] );
      } /* end for */

   } /* end else */

   fclose( foutPtr ); /* close file pointer */
} /* end function wordGenerator
```

```
Enter a phone number one digit at a time using the digits 2 thru 9:
? 8
? 4
? 3
? 2
? 6
? 7
? 7
```

The contents of phone.out are:

```
TGDAMPP
TGDAMPR
TGDAMPS
TGDAMRP
TGDAMRR
TGDAMRS
TGDAMSP
TGDAMSR
.
.
.
VIFCORP
VIFCORR
VIFCORS
VIFCOSP
VIFCOSR
VIFCOSS

Phone number is 843-2677
```

11.14 If you have a computerized dictionary available, modify the program you wrote in Exercise 11.13 to look up the words in the dictionary. Some seven-letter combinations created by this program consist of two or more words (the phone number 843-2677 produces "THEBOSS").

11.15 Modify the example of Fig. 8.14 to use functions `fgetc` and `fputs` rather than `getchar` and `puts`. The program should give the user the option to read from the standard input and write to the standard output or to read from a specified file and write to a specified file. If the user chooses the second option, have the user enter the file names for the input and output files.

ANS:

```
/* Exercise 11.15 Solution */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char c;                /* current character */
   char sentence[ 80 ]; /* text from user or input file */
   char input[ 20 ];     /* input file */
   char output[ 20 ];    /* output file */
   char choice[ 2 ];     /* user's menu choice */
   int i = 0;            /* character counter */
   FILE *infilePtr;      /* input file pointer */
   FILE *outfilePtr;     /* output file pointer */

   /* display choices to user */
   printf( "%s%s\n%s\n%s", "1 Read from standard input; ",
      "write to standard output", "2 Read from a file; write to file",
      "Enter choice: " );
   scanf( "%s", choice );

   /* while user does not enter a valid choice */
   while ( choice[ 0 ] != '1' && choice[ 0 ] != '2' ) {
      printf( "Invalid choice. Choose again: " );
      scanf( "%s", choice );
   } /* end while */

   /* if user chooses option 2 */
   if ( choice[ 0 ] == '2' ) {
      printf( "Enter input file name: " ); /* get input file name */
      scanf( "%s", input );

      printf( "Enter output file name: " ); /* get output file name */
      scanf( "%s", output );

      /* exit program if unable to open input file */
      if ( ( infilePtr = fopen( input, "r" ) ) == NULL ) {
         printf( "Unable to open %s\n", input );
         exit( 1 );
      } /* end if */

      /* exit program if unable to open output file */
      else if ( ( outfilePtr = fopen( output, "w" ) ) == NULL ) {
         printf( "Unable to open %s\n", output );
         fclose( infilePtr );
         exit( 1 );
      } /* end if */

   } /* end if */
   else { /* if user chooses option 1 */
      infilePtr = stdin;
      outfilePtr = stdout;
   } /* end else */

   /* if user chooses option 1 */
   if ( choice[ 0 ] == '1' ) {

      /* prompt user for text */
      printf( "Enter a line of text:\n" );
      scanf( " " ); /* Eliminate spaces and newlines at the
                       start of the input stream */
   } /* end if */
```

```

   /* read each character using fgetc */
   while ( ( c = fgetc( infilePtr ) ) != '\n' && !feof( infilePtr ) ) {
      sentence[ i++ ] = c;
   } /* end while */

   /* add terminating character and output text with fputs */
   sentence[ i ] = '\0';
   fprintf( outfilePtr, "\nThe line entered was:\n" );
   fputs( sentence, outfilePtr );

   /* close file pointers */
   if ( choice[ 0 ] == '2' ) {
      fclose( infilePtr );
      fclose( outfilePtr );
   } /* end if */

   return 0; /* indicate successful termination */

} /* end main */
```

```
1 Read from standard input; write to standard output
2 Read from a file; write to file
Enter choice: 1
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

```
1 Read from standard input; write to standard output
2 Read from a file; write to a file
Enter choice: 2
Enter input file name: test.dat
Enter output file name: output.dat
```

Contents of test.dat

```
This is a test file for exercise 11.15.
```

Contents of output.dat

```
The line entered was:
This is a test file for exercise 11.15.
```

11.16 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 as follows:

```
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               16
```

[*Note:* The type sizes on your computer might be different from those listed above.]

ANS:

```c
/* Exercise 11.16 Solution */
#include <stdio.h>

int main()

{
   FILE *outPtr; /* output file pointer */

   /* open datasize.dat for writing */
   outPtr = fopen( "datasize.dat", "w" );

   /* write size of various data types */
   fprintf( outPtr, "%s%16s\n", "Data type", "Size" );
   fprintf( outPtr, "%s%21d\n", "char", sizeof( char ) );
   fprintf( outPtr, "%s%12d\n", "unsigned char",
      sizeof( unsigned char ) );
   fprintf( outPtr, "%s%16d\n", "short int", sizeof( short int ) );
   fprintf( outPtr, "%s%7d\n", "unsigned short int",
      sizeof( unsigned short int ) );
   fprintf( outPtr, "%s%22d\n", "int", sizeof( int ) );
   fprintf( outPtr, "%s%13d\n", "unsigned int",
      sizeof( unsigned int ) );
   fprintf( outPtr, "%s%17d\n", "long int", sizeof( long int ) );
   fprintf( outPtr, "%s%8d\n", "unsigned long int",
      sizeof( unsigned long int ) );
   fprintf( outPtr, "%s%20d\n", "float", sizeof( float ) );
   fprintf( outPtr, "%s%19d\n", "double", sizeof( double ) );
   fprintf( outPtr, "%s%14d\n", "long double", sizeof( long double ) );

   fclose( outPtr ); /* close file pointer */

   return 0; /* indicate successful termination */

} /* end main */
```

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
```

11.17 In Exercise 7.19, you wrote a software simulation of a computer that used a special machine language called Simpletron Machine Language (SML). In the simulation, each time you wanted to run an SML program, you entered the program into the simulator from the keyboard. If you made a mistake while typing the SML program, the simulator was restarted and the SML code was reentered. It would be nice to be able to read the SML program from a file rather than type it each time. This would reduce time and mistakes in preparing to run SML programs.

a) Modify the simulator you wrote in Exercise 7.19 to read SML programs from a file specified by the user at the keyboard.

b) After the Simpletron executes, it outputs the contents of its registers and memory on the screen. It would be nice to capture the output in a file, so modify the simulator to write its output to a file in addition to displaying the output on the screen.

ANS:

```
/* Exercise 11.17 Solution */
#include <stdio.h>

/* define commands */
#define SIZE 100
#define TRUE 1
#define FALSE 0
#define READ 10
#define WRITE 11
#define LOAD 20
#define STORE 21
#define ADD 30
#define SUBTRACT 31
#define DIVIDE 32
#define MULTIPLY 33
#define BRANCH 40
#define BRANCHNEG 41
#define BRANCHZERO 42
#define HALT 43

/* function prototype */
void load( int *loadMemory );
void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
              int *opCodePtr, int *opPtr );
void dump( int *memory, int accumulator, int instructionCounter,
           int instructionRegister, int operationCode,
           int operand );
int validWord( int word );

int main()
{
   int memory[ SIZE ]; /* define memory array */
   int ac = 0;         /* accumulator */
   int ic = 0;         /* instruction counter */
   int opCode = 0;     /* operation code */
   int op = 0;         /* operand */
   int ir = 0;         /* instruction register */
   int i;              /* counter */

   /* clear memory */
   for ( i = 0; i < SIZE; i++ ) {
      memory[ i ] = 0;
   } /* end for */

   load( memory );
   execute( memory, &ac, &ic, &ir, &opCode, &op );
   dump( memory, ac, ic, ir, opCode, op );

   return 0; /* indicate successful termination */

} /* end main */

/* function loads instructions */
void load( int *loadMemory )
{
```

```
   int instruction;     /* current instruction */
   int i = 0;           /* indexing variable */
   char fileName[ 36 ]; /* input file name */
   FILE *finPtr;        /* input file pointer */

   /* prompt user for input file name */
   printf( "Enter input file: " );
   scanf( "%s", fileName );

   /* open input file */
   if ( ( finPtr = fopen( fileName, "r" ) ) == NULL ) {
      printf( "Data file was NOT opened.\n" );
   } /* end if */
   else { /* if file opened correctly */
      fscanf( finPtr, "%d", &instruction );

      /* while not end of file */
      while ( !feof( finPtr ) ) {

         /* check if instruction is valid */
         while ( !validWord( instruction ) ) {
            printf( "***DATA ERROR.\n" );
            printf( "***check instructions in data file.\n" );
            fscanf( finPtr, "%d", &instruction );
         } /* end while */

         /* load instruction and read next instruction */
         loadMemory[ i++ ] = instruction;
         fscanf( finPtr, "%d", &instruction );
      } /* end while */

   } /* end else */

   fclose( finPtr ); /* close file pointer */
} /* end function load */

/* carry out the commands */
void execute( int *memory, int *acPtr, int *icPtr, int *irPtr,
              int *opCodePtr, int *opPtr )
{
   int fatal = FALSE; /* fatal error flag */
   int temp;          /* temporary holding space */

   printf( "\n************START SIMPLETRON EXECUTION************\n\n" );

   /* separate operation code and operand */
   *irPtr = memory[ *icPtr ];
   *opCodePtr = *irPtr / 100;
   *opPtr = *irPtr % 100;

   /* loop while command is not HALT or fatal */
   while ( *opCodePtr != HALT && !fatal ) {

      /* determine appropriate action */
      switch ( *opCodePtr ) {

         /* read data into location in memory */
         case READ:
            printf( "Enter an integer: " );
            scanf( "%d", &temp );

            /* check for validity */
            while ( !validWord( temp ) ) {
               printf( "Number out of range. Please enter again: " );
               scanf( "%d", &temp );
            } /* end while */

            memory[ *opPtr ] = temp; /* write to memory */
```

```
               ++( *icPtr );
               break; /* exit switch */

            /* write data from memory to screen */
            case WRITE:
               printf( "Contents of %02d: %d\n", *opPtr, memory[ *opPtr ] );
               ++( *icPtr );
               break; /* exit switch */

            /* load data from memory into accumulator */
            case LOAD:
               *acPtr = memory[ *opPtr ];
               ++( *icPtr );
               break; /* exit switch */

            /* store data from accumulator into memory */
            case STORE:
               memory[ *opPtr ] = *acPtr;
               ++( *icPtr );
               break; /* exit switch */

            /* add data from memory to data in accumulator */
            case ADD:
               temp = *acPtr + memory[ *opPtr ];

               /* check validity */
               if ( !validWord( temp ) ) {
                  printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
                  printf( "*** Simpletron execution abnormally terminated ***\n" );
                  fatal = TRUE;
               } /* end if */
               else {
                  *acPtr = temp;
                  ++( *icPtr );
               } /* end else */

               break; /* exit switch */

            /* subtract data in memory from data in accumulator */
            case SUBTRACT:
               temp = *acPtr - memory[ *opPtr ];

               /* check validity */
               if ( !validWord( temp ) ) {
                  printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
                  printf( "*** Simpletron execution abnormally terminated ***\n" );
                  fatal = TRUE;
               } /* end if */
               else {
                  *acPtr = temp;
                  ++( *icPtr );
               } /* end else */

               break; /* exit switch */

            /* divide data in memory into data in accumulator */
            case DIVIDE:

               /* check for divide by zero error */
               if ( memory[ *opPtr ] == 0 ) {
                  printf( "*** FATAL ERROR: Attempt to divide by zero     ***\n" );
                  printf( "*** Simpletron execution abnormally terminated ***\n" );
                  fatal = TRUE;
               } /* end if */
               else {
                  *acPtr /= memory[ *opPtr ];
                  ++( *icPtr );
               } /* end else */
```

```

            break; /* exit switch */

         /* multiple data in memory by data in accumulator */
         case MULTIPLY:
            temp = *acPtr * memory[ *opPtr ];

            /* check validity */
            if ( !validWord( temp ) ) {
               printf( "*** FATAL ERROR: Accumulator overflow          ***\n" );
               printf( "*** Simpletron execution abnormally terminated ***\n" );
               fatal = TRUE;
            } /* end if */
            else {
               *acPtr = temp;
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         /* branch to specific location in memory */
         case BRANCH:
            *icPtr = *opPtr;
            break; /* exit switch */

         /* branch to location in memory if accumulator is negative */
         case BRANCHNEG:

            /* if accumulator is negative */
            if ( *acPtr < 0 ) {
               *icPtr = *opPtr;
            } /* end if */
            else {
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         /* branch to location in memory if accumulator is zero */
         case BRANCHZERO:

            /* if accumulator is zero */
            if ( *acPtr == 0 ) {
               *icPtr = *opPtr;
            } /* end if */
            else {
               ++( *icPtr );
            } /* end else */

            break; /* exit switch */

         default:
            printf( "*** FATAL ERROR: Invalid opcode detected       ***\n" );
            printf( "*** Simpletron execution abnormally terminated ***\n" );
            fatal = TRUE;
            break; /* exit switch */
      } /* end switch */

      /* separate next operation code and operand */
      *irPtr = memory[ *icPtr ];
      *opCodePtr = *irPtr / 100;
      *opPtr = *irPtr % 100;
   } /* end while */

   printf( "\n*************END SIMPLETRON EXECUTION*************\n" );
} /* end function execute */

```

```c
/* print out name and content of each register and memory */
void dump( const int *memory, int accumulator, int instructionCounter,
          int instructionRegister, int operationCode, int operand )
{
   int i;                 /* counter */
   char outputFile[ 36 ]; /* output file name */
   FILE *foutPtr;         /* output file pointer */

   /* prompt user for output file name */
   printf( "Enter output file name: " );
   scanf( "%s", outputFile );

   /* open output file for writing */
   if ( ( foutPtr = fopen( outputFile, "w" ) ) == NULL ) {
      printf( "Output file was not opened.\n" );
   } /* end if */
   else { /* if file opened correctly, print headers to file */
      fprintf( foutPtr, "\n%s\n%-23s%+05d\n%-23s%5.2d\n%-23s%+05d\n",
               "REGISTERS:", "accumulator", accumulator,
               "instructioncounter",instructionCounter,
               "instructionregister", instructionRegister );
      fprintf( foutPtr, "%-23s%5.2d\n%-23s%5.2d",
               "operationcode", operationCode, "operand", operand );
      fprintf( foutPtr, "\n\nMEMORY:\n   " );
   } /* end else */

   /* print headers to screen */
   printf( "\n%s\n%-23s%+05d\n%-23s%5.2d\n%-23s%+05d",
           "REGISTERS:", "accumulator", accumulator, "instructioncounter",
           instructionCounter, "instructionregister", instructionRegister);
   printf( "\n%-23s%5.2d\n%-23s%5.2d",
           "operationcode", operationCode, "operand", operand );
   printf( "\n\nMEMORY:\n   " );

   /* print column headers */
   for ( i = 0; i <= 9; i++ ) {
      printf( "%5d ", i );
      fprintf( foutPtr, "%5d ", i );
   } /* end for */

   /* print row headers and memory contents */
   for ( i = 0; i < SIZE; i++ ) {

      /* print in increments of 10 */
      if ( i % 10 == 0 ) {
         printf( "\n%2d ", i );
         fprintf( foutPtr, "\n%2d ", i );
      } /* end for */

      printf( "%+05d ", memory[ i ] );
      fprintf( foutPtr, "%+05d ", memory[ i ] );
   } /* end for */

   printf( "\n" );
   fprintf( foutPtr, "\n" );
   fclose( foutPtr ); /* close file pointer */
} /* end function dump */

/* function tests validity of word */
int validWord( int word )
{
   return word >= -9999 && word <= 9999;

} /* end function validWord */
```

```
Enter input file: simple.in

************START SIMPLETRON EXECUTION************

Enter an integer: 5
Enter an integer: 2
Contents of 09: 7

*************END SIMPLETRON EXECUTION*************
Enter output file name: simple.out

REGISTERS:
accumulator           +0007
instructioncounter       06
instructionregister   +4300
operationcode            43
operand                  00

MEMORY:
       0     1     2     3     4     5     6     7     8     9
 0 +1007 +1008 +2007 +3008 +2109 +1109 +4300 +0005 +0002 +0007
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
```

Contents of simple.in (a simple addition program)

```
1007
1008
2007
3008
2109
1109
4300
0000
0000
0000
```

12

Data Structures: Solutions

SOLUTIONS

12.6 Write a program that concatenates two linked lists of characters. The program should include function `concatenate` that takes pointers to both lists as arguments and concatenates the second list to the first list.

ANS:

```
/* Exercise 12.6 Solution */
#include <stdio.h>
#include <stdlib.h>

/* ListNode structure definition */
struct ListNode {
   char data;                 /* node data */
   struct ListNode *nextPtr; /* pointer to next node */
}; /* end struct ListNode */

typedef struct ListNode ListNode;
typedef ListNode *ListNodePtr;

/* function prototypes */
void concatenate( ListNodePtr a, ListNodePtr b );
void insert( ListNodePtr *sPtr, char value );
void printList( ListNodePtr currentPtr );

int main()
{
   ListNodePtr list1Ptr = NULL; /* pointer to first list */
   ListNodePtr list2Ptr = NULL; /* pointer to second list */
   char i; /* loop counter */

   /* assign letters from A to C into first list */
   for ( i = 'A'; i <= 'C'; i++ ) {
      insert( &list1Ptr, i );
   } /* end for */

   printf( "List 1 is: " );
```

```
   printList( list1Ptr );

   /* assign letters from D to F into second list */
   for ( i = 'D'; i <= 'F'; i++ ) {
      insert( &list2Ptr, i );
   } /* end for */

   printf( "List 2 is: " );
   printList( list2Ptr );

   concatenate( list1Ptr, list2Ptr );
   printf( "The concatenated list is: " );
   printList( list1Ptr );

   return 0; /* indicate successful termination */

} /* end main */

/* Concatenate two lists */
void concatenate( ListNodePtr a, ListNodePtr b )
{
   ListNodePtr currentPtr; /* temporary pointer */

   currentPtr = a; /* set currentPtr to first linked list */

   /* while currentPtr does not equal NULL */
   while( currentPtr->nextPtr != NULL ) {
      currentPtr = currentPtr->nextPtr;
   } /* end while */

   currentPtr->nextPtr = b; /* concatenate both lists */
} /* end function concatenate */

/* Insert a new value into the list in sorted order */
void insert( ListNodePtr *sPtr, char value )
{
   ListNodePtr newPtr;      /* new node */
   ListNodePtr previousPtr; /* previous node */
   ListNodePtr currentPtr;  /* current node */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      previousPtr = NULL;
      currentPtr = *sPtr; /* set currentPtr to start of list */

      /* loop to find correct location in list */
      while ( currentPtr != NULL && value > currentPtr->data ) {
         previousPtr = currentPtr;
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      /* insert at beginning of list */
      if ( previousPtr == NULL ) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      } /* end if */
```

```
      else { /* insert node between previousPtr and currentPtr */
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      } /* end else */

   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", value );
   } /* end else */

} /* end function insert */

/* Print the list */
void printList( ListNodePtr currentPtr )
{

   /* if list is empty */
   if ( !currentPtr ) {
      printf( "List is empty.\n\n" );
   } /* end if */
   else {

      /* loop while currentPtr does not equal NULL */
      while ( currentPtr ) {
         printf( "%c ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      printf( "*\n\n" );
   } /* end else */

} /* end function printList */
```

```
List 1 is: A B C *

List 2 is: D E F *

The concatenated list is: A B C D E F *
```

12.7 Write a program that merges two ordered lists of integers into a single ordered list of integers. Function `merge` should receive pointers to the first node of each of the lists to be merged and should return a pointer to the first node of the merged list.

ANS:

```c
/* Exercise 12.7 Solution */
#include <stdio.h>
#include <stdlib.h>

/* ListNode structure definition */
struct ListNode {
   int data;                 /* node data */
   struct ListNode *nextPtr; /* pointer to next node */
}; /* end struct ListNode */

typedef struct ListNode ListNode;
typedef ListNode *ListNodePtr;

/* function prototype */
void insert( ListNodePtr *sPtr, int value );
void printList( ListNodePtr currentPtr );
ListNodePtr merge( ListNodePtr a, ListNodePtr b );

int main()
{
   ListNodePtr list1Ptr = NULL; /* pointer to first list */
   ListNodePtr list2Ptr = NULL; /* pointer to second list */
   ListNodePtr list3Ptr;        /* pointer to merged list */
   int i; /* loop counter */

   /* build first list */
   for ( i = 2; i <= 10; i += 2 ) {
      insert( &list1Ptr, i );
   } /* end for */

   printf( "List 1 is: " );
   printList( list1Ptr );

   /* build second list */
   for ( i = 1; i <= 9; i += 2 ) {
      insert( &list2Ptr, i );
   } /* end for */

   printf( "List 2 is: " );
   printList( list2Ptr );

   /* merge both lists and print results */
   list3Ptr = merge( list1Ptr, list2Ptr );
   printf( "The merged list is: " );
   printList( list3Ptr );

   return 0; /* indicate successful termination */

} /* end main */

/* Merge two lists of integers */
ListNodePtr merge( ListNodePtr a, ListNodePtr b )
{
   ListNodePtr currentPtr1; /* pointer to first list */
   ListNodePtr currentPtr2; /* pointer to second list */
   ListNodePtr c = NULL;    /* pointer to merged list */

```

```
   currentPtr1 = a; /* set currentPtr1 to first linked list */
   currentPtr2 = b; /* set currentPtr2 to second linked list */

   /* while currentPtr1 does not equal NULL */
   while ( currentPtr1 != NULL ) {

      /* compare currentPtr1 and currentPtr2, insert lesser node */
      if ( currentPtr2 == NULL || currentPtr1->data <
         currentPtr2->data ) {

         /* insert currentPtr1 node */
         insert( &c, currentPtr1->data );
         currentPtr1 = currentPtr1->nextPtr;
      } /* end if */
      else {

         /* insert currentPtr2 node */
         insert( &c, currentPtr2->data );
         currentPtr2 = currentPtr2->nextPtr;
      } /* end else */

   } /* end while */

   /* insert any remaining nodes in currentPtr2 list */
   while ( currentPtr2 != NULL ) {
      insert( &c, currentPtr2->data );
      currentPtr2 = currentPtr2->nextPtr;
   } /* end while */

   return c; /* return merged list */

} /* end function merge */

/* Insert a new value into the list in sorted order */
void insert( ListNodePtr *sPtr, int value )
{
   ListNodePtr newPtr;      /* new node */
   ListNodePtr previousPtr; /* previous node */
   ListNodePtr currentPtr;  /* current node */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      previousPtr = NULL;
      currentPtr = *sPtr; /* set currentPtr to start of list */

      /* loop to find correct location in list */
      while ( currentPtr != NULL && value > currentPtr->data ) {
         previousPtr = currentPtr;
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      /* insert at beginning of list */
      if ( previousPtr == NULL ) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      } /* end if */
```

```
      else { /* insert node between previousPtr and currentPtr */
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      } /* end else */

   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", value );
   } /* end else */

} /* end function insert */

/* Print the list */
void printList( ListNodePtr currentPtr )
{

   /* if list is empty */
   if ( !currentPtr ) {
      printf( "List is empty.\n\n" );
   } /* end if */
   else {

      /* loop while currentPtr does not equal NULL */
      while ( currentPtr ) {
         printf( "%d ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      printf( "*\n\n" );
   } /* end else */

} /* end function printList */
```

```
List 1 is: 2 4 6 8 10 *

List 2 is: 1 3 5 7 9 *

The merged list is: 1 2 3 4 5 6 7 8 9 10 *
```

12.8 Write a program that inserts 25 random integers from 0 to 100 in order in a linked list. The program should calculate the sum of the elements and the floating-point average of the elements.

ANS: .

```
/* Exercise 12.8 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* ListNode structure definition */
typedef struct ListNode {
   int data;                 /* node data */
   struct ListNode *nextPtr; /* pointer to next node */
} ListNode; /* end struct ListNode */

typedef ListNode *ListNodePtr;

/* function prototypes */
int sumList( ListNodePtr a );
double averageList( ListNodePtr a );
void insert( ListNodePtr *sPtr, int value );
void printList( ListNodePtr currentPtr );

int main()
{
   ListNodePtr listPtr = NULL; /* list pointer */
   int i; /* loop counter */

   srand( time( NULL ) ); /* randomize */

   /* build list with random numbers from 0 to 100 */
   for ( i = 1; i <= 25; i++ ) {
      insert( &listPtr, rand() % 101 );
   } /* end for */

   printf( "The list is:\n" );
   printList( listPtr );

   /* calculate and display the sum and average of list values */
   printf( "The sum is %d\n", sumList( listPtr ) );
   printf( "The average is %f\n", averageList( listPtr ) );

   return 0; /* indicate successful termination */

} /* end main */

/* Sum the integers in a list */
int sumList( ListNodePtr a )
{
   ListNodePtr currentPtr; /* temporary pointer to list a */
   int total = 0;          /* sum of node values */

   currentPtr = a; /* set currentPtr to list a */

   /* loop through list */
   while ( currentPtr != NULL ) {

      /* add node value to total */
      total += currentPtr->data;
      currentPtr = currentPtr->nextPtr;
   } /* end while */
```

```c

   return total;

} /* end function sumList */

/* Average the integers in a list */
double averageList( ListNodePtr a )
{
   ListNodePtr currentPtr; /* temporary pointer to list a */
   double total = 0.0;     /* sum of node values */
   int count = 0;          /* number of nodes in list */

   currentPtr = a; /* set currentPtr to list a */

   /* loop through list */
   while ( currentPtr != NULL ) {
      ++count; /* increment count */
      total += currentPtr->data; /* update total */
      currentPtr = currentPtr->nextPtr;
   } /* end while */

   return total / count; /* return average */

} /* end function averageList */

/* Insert a new value into the list in sorted order */
void insert( ListNodePtr *sPtr, int value )
{
   ListNodePtr newPtr;      /* new node */
   ListNodePtr previousPtr; /* previous node */
   ListNodePtr currentPtr;  /* current node */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      previousPtr = NULL;
      currentPtr = *sPtr; /* set currentPtr to start of list */

      /* loop to find correct location in list */
      while ( currentPtr != NULL && value > currentPtr->data ) {
         previousPtr = currentPtr;
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      /* insert at beginning of list */
      if ( previousPtr == NULL ) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      } /* end if */
      else { /* insert node between previousPtr and currentPtr */
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      } /* end else */

   } /* end if */
   else {
```

```
         printf( "%c not inserted. No memory available.\n", value );
      } /* end else */

   } /* end function insert */

   /* Print the list */
   void printList( ListNodePtr currentPtr )
   {

      /* if list is empty */
      if ( !currentPtr ) {
         printf( "List is empty.\n\n" );
      } /* end if */
      else {

         /* loop while currentPtr does not equal NULL */
         while ( currentPtr ) {
            printf( "%d ", currentPtr->data );
            currentPtr = currentPtr->nextPtr;
         } /* end while */

         printf( "*\n\n" );
      } /* end else */

   } /* end function printList */
```

```
The list is:
6 12 14 20 27 31 31 34 37 38 56 59 63 66 72 73 73 76 77 79 88 94 95 96 97 *

The sum is 1414
The average is 56.560000
```

12.9 Write a program that creates a linked list of 10 characters, then creates a copy of the list in reverse order.
ANS:

```
/* Exercise 12.9 Solution */
#include <stdio.h>
#include <stdlib.h>

/* ListNode structure definition */
struct ListNode {
   char data;                /* node data */
   struct ListNode *nextPtr; /* pointer to next node */
}; /* end struct ListNode */

typedef struct ListNode ListNode;
typedef ListNode *ListNodePtr;

/* function prototypes */
ListNodePtr reverseList( ListNodePtr currentPtr );
void insert( ListNodePtr *sPtr, char value );
void printList( ListNodePtr currentPtr );
void push( ListNodePtr *topPtr, char info );

int main()
{
   ListNodePtr listPtr = NULL; /* list pointer */
   char i; /* loop counter */

   /* build list with characters A to J */
   for ( i = 'A'; i <= 'J'; i++ ) {
      insert( &listPtr, i );
   } /* end for */

   printf( "The list is:\n" );
   printList( listPtr );

   /* reverse the list and display result */
   printf( "The list in reverse is:\n" );
   printList( reverseList( listPtr ) );

   return 0; /* indicate successful termination */

} /* end main */

/* Create a list in the reverse order of the list argument */
ListNodePtr reverseList( ListNodePtr currentPtr )
{
   ListNodePtr stack = NULL; /* pointer to reversed list */

   /* loop through list currentPtr */
   while ( currentPtr != NULL ) {

      /* push current element on to stack */
      push( &stack, currentPtr->data );
      currentPtr = currentPtr->nextPtr;
   } /* end while */

   return stack; /* return reversed list */

} /* end function reverseList */

```

```c
/* Insert a new value into the list in sorted order */
void insert( ListNodePtr *sPtr, char value )
{
   ListNodePtr newPtr;      /* new node */
   ListNodePtr previousPtr; /* previous node */
   ListNodePtr currentPtr;  /* current node */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      previousPtr = NULL;
      currentPtr = *sPtr; /* set currentPtr to start of list */

      /* loop to find correct location in list */
      while ( currentPtr != NULL && value > currentPtr->data ) {
         previousPtr = currentPtr;
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      /* insert at beginning of list */
      if ( previousPtr == NULL ) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      } /* end if */
      else { /* insert node between previousPtr and currentPtr */
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      } /* end else */

   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", value );
   } /* end else */

} /* end function insert */

/* Insert a node at the stack top */
void push( ListNodePtr *topPtr, char info )
{
   ListNodePtr newPtr; /* temporary node pointer */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if memory was allocated, insert node at top of list */
   if ( newPtr ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", info );
   } /* end else */

} /* end function push */
```

```
/* Print the list */
void printList( ListNodePtr currentPtr )
{

   /* if list is empty */
   if ( !currentPtr ) {
      printf( "List is empty.\n\n" );
   } /* end if */
   else {

      /* loop while currentPtr does not equal NULL */
      while ( currentPtr ) {
         printf( "%c ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      printf( "*\n\n" );
   } /* end else */

} /* end function printList */
```

```
The list is:
A B C D E F G H I J *

The list in reverse is:
J I H G F E D C B A *
```

12.10 Write a program that inputs a line of text and uses a stack to print the line reversed.
ANS:

```
/* Exercise 12.10 Solution */
#include <stdio.h>
#include <stdlib.h>

/* stackNode structure definition */
struct stackNode {
   char data;                /* node data */
   struct stackNode *nextPtr; /* pointer to next node */
}; /* end struct stackNode */

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;

/* function prototypes */
void push( StackNodePtr *topPtr, char info );
char pop( StackNodePtr *topPtr );
int isEmpty( StackNodePtr topPtr );

int main()
{
   StackNodePtr stackPtr = NULL; /* points to the stack top */
   char c; /* current character from text */

   printf( "Enter a line of text:\n" );

   /* read each letter with getchar and push on stack */
   while ( ( c = getchar() ) != '\n' ) {
      push( &stackPtr, c );
   } /* end while */

   printf( "\nThe line is reverse is:\n" );

   /* while the stack is not empty, pop next character */
   while ( !isEmpty( stackPtr ) ) {
      printf( "%c", pop( &stackPtr ) );
   } /* end while */

   return 0; /* indicate successful termination */

} /* end main */

/* Insert a node at the stack top */
void push( StackNodePtr *topPtr, char info )
{
   StackNodePtr newPtr; /* temporary node pointer */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( StackNode ) );

   /* if memory was allocated, insert node at top of stack */
   if ( newPtr ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   } /* end if */
   else {
      printf( "%d not inserted. No memory available.\n", info );
   } /* end else */
```

```
} /* end function push */

/* Remove a node from the stack top */
char pop( StackNodePtr *topPtr )
{
   StackNodePtr tempPtr; /* temporary node pointer */
   int popValue;         /* value of popped node */

   tempPtr = *topPtr;
   popValue = ( *topPtr )->data;
   *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
   free( tempPtr ); /* free memory */

   return popValue; /* return value of popped node */

} /* end function pop */

/* Is the stack empty? */
int isEmpty( StackNodePtr topPtr )
{
   return !topPtr; /* return NULL if stack is empty */

} /* end function isEmpty */
```

```
Enter a line of text:
this is a line of text

The line is reverse is:
txet fo enil a si siht
```

12.11 Write a program that uses a stack to determine if a string is a palindrome (i.e., the string is spelled identically backward and forward). The program should ignore spaces and punctuation.

ANS:

```
/* Exercise 12.11 Solution  */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define YES 1
#define NO 0

/* stackNode structure definition */
struct stackNode {
   char data;                 /* node data */
   struct stackNode *nextPtr; /* pointer to next node */
}; /* end struct stackNode */

typedef struct stackNode STACKNODE;
typedef STACKNODE *STACKNODEPTR;

/* function prototypes */
void push( STACKNODEPTR *topPtr, char info );
char pop( STACKNODEPTR *topPtr );
int isEmpty( STACKNODEPTR topPtr );

int main()
{
   STACKNODEPTR stackPtr = NULL; /* points to the stack top */
   char c;                       /* current character from text */
   char line[ 50 ];              /* text from user */
   char condensedLine[ 50 ];     /* text with only letters */
   int i = 0;                    /* length of condensed line */
   int j = 0;                    /* length of line */
   int palindrome = YES;         /* result of palindrome test */

   printf( "Enter a line of text:\n" );

   /* read each letter with getchar and add to line */
   while ( ( c = getchar() ) != '\n' ) {
      line[ j++ ] = c;

      /* remove all spaces and punctuation */
      if ( isalpha( c ) ) {
         condensedLine[ i++ ] = tolower( c );
         push( &stackPtr, tolower( c ) );
      } /* end if */

   } /* end while */

   line[ j ] = '\0';

   /* loop through condensedLine */
   for ( j = 0; j < i; j++ ) {

      /* if condensedLine does not equal stack */
      if ( condensedLine[ j ] != pop( &stackPtr ) ) {
         palindrome = NO;
         break; /* exit loop */
      } /* end if */

   } /* end for */
```

```

   /* if text is a palindrome */
   if ( palindrome ) {
      printf( "\"%s\" is a palindrome\n", line );
   } /* end if */
   else {
      printf( "\"%s\" is not a palindrome\n", line );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */

/* Insert a node at the stack top */
void push( STACKNODEPTR *topPtr, char info )
{
   STACKNODEPTR newPtr; /* temporary node pointer */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( STACKNODE ) );

   /* if memory was allocated, insert node at top of stack */
   if ( newPtr ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   } /* end if */
   else {
      printf( "%d not inserted. No memory available.\n", info );
   } /* end else */

} /* end function push */

/* Remove a node from the stack top */
char pop( STACKNODEPTR *topPtr )
{
   STACKNODEPTR tempPtr; /* temporary node pointer */
   int popValue;         /* value of popped node */

   tempPtr = *topPtr;
   popValue = ( *topPtr )->data;
   *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
   free( tempPtr ); /* free memory */

   return popValue; /* return value of popped node */

} /* end function pop */

/* Is the stack empty? */
int isEmpty( STACKNODEPTR topPtr )
{
   return !topPtr; /* return NULL if stack is empty */

} /* end function isEmpty */
```

```
Enter a line of text:
able was i ere i saw elba
"able was i ere i saw elba" is a palindrome
```

```
Enter a line of text:
this is not a palindrome
"this is not a palindrome" is not a palindrome
```

12.12 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 in support of its operation, and in each the stack is used for a different purpose.

In this exercise, you will write a version of the infix-to-postfix conversion algorithm. In the next exercise, you will write a version of the postfix expression evaluation algorithm.

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
- `%` remainder

The stack should be maintained with the following declarations:

```
struct stackNode {
   char data;
   struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

The program should consist of `main` and eight other functions with the following function headers:

```
void convertToPostfix( char infix[], char postfix[] )
```

Convert the infix expression to postfix notation.

`int isOperator( char c )`

Determine if `c` is an operator.

`int precedence( char operator1, char operator2 )`

Determine 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.

`void push( StackNodePtr *topPtr, char value )`

Push a value on the stack.

`char pop( StackNodePtr *topPtr )`

Pop a value off the stack.

`char stackTop( StackNodePtr topPtr )`

Return the top value of the stack without popping the stack.

`int isEmpty( StackNodePtr topPtr )`

Determine if the stack is empty.

`void printStack( StackNodePtr topPtr )`

Print the stack.

ANS:

```
/* Exercise 12.12 Solution */
/* Infix to postfix conversion */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#define MAXCOLS 100

/* stackNode structure definition */
typedef struct stackNode {
   char data;                 /* node data */
   struct stackNode *nextPtr; /* pointer to next node */
} STACKNODE; /* end struct stackNode */

typedef STACKNODE *STACKNODEPTR;

/* function prototypes */
void convertToPostfix( char inFix[], char postFix[] );
int isOperator( char c );
int precedence( char operator1, char operator2 );
void push( STACKNODEPTR *topPtr, char info );
char pop( STACKNODEPTR *topPtr );
char stackTop( STACKNODEPTR topPtr );
int isEmpty( STACKNODEPTR topPtr );
void printStack( STACKNODEPTR currentPtr );

int main()
{
   char c;                  /* current character from expression */
   char inFix[ MAXCOLS ];   /* expression in infix notation */
```

```
   char postFix[ MAXCOLS ]; /* expression in postfix notation */
   int pos = 0;             /* indexing variable */

   printf( "Enter the infix expression.\n" );

   /* read each character with getchar */
   while ( ( c = getchar() ) != '\n' ) {

      /* remove any spaces */
      if ( c != ' ' ) {
         inFix[ pos++ ] = c;
      } /* end if */

   } /* end while */

   inFix[ pos ] = '\0';

   /* print infix expression, convert to postfix and print */
   printf( "%s\n%s\n", "The original infix expression is:", inFix );
   convertToPostfix( inFix, postFix );
   printf( "The expression in postfix notation is:\n%s\n", postFix );

   return 0; /* indicate successful termination */

} /* end main */

/* convert infix expression to postfix notation */
void convertToPostfix( char inFix[], char postFix[] )
{
   STACKNODEPTR stackPtr = NULL; /* points to the stack top */
   int i;                        /* loop counter */
   int j;                        /* indexing variable */
   int higher;                   /* operator flag */
   char popValue;                /* value of popped node */

   /* push left parenthesis onto the stack */
   push( &stackPtr, '(' );
   printStack( stackPtr );

   /* add a right parenthesis to infix */
   strcat( inFix, " )" );

   /* convert the infix expression to postfix */
   for ( i = 0, j = 0; stackTop( stackPtr ); i++ ) {

      /* if current character is a digit */
      if ( isdigit( inFix[ i ] ) ) {
         postFix[ j++ ] = inFix[ i ];
      } /* end if */

      /* if character is left parenthesis, push on stack */
      else if ( inFix[ i ] == '(' ) {
         push( &stackPtr, '(' );
         printStack( stackPtr );
      } /* end else if */

      /* if character is an operator */
      else if ( isOperator( inFix[ i ] ) ) {
         higher = 1; /* used to store value of precedence test */

```

```
         /* loop while current operator does not have
            the highest precedence */
         while ( higher ) {

            /* if the top of the stack is an operator */
            if ( isOperator( stackTop( stackPtr ) ) ) {

               /* compare precedence of operators */
               if ( precedence( stackTop( stackPtr ), inFix[ i ] ) ) {
                  postFix[ j++ ] = pop( &stackPtr );
                  printStack( stackPtr );
               } /* end if */
               else  {
                  higher = 0; /* reset flag */
               } /* end else */

            } /* end if */
            else {
               higher = 0; /* reset flag */
            } /* end else */

         } /* end while */

         push( &stackPtr, inFix[ i ] );
         printStack( stackPtr );
      } /* end else if */

      /* if character is a right parenthesis */
      else if ( inFix[ i ] == ')' ) {

         /* pop stack until popped value is a left parenthesis */
         while ( ( popValue = pop( &stackPtr ) ) != '(' ) {
            printStack( stackPtr );
            postFix[ j++ ] = popValue;
         } /* end while */

         printStack( stackPtr );
      } /* end else if */

   } /* end for */

   postFix[ j ] = '\0';
} /* end function convertToPostfix */

/* check if c is an operator */
int isOperator( char c )
{

   /* if c is an operator return true */
   if ( c == '+' || c == '-' || c == '*' || c == '/' || c == '^' ) {
      return 1;

   } /* end if */
   else { /* return false */
      return 0;

   } /* end else */

} /* end function isOperator */

```

```
/* if the precedence of operator1 is >= operator2,
   return 1 ( true ), else return 0 ( false ) */
int precedence( char operator1, char operator2 )
{

   /* compare precedence of operator1 and operator2 */
   if ( operator1 == '^' ) {
      return 1;
   } /* end if */
   else if ( operator2 == '^' ) {
      return 0;
   } /* end else if */
   else if ( operator1 == '*' || operator1 == '/' ) {
      return 1;
   } /* end else if */
   else if ( operator1 == '+' || operator1 == '-' ) {

      /* if operator2 is * or / than return true */
      if ( operator2 == '*' || operator2 == '/' ) {
         return 0;
      } /* end if */
      else {
         return 1;
      } /* end else */

   } /* end else if */

   return 0; /* default */

} /* end function precedence */

/* Insert a node at the stack top */
void push( STACKNODEPTR *topPtr, char info )
{
   STACKNODEPTR newPtr; /* temporary node pointer */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( STACKNODE ) );

   /* if memory was allocated, insert node at top of stack */
   if ( newPtr ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", info );
   } /* end else */

} /* end function push */

/* Remove a node from the stack top */
char pop( STACKNODEPTR *topPtr )
{
   STACKNODEPTR tempPtr; /* temporary node pointer */
   char popValue;        /* value of popped node */

   tempPtr = *topPtr;
   popValue = ( *topPtr )->data;
   *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
   free( tempPtr ); /* free memory */
```

```
   return popValue; /* return value of popped node */

} /* end function pop */

/* View the top element of the stack */
char stackTop( STACKNODEPTR topPtr )
{

   /* if the stack is not empty */
   if ( !isEmpty( topPtr ) ) {
      return topPtr->data;
   } /* end if */
   else {
      return 0;
   } /* end else */

} /* end function stackTop */

/* Is the stack empty? */
int isEmpty( STACKNODEPTR topPtr )
{
   return !topPtr; /* return NULL if stack is empty */

} /* end function isEmpty */

/* Print the stack */
void printStack( STACKNODEPTR currentPtr )
{

   /* if the stack is empty */
   if ( currentPtr == NULL ) {
      printf( "The stack is empty.\n\n" );
   } /* end if */
   else { /* print stack */

      /* loop through stack */
      while ( currentPtr != NULL ) {
         printf( "%c   ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      printf( "NULL\n" );
   } /* end else */

} /* end function printStack */
```

```
Enter the infix expression.
1+(2*3-(4/5^6)*7)*8
The original infix expression is:
1+(2*3-(4/5^6)*7)*8
(   NULL
+   (   NULL
(   +   (   NULL
*   (   +   (   NULL
(   +   (   NULL
-   (   +   (   NULL
(   -   (   +   (   NULL
/   (   -   (   +   (   NULL
^   /   (   -   (   +   (   NULL
/   (   -   (   +   (   NULL
(   -   (   +   (   NULL
-   (   +   (   NULL
*   -   (   +   (   NULL
-   (   +   (   NULL
(   +   (   NULL
+   (   NULL
*   +   (   NULL
+   (   NULL
(   NULL
The stack is empty.

The expression in postfix notation is:
123*456^/7*-8*+
```

12.13 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 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 on the stack. This note also applies to operator `'-'`.] The arithmetic operations allowed in an expression are:

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `^` exponentiation
- `%` remainder]

The stack should be maintained with the following declarations:

```
struct stackNode {
   int data;
   struct stackNode *nextPtr;
};

typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
```

The program should consist of `main` and six other functions with the following function headers:

```
int evaluatePostfixExpression( char *expr )
```

Evaluate the postfix expression.

```
int calculate( int op1, int op2, char operator )
```

Evaluate the expression `op1 operator op2`.

```
void push( StackNodePtr *topPtr, int value )
```

Push a value on the stack.

```
int pop( StackNodePtr *topPtr )
```

Pop a value off the stack.

```
int isEmpty( StackNodePtr topPtr )
```

Determine if the stack is empty.

```
void printStack( StackNodePtr topPtr )
```

Print the stack.

ANS:

```
/* Exercise 12.13 Solution */
/* Using a stack to evaluate an expression in postfix notation */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

/* StackNode structure definition */
struct StackNode {
   int data;                  /* node data */
   struct StackNode *nextPtr; /* pointer to next node */
}; /* end struct StackNode */

typedef struct StackNode StackNode;
typedef StackNode *StackNodePtr;

/* function prototypes */
int evaluatePostfixExpression( char *expr );
int calculate( int op1, int op2, char operator );
void push( StackNodePtr *topPtr, int info );
int pop( StackNodePtr *topPtr );
int isEmpty( StackNodePtr topPtr );
void printStack( StackNodePtr currentPtr );

int main()
{
   char expression[ 100 ]; /* postfix expression */
   char c;                 /* current character from expression */
   int answer;             /* expression answer */
   int i = 0;              /* indexing variable */

   printf( "Enter a postfix expression:\n" );

   /* read each character with getchar */
   while ( ( c = getchar() ) != '\n' ) {

      /* remove any spaces */
      if ( c != ' ' ) {
         expression[ i++ ] = c;
      } /* end if */

   } /* end while */

   expression[ i ] = '\0';

   /* calculate answer and print result */
   answer = evaluatePostfixExpression( expression );
   printf( "The value of the expression is: %d\n", answer );

   return 0; /* indicate successful termination */

} /* end main */

/* evaluate the postfix expression */
int evaluatePostfixExpression( char *expr )
{
   int i;                          /* loop counter */
```

```
   int popVal1;                  /* right value of current operation */
   int popVal2;                  /* left value of current operation */
   StackNodePtr stackPtr = NULL; /* points to the stack top */
   char c;                       /* current character */

   /* loop through expression */
   for ( i = 0; ( c = expr[ i ] ) != '\0'; i++ ) {

      /* if character is a digit, push it on stack */
      if ( isdigit( c ) ) {
         push( &stackPtr, c - '0' );
         printStack( stackPtr );
      } /* end if */
      else { /* calculate current operation */
         popVal2 = pop( &stackPtr );
         printStack( stackPtr );
         popVal1 = pop( &stackPtr );
         printStack( stackPtr );

         /* calculate answer and push on stack */
         push( &stackPtr, calculate( popVal1, popVal2, c ) );
         printStack( stackPtr );
      } /* end else */

   } /* end for */

   return pop( &stackPtr ); /* return final answer */

} /* end function evaluatePostfixExpression */

/* evaluate the expression op1 operator op2 */
int calculate( int op1, int op2, char operator )
{

   /* use correct operator to calculate answer */
   switch( operator ) {

      case '+': /* addition */
         return op1 + op2;

      case '-': /* subtraction */
         return op1 - op2;

      case '*': /* multiplication */
         return op1 * op2;

      case '/': /* division */
         return op1 / op2;

      case '^': /* exponentiation */
         return pow( op1, op2 );
   } /* end switch */

   return 0; /* default */

} /* end function calculate */

/* Insert a node at the stack top */
void push( StackNodePtr *topPtr, int info )
{
   StackNodePtr newPtr; /* temporary node pointer */
```

```c

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( StackNode ) );

   /* if memory was allocated, insert node at top of stack */
   if ( newPtr ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   } /* end if */
   else {
      printf( "%d not inserted. No memory available.\n", info );
   } /* end else */

} /* end function push */

/* Remove a node from the stack top */
int pop( StackNodePtr *topPtr )
{
   StackNodePtr tempPtr; /* temporary node pointer */
   int popValue;         /* value of popped node */

   tempPtr = *topPtr;
   popValue = ( *topPtr )->data;
   *topPtr = ( *topPtr )->nextPtr; /* reset topPtr */
   free( tempPtr ); /* free memory */

   return popValue; /* return value of popped node */

} /* end function pop */

/* Is the stack empty? */
int isEmpty( StackNodePtr topPtr )
{
   return !topPtr; /* return NULL if stack is empty */

} /* end function isEmpty */

/* Print the stack */
void printStack( StackNodePtr currentPtr )
{

   /* loop through stack */
   while ( currentPtr != NULL ) {
      printf( "%d   ", currentPtr->data );
      currentPtr = currentPtr->nextPtr;
   } /* end while */

   printf( "NULL\n" );
} /* end function printStack */
```

```
Enter a postfix expression:
123*456^/7*-8*+
1   NULL
2   1   NULL
3   2   1   NULL
2   1   NULL
1   NULL
6   1   NULL
4   6   1   NULL
5   4   6   1   NULL
6   5   4   6   1   NULL
5   4   6   1   NULL
4   6   1   NULL
15625   4   6   1   NULL
4   6   1   NULL
6   1   NULL
0   6   1   NULL
7   0   6   1   NULL
0   6   1   NULL
6   1   NULL
0   6   1   NULL
6   1   NULL
1   NULL
6   1   NULL
8   6   1   NULL
6   1   NULL
1   NULL
48   1   NULL
1   NULL
NULL
49   NULL
The value of the expression is: 49
```

12.14 Modify the postfix evaluator program of Exercise 12.13 so that it can process integer operands larger than 9.

12.15 *(Supermarket Simulation)* Write a program that simulates a check-out line at a supermarket. The line is a queue. Customers arrive in random integer intervals of 1 to 4 minutes. Also, each customer is serviced 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:

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 experienced?
c) What happens if the arrival interval is changed from 1 to 4 minutes to 1 to 3 minutes?

12.16 Modify the program of Fig. 12.19 to allow the binary tree to contain duplicate values.
ANS:

```
/* Exercise 12.16 Solution */
/* This is a modification of figure 12.19 */
/* Only function insertNode has been modified */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* TreeNode structure definition */
struct TreeNode {
   struct TreeNode *leftPtr;  /* pointer to left subtree */
   int data;                  /* node data */
   struct TreeNode *rightPtr; /* pointer to right subtree */
}; /* end struct TreeNode */

typedef struct TreeNode TreeNode;
typedef TreeNode *TreeNodePtr;

/* function prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
void inOrder( TreeNodePtr treePtr );
void preOrder( TreeNodePtr treePtr );
void postOrder( TreeNodePtr treePtr );

int main()
{
   int i;                      /* loop counter */
   int item;                   /* random value to insert in tree */
   TreeNodePtr rootPtr = NULL; /* points to the tree root */

   srand( time( NULL ) ); /* randomize */
   printf( "The numbers being placed in the tree are:\n" );

   /* insert random values between 1 and 15 in the tree */
   for ( i = 1; i <= 10; i++ ) {
      item = rand() % 15;
      printf( "%3d", item );
      insertNode( &rootPtr, item );
   } /* end for */

   /* traverse the tree preorder */
   printf( "\n\nThe preorder traversal is:\n" );
   preOrder( rootPtr );

   /* traverse the tree inorder */
   printf( "\n\nThe inorder traversal is:\n" );
   inOrder( rootPtr );

   /* traverse the tree postorder */
   printf( "\n\nThe postorder traversal is:\n" );
   postOrder( rootPtr );

   return 0; /* indicate successful termination */

} /* end main */

/* insert a node into the tree */
void insertNode( TreeNodePtr *treePtr, int value )
{
```

```
   /* if treePtr is NULL */
   if ( !*treePtr ) {

      /* dynamically allocate memory */
      *treePtr = malloc( sizeof( TreeNode ) );

      /* if memory was allocated, insert node */
      if ( *treePtr ) {
         ( *treePtr )->data = value;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      } /* end if */
      else {
         printf( "%d not inserted. No memory available.\n", value );
      } /* end else */

      return;
   } /* end if */
   else { /* recursively call insertNode */

      /* insert node in left subtree */
      if ( value <= ( *treePtr )->data ) {
         insertNode( &( ( *treePtr )->leftPtr ), value );
      } /* end if */
      else { /* insert node in right subtree */
         insertNode( &( ( *treePtr )->rightPtr ), value );
      } /* end else */

   } /* end else */

} /* end function insertNode */

/* traverse the tree inorder */
void inOrder( TreeNodePtr treePtr )
{

   /* traverse left subtree, print node, traverse right subtree */
   if ( treePtr ) {
      inOrder( treePtr->leftPtr );
      printf( "%3d", treePtr->data );
      inOrder( treePtr->rightPtr );
   } /* end if */

} /* end function inOrder */

/* traverse the tree preorder */
void preOrder( TreeNodePtr treePtr )
{

   /* print node, traverse left subtree, traverse right subtree */
   if ( treePtr ) {
      printf( "%3d", treePtr->data );
      preOrder( treePtr->leftPtr );
      preOrder( treePtr->rightPtr );
   } /* end if */

} /* end function preOrder */

```

```
/* traverse the tree postorder */
void postOrder( TreeNodePtr treePtr )
{

   /* traverse left subtree, traverse right subtree, print node */
   if ( treePtr ) {
      postOrder( treePtr->leftPtr );
      postOrder( treePtr->rightPtr );
      printf( "%3d", treePtr->data );
   } /* end if */

} /* end function postOrder */
```

```
The numbers being placed in the tree are:
 14  3  7  7  3 12  7 11  1  2

The preorder traversal is:
 14  3  3  1  2  7  7  7 12 11

The inorder traversal is:
  1  2  3  3  7  7  7 11 12 14

The postorder traversal is:
  2  1  3  7  7 11 12  7  3 14
```

12.17 Write a program based on the program of Fig. 12.19 that inputs a line of text, tokenizes the sentence into separate words, inserts the words in a binary search tree, and prints the inorder, preorder, and postorder traversals of the tree.

[*Hint:* Read the line of text into an array. Use `strtok` to tokenize the text. When a token is found, create a new node for the tree, assign the pointer returned by `strtok` to member `string` of the new node, and insert the node in the tree.]

ANS:

```
/* Exercise 12.17 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* TreeNode structure definition */
struct TreeNode {
   struct TreeNode *leftPtr;  /* pointer to left subtree */
   char *token;               /* node data */
   struct TreeNode *rightPtr; /* pointer to right subtree */
}; /* end struct TreeNode */

typedef struct TreeNode TreeNode;
typedef TreeNode *TreeNodePtr;

/* function prototypes */
void insertNode( TreeNodePtr *treePtr, char *tokenPtr );
void inOrder( TreeNodePtr treePtr );
void preOrder( TreeNodePtr treePtr );
void postOrder( TreeNodePtr treePtr );

int main()
{
   TreeNodePtr rootPtr = NULL; /* points to the tree root */
   char sentence[ 80 ];        /* text from user */
   char *tokenPtr;             /* pointer to current token */

   /* prompt user and read a sentence */
   printf( "Enter a sentence:\n" );
   gets( sentence );

   /* tokenize the sentence */
   tokenPtr = strtok( sentence, " " );

   /* insert the tokens in the tree */
   while ( tokenPtr ) {
      insertNode( &rootPtr, tokenPtr );
      tokenPtr = strtok( NULL, " " );
   } /* end while */

   /* traverse the tree preorder */
   printf( "\nThe preorder traversal is:\n" );
   preOrder( rootPtr );

   /* traverse the tree inorder */
   printf( "\n\nThe inorder traversal is:\n" );
   inOrder( rootPtr );

   /* traverse the tree postorder */
   printf( "\n\nThe postorder traversal is:\n" );
   postOrder( rootPtr );

   return 0; /* indicate successful termination */

} /* end main */
```

```c

/* insert a node into the tree */
void insertNode( TreeNodePtr *treePtr, char *tokenPtr )
{

   /* if treePtr is NULL */
   if ( !*treePtr ) {

      /* dynamically allocate memory */
      *treePtr = malloc( sizeof( TreeNode ) );

      /* if memory was allocated, insert node */
      if ( *treePtr ) {
         ( *treePtr )->token = tokenPtr;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      } /* end if */
      else {
         printf( "\"%s\" not inserted. No memory available.\n",
            tokenPtr );
      } /* end else */

      return;
   } /* end if */
   else  { /* recursively call insertNode */

      /* insert node in left subtree */
      if ( strcmp( tokenPtr, ( *treePtr )->token ) <= 0 ) {
         insertNode( &( ( *treePtr )->leftPtr ), tokenPtr );
      } /* end if */
      else { /* insert node in right subtree */
         insertNode( &( ( *treePtr )->rightPtr ), tokenPtr );
      } /* end else */

   } /* end else */

} /* end function insertNode */

/* traverse the tree inorder */
void inOrder( TreeNodePtr treePtr )
{

   /* traverse left subtree, print node, traverse right subtree */
   if ( treePtr ) {
      inOrder( treePtr->leftPtr );
      printf( "%s ", treePtr->token );
      inOrder( treePtr->rightPtr );
   } /* end if */

} /* end function inOrder */

/* traverse the tree preorder */
void preOrder( TreeNodePtr treePtr )
{

   /* print node, traverse left subtree, traverse right subtree */
   if ( treePtr ) {
      printf( "%s ", treePtr->token );
      preOrder( treePtr->leftPtr );
      preOrder( treePtr->rightPtr );
   } /* end if */
```

```

} /* end function preOrder */

/* traverse the tree postorder */
void postOrder( TreeNodePtr treePtr )
{

   /* traverse left subtree, traverse right subtree, print node */
   if ( treePtr ) {
      postOrder( treePtr->leftPtr );
      postOrder( treePtr->rightPtr );
      printf( "%s ", treePtr->token );
   } /* end if */

} /* end function postOrder */
```

```
Enter a sentence:
this program inserts strings of different lengths in a tree

The preorder traversal is:
this program inserts different a in of lengths strings tree

The inorder traversal is:
a different in inserts lengths of program strings this tree

The postorder traversal is:
a in different lengths of inserts strings program tree this
```

12.18 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.

ANS: Using a single subscripted array, it is necessary to compare each value to be inserted in the array with all the array elements until a match is found or until it is determined that there is not a duplicate value in the array. If there is not a duplicate, the value can be inserted in the array. On average, half the array elements must be searched when the value is a duplicate and all the array elements must be searched when the value is not a duplicate. The binary search tree only compares the value to be inserted with the values in its path down the tree. If a leaf node is reached, and the value does not match the value in the leaf node, the value can be inserted. Otherwise the value can be discarded.

12.19 Write a function `depth` that receives a binary tree and determines how many levels it has.

12.20 (*Recursively Print a List Backwards*) Write a function `printListBackwards` that recursively outputs the items in a list in reverse order. Use your function in a test program that creates a sorted list of integers and prints the list in reverse order.

ANS:

```
/* Exercise 12.20 Solution */
#include <stdio.h>
#include <stdlib.h>

/* ListNode structure definition */
struct ListNode {
   int data;                 /* node data */
   struct ListNode *nextPtr; /* pointer to next node */
}; /* end struct ListNode */

typedef struct ListNode ListNode;
typedef ListNode *ListNodePtr;

/* function prototype */
void printList( ListNodePtr currentPtr );
```

```
void printListBackwards( ListNodePtr currentPtr );
void insertItem( ListNodePtr *sPtr, int value );

int main()
{
   ListNodePtr startPtr = NULL; /* list pointer */
   int item; /* loop counter */

   /* insert integers into list */
   for ( item = 1; item < 11; item++ ) {
      insertItem( &startPtr, item );
   } /* end for */

   printList( startPtr );
   printf( "\n" );
   printListBackwards( startPtr );

   return 0; /* indicate successful termination */

} /* end main */

/* Insert a new value into the list in sorted order */
void insertItem( ListNodePtr *sPtr, int value )
{
   ListNodePtr newPtr;      /* new node */
   ListNodePtr previousPtr; /* previous node */
   ListNodePtr currentPtr;  /* current node */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      previousPtr = NULL;
      currentPtr = *sPtr; /* set currentPtr to start of list */

      /* loop to find correct location in list */
      while ( currentPtr != NULL && value > currentPtr->data ) {
         previousPtr = currentPtr;
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      /* insert at beginning of list */
      if ( previousPtr == NULL ) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      } /* end if */
      else { /* insert node between previousPtr and currentPtr */
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      } /* end else */

   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", value );
   } /* end else */

} /* end function insertItem */
```

```
/* Print the list */
void printList( ListNodePtr currentPtr )
{

   /* if list is empty */
   if ( !currentPtr ) {
      printf( "List is empty.\n\n" );
   } /* end if */
   else {

      /* loop while currentPtr does not equal NULL */
      while ( currentPtr ) {
         printf( "%d ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      printf( "*\n\n" );
   } /* end else */

} /* end function printList */

/* Print the list recursively backwards */
void printListBackwards( ListNodePtr currentPtr )
{

   /* if at end of list */
   if ( currentPtr == NULL ) {
      printf( "The list reversed is:\n" );
   } /* end if */
   else { /* recursive call */
      printListBackwards( currentPtr->nextPtr );
      printf( "%d ", currentPtr->data );
   } /* end else */

} /* end function printListBackwards */
```

```
the list is:
1 2 3 4 5 6 7 8 9 10

The list reversed is:
10 9 8 7 6 5 4 3 2 1
```

12.21 (*Recursively Search a List*) Write a function `searchList` that recursively searches a linked list 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.

ANS:

```
/* Exercise 12.21 Solution */
#include <stdio.h>
#include <stdlib.h>

/* ListNode structure definition */
struct ListNode {
   int data;                  /* node data */
   struct ListNode *nextPtr; /* pointer to next node */
}; /* end struct ListNode */

typedef struct ListNode ListNode;
typedef ListNode *ListNodePtr;

/* function prototypes */
void insertItem( ListNodePtr *sPtr, int value );
void printList( ListNodePtr currentPtr );
void instructions( void );
ListNodePtr searchList( ListNodePtr currentPtr, const int key );

int main()
{
   ListNodePtr startPtr = NULL; /* list pointer */
   ListNodePtr searchResultPtr; /* pointer to search result */
   int choice;                  /* user's menu choice */
   int item;                    /* value to insert into list */
   int searchKey;               /* value to search for in list */

   instructions(); /* display the menu */
   printf( "? " );
   scanf( "%d", &choice );

   /* while user does not choose 3 */
   while ( choice != 3 ) {

      /* determine user's choice */
      switch ( choice ) {

         /* insert an integer into the list */
         case 1:

            /* prompt user and read integer */
            printf( "Enter an integer: " );
            scanf( "\n%d", &item );

            /* insert integer and print list */
            insertItem( &startPtr, item );
            printList( startPtr );
            break; /* exit switch */

         /* search for given integer */
         case 2:

            /* prompt user and read integer */
            printf( "Enter integer to recursively search for: " );
            scanf( "%d", &searchKey );

```

```
            searchResultPtr = searchList( startPtr, searchKey );

            /* if searchKey not found */
            if ( searchResultPtr == NULL ) {
               printf( "%d is not in the list.\n\n", searchKey );
            } /* end if */
            else { /* if searchKey was found */
               printf( "%d is in the list.\n\n",
                  searchResultPtr->data );
            } /* end else */

            break; /* exit switch */

         /* default case */
         default:
            printf( "Invalid choice.\n\n" );
            instructions();
            break; /* exit switch */
      } /* end switch */

      printf( "? " );
      scanf( "%d", &choice ); /* get next choice */
   } /* end while */

   printf( "End of run.\n" );

   return 0; /* indicate successful termination */

} /* end main */

/* Print the instructions */
void instructions( void )
{
   printf( "Enter your choice:\n"
          "   1 to insertItem an element into the list.\n"
          "   2 to recursively search list for an element.\n"
          "   3 to end.\n" );
} /* end function instructions */

/* Insert a new value into the list in sorted order */
void insertItem( ListNodePtr *sPtr, int value )
{
   ListNodePtr newPtr;      /* new node */
   ListNodePtr previousPtr; /* previous node */
   ListNodePtr currentPtr;  /* current node */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( ListNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr ) {
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      previousPtr = NULL;
      currentPtr = *sPtr; /* set currentPtr to start of list */

      /* loop to find correct location in list */
      while ( currentPtr != NULL && value > currentPtr->data ) {
         previousPtr = currentPtr;
         currentPtr = currentPtr->nextPtr;
      } /* end while */
```

```

      /* insert at beginning of list */
      if ( previousPtr == NULL ) {
         newPtr->nextPtr = *sPtr;
         *sPtr = newPtr;
      } /* end if */
      else { /* insert node between previousPtr and currentPtr */
         previousPtr->nextPtr = newPtr;
         newPtr->nextPtr = currentPtr;
      } /* end else */

   } /* end if */
   else {
      printf( "%c not inserted. No memory available.\n", value );
   } /* end else */

} /* end function insertItem */

/* Print the list */
void printList( ListNodePtr currentPtr )
{

   /* if list is empty */
   if ( !currentPtr ) {
      printf( "List is empty.\n\n" );
   } /* end if */
   else {

      /* loop while currentPtr does not equal NULL */
      while ( currentPtr ) {
         printf( "%d --> ", currentPtr->data );
         currentPtr = currentPtr->nextPtr;
      } /* end while */

      printf( "NULL\n\n" );
   } /* end else */

} /* end function printList */

/* search for key in list */
ListNodePtr searchList( ListNodePtr currentPtr, const int key )
{

   /* if currentPtr is at end of list */
   if ( currentPtr == NULL ) {
      return NULL; /* key not found */
   } /* end if */
   else if ( currentPtr->data == key ) {
      return currentPtr; /* key found */
   } /* end else if */
   else {
      searchList( currentPtr->nextPtr, key ); /* keep searching */
   } /* end else */

} /* end function ListNodePtr */
```

```
Enter your choice:
   1 to insertItem an element into the list.
   2 to recursively search list for an element
   3 to end.
? 1
Enter an integer: 7
The list is:
7 --> NULL

? 1
Enter an integer: 99
The list is:
7 --> 99 --> NULL

? 1
Enter an integer: 56
The list is:
7 --> 56 --> 99 --> NULL

? 1
Enter an integer: 73
The list is:
7 --> 56 --> 73 --> 99 --> NULL

? 2
Enter integer to recursively search for: 7
7 is in the list.

? 2
Enter integer to recursively search for: 55
55 is not in the list.

? 2
Enter integer to recursively search for: 99
99 is in the list.

? 3
End of run.
```

12.22 (*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 node 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 must take its place. However, the pointer in the parent node cannot simply 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: *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 to the replacement node's position. 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 function `deleteNode` which takes as its arguments a pointer to the root node of the tree 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. 12.19 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.

12.23 (*Binary Tree Search*) Write function `binaryTreeSearch` that attempts to locate a specified value in a binary search tree. 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.

ANS:

```c
/* Exercise 12.23 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* TreeNode structure definition */
struct TreeNode {
   struct TreeNode *leftPtr;  /* pointer to left subtree */
   int data;                  /* node data */
   struct TreeNode *rightPtr; /* pointer to right subtree */
}; /* end struct TreeNode */

typedef struct TreeNode TreeNode;
typedef TreeNode *TreeNodePtr;

/* function prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
TreeNodePtr binaryTreeSearch( TreeNodePtr treePtr, const int key );

int main()
{
   int i;                          /* loop counter */
   int item;                       /* random value to insert in tree */
   int searchKey;                  /* value to search for */
   TreeNodePtr rootPtr = NULL;     /* points to the tree root */
   TreeNodePtr searchResultPtr;    /* pointer to search result */

   srand( time( NULL ) ); /* randomize */
   printf( "The numbers being placed in the tree are:\n" );

   /* insert random values between 1 and 20 in the tree */
   for ( i = 1; i <= 10; i++ ) {
      item = 1 + rand() % 20;
      printf( "%3d", item );
      insertNode( &rootPtr, item );
   } /* end for */

   /* prompt user and read integer search key */
   printf( "\n\nEnter an integer to search for: " );
   scanf( "%d", &searchKey );

   searchResultPtr = binaryTreeSearch( rootPtr, searchKey );

   /* if searchKey not found */
   if ( searchResultPtr == NULL ) {
      printf( "\n%d was not found in the tree.\n\n", searchKey );
   } /* end if */
   else { /* if key found */
      printf( "\n%d was found in the tree.\n\n",
         searchResultPtr->data );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```c

/* insert a node into the tree */
void insertNode( TreeNodePtr *treePtr, int value )
{

   /* if treePtr is NULL */
   if ( *treePtr == NULL ) {

      /* dynamically allocate memory */
      *treePtr = malloc( sizeof( TreeNode ) );

      /* if memory was allocated, insert node */
      if ( *treePtr != NULL ) {
         ( *treePtr )->data = value;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      } /* end if */
      else {
         printf( "%d not inserted. No memory available.\n", value );
      } /* end else */

   } /* end if */
   else { /* recursively call insertNode */

      /* insert node in left subtree */
      if ( value < ( *treePtr )->data ) {
         insertNode( &( ( *treePtr )->leftPtr ), value );
      } /* end if */
      else {

         /* insert node in right subtree */
         if ( value > ( *treePtr )->data ) {
            insertNode( &( ( *treePtr )->rightPtr ), value );
         } /* end if */
         else { /* duplicate value */
            printf( "dup" );
         } /* end else */

      } /* end else */

   } /* end else */

} /* end function insertNode */

/* search for key in tree */
TreeNodePtr binaryTreeSearch( TreeNodePtr treePtr, const int key )
{

   /* traverse the tree inOrder */
   if ( treePtr == NULL ) {
      return NULL; /* key not found */
   } /* end if */
   else if ( treePtr->data == key ) {
      return treePtr; /* key found */
   } /* end else if */
   else if ( key < treePtr->data ) {
      binaryTreeSearch( treePtr->leftPtr, key ); /* search left */
   } /* end else if */
   else if ( key > treePtr->data ) {
      binaryTreeSearch( treePtr->rightPtr, key ); /* search right */
   } /* end else if */
```

```

} /* end function binaryTreeSearch */
```

```
The numbers being placed in the tree are:
 18   9   7   2 13   2dup 10   1 19   2dup

Enter an integer to search for: 8

8 was not found in the tree.
```

12.24 (*Level Order Binary Tree Traversal*) The program of Fig. 12.19 illustrated three recursive methods of traversing a binary tree—inorder traversal, preorder traversal, and postorder traversal. 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 the queue data structure 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 function `levelOrder` to perform a level order traversal of a binary tree. The function should take as an argument a pointer to the root node of the binary tree. Modify the program of Fig. 12.19 to use this function. Compare the output from this function to the outputs of the other traversal algorithms to see that it worked correctly. [*Note:* You will also need to modify and incorporate the queue processing functions of Fig. 12.13 in this program.]

ANS:

```
/* Exercise 12.24 solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* TreeNode structure definition */
struct TreeNode {
   struct TreeNode *leftPtr;  /* pointer to left subtree */
   int data;                  /* node data */
   struct TreeNode *rightPtr; /* pointer to right subtree */
}; /* end struct TreeNode */

typedef struct TreeNode TreeNode;
typedef TreeNode *TreeNodePtr;

/* tree function prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
void levelOrderTraversal( TreeNodePtr treePtr );

/* QueueNode structure definition */
struct QueueNode {
   TreeNodePtr data;          /* node data */
   struct QueueNode *nextPtr; /* pointer to next node */
}; /* end struct QueueNode */

typedef struct QueueNode QueueNode;
typedef QueueNode *QueueNodePtr;

```

```
/* queue function prototypes */
int isEmpty( QueueNodePtr headPtr );
TreeNodePtr dequeue( QueueNodePtr *headPtr, QueueNodePtr * tailPtr );
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr,
              TreeNodePtr node );

int main()
{
   int i;                     /* loop counter */
   int item;                  /* random value to insert in tree */
   TreeNodePtr rootPtr = NULL; /* points to the tree root */

   srand( time( NULL ) ); /* randomize */
   printf( "The values being inserted in the tree are:\n" );

   /* insert random values between 1 and 15 in the tree */
   for ( i = 1; i <= 15; i++ ) {
      item = 1 + rand() % 20;
      printf( " %d", item );
      insertNode( &rootPtr, item );
   } /* end for */

   /* traverse the tree level order */
   printf( "\n\nThe level order traversal is:\n" );
   levelOrderTraversal( rootPtr );
   printf( "\n" );

   return 0; /* indicate successful termination */

} /* end main */

/* Level order traversal of a binary tree */
void levelOrderTraversal( TreeNodePtr ptr )
{
   QueueNodePtr head = NULL; /* points to queue head */
   QueueNodePtr tail = NULL; /* points to queue tail */
   TreeNodePtr node;         /* current tree node */

   /* if tree is not empty */
   if ( ptr != NULL ) {
      enqueue( &head, &tail, ptr ); /* enqueue root nood */

      /* while queue is not empty */
      while ( !isEmpty( head ) ) {

         /* dequeue next node and print data */
         node = dequeue( &head, &tail );
         printf( "%d ", node->data );

         /* insert left child node in the queue */
         if ( node->leftPtr != NULL ) {
            enqueue( &head, &tail, node->leftPtr );
         } /* end if */

         /* insert right child node in the queue */
         if ( node->rightPtr != NULL ) {
            enqueue( &head, &tail, node->rightPtr );
         } /* end if */

      } /* end while */

   } /* end if */
```

```
} /* end function levelOrderTraversal */

/* insert a node into the tree */
void insertNode( TreeNodePtr *treePtr, int value )
{

   /* if treePtr is NULL */
   if ( *treePtr == NULL ) {

      /* dynamically allocate memory */
      *treePtr = malloc( sizeof( TreeNode ) );

      /* if memory was allocated, insert node */
      if ( *treePtr != NULL ) {
         ( *treePtr )->data = value;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      } /* end if */
      else {
         printf( "%d not inserted. No memory available.\n", value );
      } /* end else */

   } /* end if */
   else { /* recursively call insertNode */

      /* insert node in left subtree */
      if ( value < ( *treePtr )->data ) {
         insertNode( &( ( *treePtr )->leftPtr ), value );
      } /* end if */
      else {

         /* insert node in right subtree */
         if ( value > ( *treePtr )->data ) {
            insertNode( &( ( *treePtr )->rightPtr ), value );
         } /* end if */
         else { /* duplicate value */
            printf( "dup" );
         } /* end else */

      } /* end else */

   } /* end else */

} /* end function insertNode */

/* enqueue node */
void enqueue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr, TreeNodePtr node )
{
   QueueNodePtr newPtr; /* temporary node pointer */

   /* dynamically allocate memory */
   newPtr = malloc( sizeof( QueueNode ) );

   /* if newPtr does not equal NULL */
   if ( newPtr != NULL ) {
      newPtr->data = node;
      newPtr->nextPtr = NULL;

      /* if queue is empty, insert at head */
      if ( isEmpty( *headPtr ) ) {
```

```
            *headPtr = newPtr;
         } /* end if */
         else { /* insert at tail */
            ( *tailPtr )->nextPtr = newPtr;
         } /* end else */

         *tailPtr = newPtr;
      } /* end if */
      else {
         printf( "Node not inserted\n" );
      } /* end else */

   } /* end function enqueue */

   /* dequeue node from queue */
   TreeNodePtr dequeue( QueueNodePtr *headPtr, QueueNodePtr *tailPtr )
   {
      TreeNodePtr node;     /* dequeued node */
      QueueNodePtr tempPtr; /* temporary node pointer */

      /* dequeue node and reset queue headPtr */
      node = ( *headPtr )->data;
      tempPtr = *headPtr;
      *headPtr = ( *headPtr )->nextPtr;

      /* if queue is empty */
      if ( *headPtr == NULL ) {
         *tailPtr = NULL;
      } /* end if */

      free( tempPtr ); /* free memory */

      return node; /* return dequeued node */

   } /* end function dequeue */

   /* is queue empty? */
   int isEmpty( QueueNodePtr headPtr )
   {
      return headPtr == NULL; /* return NULL is queue is empty */

   } /* end function isEmpty */
```

```
The values being inserted in the tree are:
 5 10 7 5dup 11 9 15 1 7dup 20 6 20dup 4 16 4dup

The level order traversal is:
5 1 10 4 7 11 6 9 15 20 16
```

12.25 (*Printing Trees*) Write a recursive function `outputTree` to display a binary tree 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 the right of the screen. Each row is output vertically. For example, the binary tree illustrated in Fig. 12.22 is output as follows:

```
               99
          97
               92
     83
               72
          71
               69
49
               44
          40
               32
     28
               19
          18
               11
```

Note 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 as arguments a pointer to the root node of the tree and an integer `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 the rightmost 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` statement 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`.

ANS:

```c
/* Exercise 12.25 Solution */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* TreeNode structure definition */
struct TreeNode {
   struct TreeNode *leftPtr;  /* pointer to left subtree */
   int data;                  /* node data */
   struct TreeNode *rightPtr; /* pointer to right subtree */
}; /* end struct TreeNode */

typedef struct TreeNode TreeNode;
typedef TreeNode *TreeNodePtr;

/* function prototypes */
void insertNode( TreeNodePtr *treePtr, int value );
void outputTree( TreeNodePtr treePtr, int spaces );

int main()
{
   int i;                      /* loop counter */
```

```
   int item;                     /* random value to be inserted */
   int totalSpaces = 0;          /* spaces preceeding output */
   TreeNodePtr rootPtr = NULL; /* points to the tree root */

   srand( time( NULL ) ); /* randomize */
   printf( "The numbers being placed in the tree are:\n" );

   /* insert random values between 1 and 10 in the tree */
   for ( i = 1; i <= 10; i++ ) {
      item = rand() % 15;
      printf( "%3d", item );
      insertNode( &rootPtr, item );
   } /* end for */

   printf( "\n\n" );
   outputTree( rootPtr, totalSpaces ); /* display tree */

   return 0; /* indicate successful termination */

} /* end main */

/* insert a node into the tree */
void insertNode( TreeNodePtr *treePtr, int value )
{

   /* if treePtr is NULL */
   if ( *treePtr == NULL ) {

      /* dynamically allocate memory */
      *treePtr = malloc( sizeof( TreeNode ) );

      /* if memory was allocated, insert node */
      if ( *treePtr != NULL ) {
         ( *treePtr )->data = value;
         ( *treePtr )->leftPtr = NULL;
         ( *treePtr )->rightPtr = NULL;
      } /* end if */
      else {
         printf( "%d not inserted. No memory available.\n", value );
      } /* end else */

   } /* end if */
   else { /* recursively call insertNode */

      /* insert node in left subtree */
      if ( value < ( *treePtr )->data ) {
         insertNode( &( ( *treePtr )->leftPtr ), value );
      } /* end if */
      else {

         /* insert node in right subtree */
         if ( value > ( *treePtr )->data ) {
            insertNode( &( ( *treePtr )->rightPtr ), value );
         } /* end if */
         else { /* duplicate value */
            printf( "dup" );
         } /* end else */

      } /* end else */

   } /* end else */
```

```

} /* end function insertNode */

/* display the tree */
void outputTree( TreeNodePtr treePtr, int spaces )
{
   int loop; /* loop counter */

   /* while not the end of tree */
   while ( treePtr != NULL ) {

      /* recursive call with right subtree */
      outputTree( treePtr->rightPtr, spaces + 5 );

      /* loop and output spaces */
      for ( loop = 1; loop <= spaces; loop++ ) {
         printf( " " );
      } /* end for */

      printf( "%d\n", treePtr->data );

      /* set pointer to left subtree and make recursive call */
      outputTree( treePtr->leftPtr, spaces + 5 );
      treePtr = NULL;
   } /* end while */

} /* end function outputTree */
```

```
The numbers being placed in the tree are:
  1  5  6 14 11  2  9  9dup 14dup  8

               14
                    11
                         9
                              8
          6
     5
          2
1
```

SPECIAL SECTION: BUILDING YOUR OWN COMPILER

In Exercise 7.18 and Exercise 7.19, we introduced Simpletron Machine Language (SML) and created the Simpletron computer simulator to execute programs written in SML. In this section, we build a compiler that converts programs written in a high-level programming language to SML. This section “ties” together the entire programming process. We will write programs in this new high-level language, compile the programs on the compiler we build, and run the programs on the simulator we built in Exercise 7.19.

12.26 (*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` or `end` (see Fig. 12.23). 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.

Command	Example statement	Description
`rem`	`50 rem this is a remark`	Any text following the command `rem` is for documentation purposes only 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 equal 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 program control to line `80` if the condition is true; otherwise, continue execution with the next statement.
`end`	`99 end`	Terminate program execution.

Fig. 12.1 Simple commands.

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 the 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 manipulation (reading a string, writing a string, comparing strings, etc.). If a string is encountered in a Simple program (after a command other than `rem`), the compiler generates a syntax error. Our compiler will assume that Simple programs are entered correctly. Exercise 12.29 asks the student to modify the compiler to perform syntax error checking.

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: `<`, `>`, `<=`, `>=`, `==` or `!=`. The precedence of these operators is the same as in C.

Let us now consider several Simple programs that demonstrate Simple's features. The first program (Fig. 12.24) reads two integers from the keyboard, stores the values in variables `a` and `b`, and computes and prints their sum (stored in variable `c`).

```
10 rem   determine and print the sum of two integers
15 rem
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
```

Figure 12.25 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.

```
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
```

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 12.26 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   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
```

Using the sample programs of Fig. 12.24, Fig. 12.25 and Fig. 12.26 as your guide, write a Simple program to accomplish each of the following:

a) Input three integers, determine their average and print the result.

ANS:

```
5 rem     Exercise 12.26 Part A Solution
6 rem
10 input x
15 input y
10 input x
21 rem    calculate average a
25 let a = ( x + y + z ) / 3
26 rem
30 print a
99 end
```

b) Use a sentinel-controlled loop to input 10 integers and compute and print their sum.
ANS:

```
5 rem     Exercise 12.26 Part B Solution
6 rem
10 input n
12 rem    set up sentinel loop
15 if n == 9999 goto 40
16 rem
17 rem    add n to the sum of s
20 let s = s + n
21 rem
22 rem    loop to get next n
25 goto 10
36 rem    print sum s
40 print s
99 end
```

c) Use a counter-controlled loop to input seven integers, some positive and some negative, and compute and print their average.
ANS:

```
5 rem     exercise 12.26 Part C Solution
6 rem
10 input m
11 rem    increment counter by 1
12 rem    c is automatically initiated to 0
13 rem    when created
15 let c = c + 1
16 rem
17 rem    calculate sum s
20 let s = s + m
22 rem    loop to get next m
23 rem    if c is not yet 7
25 if c <= 7 goto 10
26 rem
27 rem    compute average a
30 let a = s / 7
31 rem
35 print a
99 end
```

d) Input a series of integers and determine and print the largest. The first integer input indicates how many numbers should be processed.

ANS:

```
5 rem      Exercise 12.26 Part D Solution
6 rem
7 rem      Enter the number of integers
8 rem      to be processed
10 input n
23 rem     begin entering numbers t
25 input t
26 rem     check if t is larger than 1
27 rem     l’s initial value is zero
30 rem     if t <= 1 goto 50
31 rem     t must be larger than 1
32 rem     so assign t as largest
35 let l = t
49 rem     decrement n
50 let n = n - 1
59 rem     test for loop exit condition
60 if  n == 0 goto 80
69 rem     loop to get next t
70 goto 25
79 rem     print largest value
80 print l
99 end
```

e) Input 10 integers and print the smallest.

ANS:

```
1 rem     Exercise 12.26 Part E Solution
2 rem
3 rem     set counter c equal to 1
5 let c = 1
6 rem     input integer m
7 rem     assign first entry to
8 rem     the smallest value s
9 input m
10 let s = m
11 rem    enter main loop
13 goto 20
14 rem    main loop
15 input m
18 rem    determine if m is smaller
19 rem    than current s
20 if m < s goto 50
29 rem    increment counter
30 let c = c + 1
34 rem    exit when c becomes 11
35 if  c == 11 goto 60
39 rem    loop for next m
40 goto 15
48 rem    assign m to s as
49 rem    smallest value
50 let s = m
51 rem    loop to counter increment
55 goto 30
59 rem    print smallest value
60 print s
99 end
```

f) Calculate and print the sum of the even integers from 2 to 30.
ANS:

```
5 rem     Exercise 12.26 Part F Solution
6 rem
7 rem
9 rem     initialize i to 2
10 let i = 2
14 rem    store sum in s
15 let s = s + 1
19 rem  increment i by 2
20 let i = i + 2
28 rem    set loop terminating
29 rem    condition at 32
30 if < 32 goto 15
39 rem    print sum
40 print s
99 end
```

g) Calculate and print the product of the odd integers from 1 to 9.
ANS:

```
5 rem     Exercise 12.26 Part G Solution
6 rem
7 rem
9 rem     initialize k to 1
10 let k = 1
11 rem    initialize p to 1
13 let p = 1
14 rem    store product in p
15 let p = p * k
19 rem    increment k by 1
20 let k = k + 2
28 rem    set loop terminating
29 rem    condition at 10
30 if k < 10 goto 15
39 rem    print product
40 print p
99 end
```

12.27 (*Building A Compiler; Prerequisite: Complete Exercise 7.18, Exercise 7.19, Exercise 12.12, Exercise 12.13 and Exercise 12.26*) Now that the Simple language has been presented (Exercise 12.26), we discuss how to build our Simple compiler. First, we consider the process by which a Simple program is converted to SML and executed by the Simpletron simulator (see Fig. 12.27). 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 7.19 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.

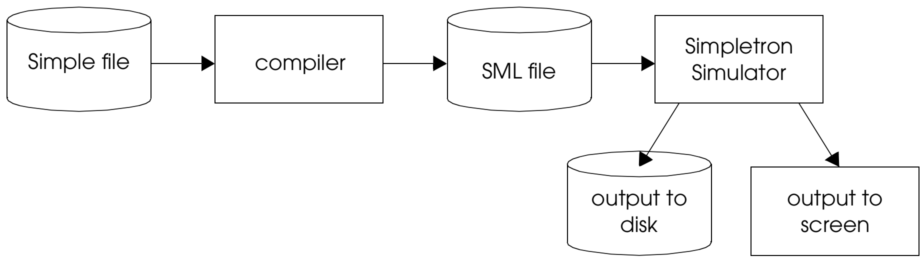


Fig. 12.2 Writing, compiling and executing a Simple language program.

The compiler performs two *passes* of the Simple program to convert it to SML. The first pass constructs a *symbol table* in which every *line number*, *variable name* and *constant* 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(s) for each Simple statement. 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 incomplete 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` is an array of `tableEntry` structures 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.

The `tableEntry` structure definition is as follows:

```
struct tableEntry {
   int symbol;
   char type;      /* 'C', 'L' or 'V' */
   int location;   /* 00 to 99 */
};
```

Each `tableEntry` structure 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, or `'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 7 that an SML instruction is a four-digit integer that comprises 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 will be 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` statement, 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 possibly 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. Exercise 12.12 and Exercise 12.13 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, 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" that allows our compiler to 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 on 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:

1) Search the `flags` array for an unresolved reference (i.e., an element with a value other than `-1`).
2) Locate the structure in array `symbolTable` containing the symbol stored in the `flags` array (be sure that the type of the symbol is `'L'` for line number).
3) Insert the memory location from structure member `location` into the instruction with the unresolved reference (remember that an instruction containing an unresolved reference has operand `00`).
4) 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).

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 are illustrated in Fig. 12.3. The symbol table constructed by the first pass is shown in Fig. 12.29

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 +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. 12.3 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

Fig. 12.4 Symbol table for program of Fig. 12.3

Symbol	Type	Location
't'	V	95
45	L	14
50	L	14
55	L	15
60	L	15
99	L	16

Fig. 12.4 Symbol table for program of Fig. 12.3

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 (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.

Step-by-Step View of the Compilation Process

Let us now walk through the compilation process for the Simple program in Fig. 12.3. 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 Chapter 8 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 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 simply 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 already 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 and pops the stack again into the left operand of the operator, 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 7.19 to take its input from a file specified by the user (see Chapter 11). Also, the simulator should output its results to a disk file in the same format as the screen output.

b) Modify the infix-to-postfix evaluation algorithm of Exercise 12.12 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 multi-digit integer operands and variable name operands. Also, the algorithm should now implement the previously discussed "hook" 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.

12.28 (*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. 12.30 illustrates five of the SML instructions that were produced in the compilation of the program in Fig. 12.3. 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, 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. This optimization would enable the Simpletron to execute the program faster because there are fewer instructions in this version. Figure 12.31 illustrates the optimized SML for the program of Fig. 12.3. Note that there are four fewer instructions in the optimized code—a memory-space savings of 25%.

```
04 +2098        (load)
05 +3097        (add)
06 +2196        (store)
07 +2096        (load)
08 +2198        (store)
```

Modify the compiler to provide an option for optimizing the Simpletron Machine Language code it produces. Manually compare the non-optimized code with the optimized code, and calculate the percentage reduction.

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
`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

Fig. 12.5 Optimized code for the program of Fig. 12.28.

Simple program	SML location and instruction	Description
	`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. 12.5 Optimized code for the program of Fig. 12.28.

12.29 (*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 7.19.

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.
c) Allow the compiler to recognize uppercase and lowercase letters in Simple statements (e.g., `'A'` is equivalent to `'a'`). No modifications to the Simpletron 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`s 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
    rem Simple statements
next
```

j) 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.
k) Allow repetition structures of the form

```
for x = 2 to 10
    rem Simple statements
next
```

l) This `for` statement loops from `2` to `10` with a default increment of `1`. No modifications to the Simpletron Simulator are required.
m) 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.

n) Allow the compiler to process floating-point values in addition to integers. The Simpletron Simulator must also be modified to process floating-point values.

12.30 (*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 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 12.26. The program should use the infix-to-postfix converter developed in Exercise 12.12 and the postfix evaluator developed in Exercise 12.13 to evaluate expressions in a `let` statement. The same restrictions placed on the Simple language in Exercise 12.26 should be adhered to in this program. Test the interpreter with the Simple programs written in Exercise 12.26. 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 7.19.

13

The Preprocessor: Solutions

SOLUTIONS

13.4 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.

ANS:

```c
/* Exercise 13.4 Solution: sphere volume macro */
#include <stdio.h>

#define PI 3.14159 /* constant representing Pi */

/* define preprocessor directive sphere volume */
#define SPHEREVOLUME( r ) ( 4.0 / 3.0 * PI * ( r ) * ( r ) * ( r ) )

int main()
{
   int i; /* loop counter */

   /* print header */
   printf( "%10s%10s\n", "Radius", "Volume" );

   /* use sphere volume macro */
   for ( i = 1; i <= 10; i++ ) {
      printf( "%10d%10.3f\n", i, SPHEREVOLUME( i ) );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
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
```

13.5 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.

ANS:

```
/* Exercise 13.5 Solution */
#include <stdio.h>

/* macro to add two value */
#define SUM( x, y ) ( ( x ) + ( y ) )

int main()
{

   /* display sum of x and y using macro SUM */
   printf( "The sum of x and y is %d\n", SUM( 6, 7 ) );

   return 0; /* indicate successful termination */

} /* end main */
```

13.6 Write a program that defines and uses macro `MINIMUM2` to determine the smallest of two numeric values. Input the values from the keyboard.

ANS:

```c
/* Exercise 13.6 Solution */
#include <stdio.h>

/* macro to determine smallest of two values */
#define MINIMUM2( x, y ) ( ( x ) < ( y ) ? ( x ) : ( y ) )

int main()
{
   int a;    /* first integer */
   int b;    /* second integer */
   double c; /* first double */
   double d; /* second double */

   /* prompt user and read two integers */
   printf( "Enter two integers: " );
   scanf( "%d%d", &a, &b );

   /* use macro MINIMUM to determine and display
      smallest user entered integer */
   printf( "The minimum of %d and %d is %d\n\n", a, b,
      MINIMUM2( a,b ) );

   /* prompt user and read two doubles */
   printf( "Enter two doubles: " );
   scanf( "%lf%lf", &c, &d );

   /* use macro MINIMUM to determine and display
      smallest user entered double */
   printf( "The minimum of %.2f and %.2f is %.2f\n\n",
           c, d, MINIMUM2( c,d ) );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter two integers: 4 9
The minimum of 4 and 9 is 4

Enter two doubles: 45.7 13.2
The minimum of 45.70 and 13.20 is 13.20
```

13.7 Write a program that defines and uses macro `MINIMUM3` to determine the smallest of three numeric values. Macro `MINIMUM3` should use macro `MINIMUM2` defined in Exercise 13.6 to determine the smallest number. Input the values from the keyboard.

ANS:

```c
/* Exercise 13.7 Solution */
#include <stdio.h>

/* macro to determine smallest of two values */
#define MINIMUM2( x, y ) ( ( x ) < ( y ) ? ( x ) : ( y ) )

/* macro that uses MINIMUM2 to determine smallest of three values */
#define MINIMUM3( u, v, w ) ( MINIMUM2( w, MINIMUM2( u, v ) ) )

int main()
{
   int a;    /* first integer */
   int b;    /* second integer */
   int c;    /* third integer */
   double d; /* first double */
   double e; /* second double */
   double f; /* third double */

   /* prompt user and read three integers */
   printf( "Enter three integers: " );
   scanf( "%d%d%d", &a, &b, &c );

   /* use macro MINIMUM3 to determine smallest
      of three user input integers */
   printf( "The minimum of %d, %d, and %d is %d\n\n",
          a, b, c, MINIMUM3( a, b, c ) );

   /* prompt user and read three doubles */
   printf( "Enter three doubles: " );
   scanf( "%lf%lf%lf", &d, &e, &f );

   /* use macro MINIMUM3 to determine smallest
      ofthree user input doubles */
   printf( "The minimum of %.2f, %.2f, and %.2f is %.2f\n\n",
          d, e, f, MINIMUM3( d, e, f ) );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter three integers: 7 2 10
The minimum of 7, 2, and 10 is 2

Enter three doubles: 4.9 93.2 1.3
The minimum of 4.90, 93.20, and 1.30 is 1.30
```

13.8 Write a program that defines and uses macro PRINT to print a string value.
ANS:

```
/* Exercise 13.8 Solution */
#include <stdio.h>

/* macro that prints its argument */
#define PRINT( string ) printf( "%s", ( string ) )

int main()
{
   char text[ 20 ]; /* array to hold user input string */

   /* prompt user and read string */
   PRINT( "Enter a string: " );
   scanf( "%s", text );

   /* use macro to output string entered by user */
   PRINT( "The string entered was: " );
   PRINT( text );
   PRINT( "\n" );

   return 0; /* indicate successful termination */

} /* end main */
```

```
Enter a string: Hello
The string entered was: Hello
```

13.9 Write a program that defines and 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.
ANS:

```
/* Exercise 13.9 Solution */
#include <stdio.h>

/* macro that prints an array of values */
#define PRINTARRAY( a, n )  for ( i = 0; i < ( n ); i++ ) \
                             printf( "%d ", a[ i ] )

int main()
{
   int i; /* defines i for use in PRINTARRAY */

   /* initialize array to be printed */
   int b[ 10 ] = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };

   printf( "The array values are:\n" );
   PRINTARRAY( b, 10 ); /* print the array */

   return 0; /* indicate successful termination */

} /* end main */
```

```
The array values are:
2 4 6 8 10 12 14 16 18 20
```

13.10 Write a program that defines and 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.

ANS:

```
/* Exercise 13.10 Solution */
#include <stdio.h>

/* macro that adds values of a numeric array */
#define SUMARRAY( a, n ) for ( i = 0; i < ( n ); i++ ) \
                            sum += a[ i ]

int main()
{
   int i;       /* loop counter */
   int sum = 0; /* sum of array elements */

   /* initialize array whose values will be added */
   int b[ 10 ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   /* use macro SUMARRAY to add elements of array */
   SUMARRAY( b, 10 );
   printf( "The sum of the elements of array b is %d\n", sum );

   return 0; /* indicate successful termination */

} /* end main */
```

```
The sum of the elements of array b is 55
```

14

Other C Topics: Solutions

SOLUTIONS

14.2 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.

ANS:

```
/* Exercise 14.2 Solution */
#include <stdio.h>
#include <stdarg.h>

/* function with variable length argument list */
int sum( int i, ... );

int main()
{
   int a = 1; /* values to sum */
   int b = 2;
   int c = 3;
   int d = 4;
   int e = 5;

   /* display integer values */
   printf( "%s%d, %s%d, %s%d, %s%d, %s%d\n", "a = ", a, "b = ",
           b, "c = ", c, "d = ", d, "e = ", e );

   /* call sum with different number of arguments in each call */
   printf( "%s%d\n%s%d\n%s%d\n%s%d\n", "The sum of a and b is: ",
      sum( 2, a, b ), "The sum of a, b, and c is: ", sum( 3, a, b, c ),
      "The sum of a, b, c, and d is: ", sum( 4, a, b, c, d ),
      "The sum of a, b, c, d, and e is: ", sum( 5, a, b, c, d, e ) );

   return 0; /* indicate successful termination */

} /* end main */

```

```
/* sums integers passed as arguments */
int sum( int i, ... )
{
   int total = 0; /* sum of integers */
   int j;         /* loop counter */
   va_list ap;    /* variable length argument list */

   va_start( ap, i ); /* invoke macro to access arguments */

   /* calculate total */
   for ( j = 1; j <= i; j++ ) {
      total += va_arg( ap, int );
   } /* end for */

   va_end( ap ); /* perform termination housekeeping */

   return total; /* return sum of arguments */

} /* end function sum */
```

```
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
```

14.3 Write a program that prints the command-line arguments of the program.
ANS:

```
/* Exercise 14.3 Solution */
#include <stdio.h>

int main( int argc, char *argv[] )
{
   int i; /* loop counter */

   printf( "The command line arguments are:\n" );

   /* display arguments given to program at command line */
   for ( i = 0; i < argc; i++ ) {
      printf( "%s ", argv[ i ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
The command line arguments are:
C:\P14_3.exe arg1 arg2 arg3
```

14.4 Write a program that sorts an array of integers 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.]

ANS:

The DOS command line `p14_4.exe -a < p14_4.dat` produces the first output shown below, and the DOS command line `p14_4.exe -d < p14_4.dat` produces the second output shown below. The data file `p14_4.dat` contains the values 8, 2, 1, 7, 5, 4, 9, 11, 19, and 13.

```
/* Exercise 14.4 Solution */
#include <stdio.h>

int main( int argc, char *argv[] )
{
   int a[ 100 ]; /* array of integers from user */
   int count;    /* count of integers entered */
   int temp;     /* temporary integer for swapping */
   int i;        /* loop counter */
   int j;        /* loop counter */
   int order;    /* sort in ascending or descending order */

   /* tell user if improper arguments were passed */
   if ( argc != 2 ) {
      printf( "Usage: p14_4 -option\n" );
   } /* end if */
   else {

      /* prompt user for integers to be sorted */
      printf( "Enter up to 100 integers ( EOF to end input ): " );

      /* store integers until 100 elements or EOF entered */
      for ( count = 0; !feof( stdin ) && count < 100; count++ ) {
         scanf( "%d", &a[ count ] );
      } /* end for */

      /* set order based on command-line argument */
      order = ( argv[ 1 ][ 1 ] == 'd' ) ? 0 : 1;

      /* loop through array and swap elements as needed */
      for ( i = 1; i < count - 1; i++ ) {

         for ( j = 0; j < count - 1; j++ ) {

            /* swap in ascending order if that option specified */
            if ( order == 1 ) {

               if ( a[ i ] < a[ j ] ) {
                  temp = a[ i ];
                  a[ i ] = a[ j ];
                  a[ j ] = temp;
               } /* end if */

            } /* end if */
            else { /* swap in descending order */

               if ( a[ i ] > a[ j ] ) {
                  temp = a[ i ];
                  a[ i ] = a[ j ];
                  a[ j ] = temp;
               } /* end if */
```

```
            } /* end else */

         } /* end for */

      } /* end for */

      printf( "\n\nThe sorted array is:\n" );

      /* display sorted array */
      for ( i = 0; i < count - 1; i++ ) {
         printf( "%d ", a[ i ] );
      } /* end for */

      printf( "\n" );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
The sorted array is:
1 2 4 5 7 8 9 11 13 19
```

```
The sorted array is:
19 13 11 9 8 7 5 4 2 1
```

14.5 Write a program that places a space between each character in a file. The program should first write the contents of the file being modified into a temporary file with spaces between each character, then copy the file back to the original file. This operation should overwrite the original contents of the file.

ANS:

```
/* Exercise 14.5 Solution */
#include <stdio.h>

int main()
{
   FILE *filePtr;       /* pointer to file being modified */
   FILE *tempFilePtr;   /* temporary file pointer */
   int c;               /* current character */
   char fileName[ 30 ]; /* name of file to be modified */

   /* prompt user and read file name */
   printf( "This program inserts spaces between each character\n"
           "of a file. Enter a file to be modified: " );
   scanf( "%s", fileName );

   /* exit program if file cannot be opened */
   if ( ( filePtr = fopen( fileName, "r+" ) ) != NULL ) {

      /* exit program if temporary file cannot be opened */
      if ( ( tempFilePtr = tmpfile() ) != NULL ) {
         printf( "\nThe file before modification is:\n" );

         /* read each character from file */
         while ( ( c = getc( filePtr ) ) != EOF ) {
            putchar( c );
```

```
            putc( c, tempFilePtr ); /* put character in temp file */

            /* write a space to temp file */
            if ( c != '\n' ) {
               putc( ' ', tempFilePtr );
            } /* end if */

         } /* end while */

         rewind( tempFilePtr ); /* rewind both file pointers */
         rewind( filePtr );
         printf( "\n\nThe file after modification is:\n" );

         /* read each character from temp file */
         while ( ( c = getc( tempFilePtr ) ) != EOF ) {
            putchar( c );
            putc( c, filePtr ); /* rewrite character to file */
         } /* end while */

      } /* end if */
      else {
         printf( "Unable to open temporary file\n" );
      } /* end else */

   } /* end if */
   else {
      printf( "Unable to open %s\n", fileName );
   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */
```

```
This program inserts spaces between each character
of a file. Enter a file to be modified: test.dat

The file before modification is:
This is a test file for
exercise 14.5.


The file after modification is:
T h i s   i s   a   t e s t   f i l e   f o r
e x e r c i s e   1 4 . 5 .
```

14.6 Read the manuals for your compiler to determine what signals are supported by the signal handling library (`signal.h`). Write a program that contains signal handlers for the standard 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 typing *<ctrl> c* to generate a signal of type `SIGINT`.

14.7 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 1/2 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.

ANS:

```c
/* Exercise 14.7 Solution */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int count;  /* number of elements in array */
   int i;      /* loop counter */
   int *array; /* pointer to the array */

   /* prompt user and read integer size of array */
   printf( "This program dynamically allocates an array of integers.\n"
          "Enter the number of elements in the array: " );
   scanf( "%d", &count );

   /* dynamically allocate memory */
   array = calloc( count, sizeof( int ) );

   /* initialize elements of array with user-entered data */
   for ( i = 0; i < count; i++ ) {
      printf( "Enter an integer: " );
      scanf( "%d", &array[ i ] );
   } /* end for */

   printf( "\nThe elements of the array are:\n" );

   /* display the original array */
   for ( i = 0; i < count; i++ ) {
      printf( "%d ", array[ i ] );
   } /* end for */

   /* reallocate to half the original size */
   realloc( array, count / 2 * sizeof( int ) );

   printf( "\n\nThe elements of the array after reallocation are:\n" );

   /* display array after cut in half */
   for ( i = 0; i < count / 2; i++ ) {
      printf( "%d ", array[ i ] );
   } /* end for */

   return 0; /* indicate successful termination */

} /* end main */
```

```
This program dynamically allocates an array of integers.
Enter the number of elements in the array: 10
Enter an integer: 1
Enter an integer: 2
Enter an integer: 3
Enter an integer: 4
Enter an integer: 5
Enter an integer: 6
Enter an integer: 7
Enter an integer: 8
Enter an integer: 9
Enter an integer: 10

The elements of the array are:
1 2 3 4 5 6 7 8 9 10

The elements of the array after reallocation are:
1 2 3 4 5
```

14.8 Write a program that takes two command-line arguments that are file names, reads the characters from the first file one at a time and writes the characters in reverse order to the second file.

ANS:

```
/* Exercise 14.8 Solution */
#include <stdio.h>

/* function prototype */
void reverseFile( FILE *inPtr, FILE *outPtr );

int main( int argc, int *argv[] )
{
   FILE *inFilePtr;  /* input file pointer */
   FILE *outFilePtr; /* output file pointer */

   /* tell user if invalid arguments */
   if ( argc != 3 ) {
      printf( "Usage: copy infile outfile\n" );
   } /* end if */
   else {

      /* exit program if input file cannot be opened */
      if ( ( inFilePtr = fopen( argv[ 1 ], "r" ) ) != NULL ) {

         /* exit program if output file cannot be opened */
         if ( ( outFilePtr = fopen( argv[ 2 ], "w" ) ) != NULL ) {
            reverseFile( inFilePtr, outFilePtr );
         } /* end if */
         else {
            printf( "File \"%s\" could not be opened\n", argv[ 2 ] );
         } /* end else */

      } /* end if */
      else {
         printf( "File \"%s\" could not be opened\n", argv[ 1 ] );
      } /* end else */

   } /* end else */

   return 0; /* indicate successful termination */

} /* end main */

/* function that writes characters in reverse order */
void reverseFile( FILE *inPtr, FILE *outPtr )
{
   int c; /* current character */

   /* if not end of file */
   if ( ( c = fgetc( inPtr ) ) != EOF ) {
      reverseFile( inPtr, outPtr );
   } /* end if */

   fputc( c, outPtr ); /* write character to output file */
} /* end function reverseFile */
```

```
.stnemugra enil dnammoc eht rof ecnetnes tset a si sihT
```

14.9 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 `printf` statements:

```
printf( "*" );
printf( " " );
printf( "\n" );
```

ANS:

```
/* Exercise 14.9 Solution */
#include <stdio.h>

int main()
{
   int size;     /* length of square sides */
   int row = 0;  /* number of rows */
   int col;      /* number of columns */

   /* obtain length of side of square from user */
   printf( "Enter the side length of the square: " );
   scanf( "%d", &size );

   start: /* label */
      ++row;
      printf( "\n" );

      /* if all rows have been made end program */
      if ( row > size ) {
         goto end;
      } /* end if */

      col = 1; /* set column variable to first character of line */

      innerLoop: /* label */

         /* if all columns have been displayed return to top of loop */
         if ( col > size ) {
            goto start;
         } /* end if */

         /* display stars and spaces in appropriate positions */
         if ( row == 1 || row == size || col == 1 || col == size ) {
            printf( "*" );
         } /* end if */
         else {
            printf( " " );
         } /* end else */

         ++col; /* increment column */
         goto innerLoop; /* continue displaying columns */

   end: /* label */

   return 0; /* indicate successful termination */

} /* end main */
```

15

C++ as a “Better C”: Solutions

SOLUTIONS

15.5 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.

ANS:

```
// Exercise 15.5 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
```

15.6 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).

ANS:

```
// Exercise 15.6 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
```

15.7 What is the purpose of the unary scope resolution operator?

ANS: The unary score 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 variable has the same name.

15.8 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.

ANS:

```
// Exercise 15.8 Solution
#include <iostream>

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

template < class T >
void minimum( 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()
{
   minimum( 7, 54 );      // integers
   minimum( 4.35, 8.46 ); // doubles
   minimum( 'g', 'T' );   // characters

   return 0;
}
```

```
7 is smaller than 54
4.35 is smaller than 8.46
T is smaller than g
```

15.9 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.

ANS:

```cpp
// Exercise 15.9 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 $
```

15.10 Determine whether the following program segmentss 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: The 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 return types differ, the compiler generates an error message.

16

C++ Classes and Data Abstraction: Solutions

SOLUTIONS

16.3 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.

16.4 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.

ANS:

```
// p16_4.H
#ifndef p16_4_H
#define p16_4_H

class Time {
public:
   Time();
   void setHour( int );
   void setMinute( int );
   void setSecond( int );
   int getHour( void ) const;
   int getMinute( void ) const;
   int getSecond( void ) const;
   void printStandard( void ) const;
private:
   int hour;
   int minute;
   int second;
};

#endif
```

```
// p16_4.cpp
// member function definitions for p16_4.cpp
#include <iostream.h>

using std::cout;

```

```
#include <ctime>
#include "p16_4.h"

Time::Time()
{
   long int totalTime;             // time in seconds since 1970
   int currentYear = 1994 - 1970; // current year
   double totalYear;               // current time in years
   double totalDay;                // days since beginning of year
   double day;                     // current time in days
   long 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.

   totalTime = time( NULL );
   divisor = ( 60.0 * 60.0 * 24.0 * 365.0 );
   totalYear = totalTime / divisor - currentYear;
   totalDay = 365 * totalYear;     // leap years ignored
   day = totalDay - ( int ) totalDay;

   setHour( day * 24 + timeShift );
   setMinute( ( day * 24 - ( int )( day * 24 ) ) * 60 );
   setSecond( ( minute * 60 - ( int )( minute * 60 ) ) * 60 );
}

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() const {  return hour; }

int Time::getMinute() const {  return minute; }

int Time::getSecond() const {  return second; }

void Time::printStandard() const
{
   cout << ( ( hour % 12 == 0 ) ? 12 : hour % 12 ) << ":"
        << ( minute < 10 ? "0" : "" ) << minute << ":"
<< ( second < 10 ? "0" : "" ) << second
<< ( hour < 12 ? " AM" : " PM" );
}
```

```
// driver for p16_4.cpp
#include "p16_4.h"

int main( void )
{
   Time t;

   t.printStandard();

   return 0;
}
```

```
12:15:00 PM
```

16.5 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.

ANS:

```
// 16.5.H
#ifndef p16_5_H
#define p16_5_H

class Complex {
public:
   Complex( double = 0.0, double = 0.0 ); // default constructor
   void addition( const Complex & );
   void subtraction( const Complex & );
   void printComplex( void ) const;
   void initialize( const double, const double );
private:
   double realPart;
   double imaginaryPart;
};

#endif
```

```
// p16_5M.cpp
// member function definitions for p16_5.cpp
#include <iostream>

using std::cout;

#include "p16_5.h"

Complex::Complex( double real, double imaginary )
{
   initialize( 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 ) const
{
   cout << "( " << realPart << ", " << imaginaryPart << " )";
}
```

```

void Complex::initialize( const double rp, const double ip )
{
   realPart = rp;
   imaginaryPart = ip;
}
```

```
// driver for p16_5.cpp
#include <iostream>

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

#include "p16_5.h"

int main( void )
{
   Complex b( 1, 7 ), c( 9, 2 );

   b.printComplex();
   cout << " + ";
   c.printComplex();
   cout << " = ";
   b.addition( c );
   b.printComplex();

   cout << "\n";
   b.initialize( 10,1 );   // reset realPart and imaginaryPart
   c.initialize( 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 )
```

16.6 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 values 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 floating-point format.

ANS:

```
// P16_6.H
#ifndef P16_6_H
#define P16_6_H

class RationalNumber {
public:
   RationalNumber( int = 0, int = 1 );  // default constructor
   RationalNumber addition( const RationalNumber& );
   RationalNumber subtraction( const RationalNumber& );
   RationalNumber multiplication( const RationalNumber& );
   RationalNumber division( RationalNumber& );
   void printRational( void ) const;
   void printRationalF( void ) const;
private:
   int numerator;
   int denominator;
   void reduction( void );
};

#endif
```

```
// P16_6M.cpp
// member function definitions for p16_6.cpp
#include <iostream>

using std::cout;

#include "p16_6.h"

RationalNumber::RationalNumber( int n, int d )
{
   numerator = n;
   denominator = d;
}

RationalNumber RationalNumber::addition( const RationalNumber &a )
{
   RationalNumber t;

   t.numerator = a.numerator * denominator + a.denominator * numerator;

   t.denominator = a.denominator * denominator;
   t.reduction();

   return t;
}
```

```
RationalNumber RationalNumber::subtraction( const RationalNumber &s )
{
   RationalNumber t;

   t.numerator = s.denominator * numerator - denominator * s.numerator;
   t.denominator = s.denominator * denominator;
   t.reduction();

   return t;
}

RationalNumber RationalNumber::multiplication( const RationalNumber &m )
{
   RationalNumber t;

   t.numerator = m.numerator * numerator;
   t.denominator = m.denominator * denominator;
   t.reduction();

   return t;
}

RationalNumber RationalNumber::division( RationalNumber &v )
{
   RationalNumber t;

   t.numerator = v.denominator * numerator;
   t.denominator = denominator * v.numerator;
   t.reduction();

   return t;
}

void RationalNumber::printRational( void ) const
{
   if ( denominator == 0 )
      cout << "\nDIVIDE BY ZERO ERROR!!!\n";
   else if ( numerator == 0 )
      cout << 0;
   else
      cout << numerator << "/" << denominator;
}

void RationalNumber::printRationalF( void ) const
{  cout << ( double ) numerator / denominator; }

void RationalNumber::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 P16_6.cpp
#include <iostream>

using std::cout;

#include "p16_6.h"

int main( void )
{
   RationalNumber 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.printRationalF();
   cout << "\n\n";

   c.printRational();
   cout << " - ";
   d.printRational();
   x = c.subtraction( d );
   cout << " = ";
   x.printRational();
   cout << "\n";
   x.printRational();
   cout << " = ";
   x.printRationalF();
   cout << "\n\n";

   c.printRational();
   cout << " x ";
   d.printRational();
   x = c.multiplication( d );
   cout << " = ";
   x.printRational();
   cout << "\n";
   x.printRational();
   cout << " = ";
   x.printRationalF();
   cout << "\n\n";

   c.printRational();
   cout << " / ";
   d.printRational();
   x = c.division( d );
   cout << " = ";
   x.printRational();
   cout << "\n";
   x.printRational();
   cout << " = ";
   x.printRationalF();
   cout << "\n";

   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
```

16.7 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.

ANS:

```
// P16_7.H
#ifndef P16_7_H
#define P16_7_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
```

```
// P16_7M.cpp
// member function definitions for p16_7.cpp

#include "p16_7.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 p16_7.cpp
#include <iostream>

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

#include <iomanip>

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

#include "p16_7.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
```

16.8 Create a more sophisticated `Rectangle` class than the one you created in Exercise 16.7. 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.

ANS:

```
// P16_8.H
#ifndef P16_8_H
#define P16_8_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
```

```
// P16_8M.cpp
// member function definitions for p16_8.cpp
#include <iostream>

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

#include <iomanip>

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

#include <cmath>

#include "p16_8.h"

Rectangle::Rectangle( double *a, double *b, double *c, double *d )
   { setCoord( a, b, c, d ); }

void Rectangle::setCoord( double *p1, double *p2, doubtle *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 p16_8.cpp

#include "p16_8.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!
```

16.9 Modify the `Rectangle` class of Exercise 16.8 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.

ANS:

```
// P16_9.H
#ifndef P16_9_H
#define P16_9_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
```

```
// P16_9M.cpp
// member function definitions for p16_9.cpp
#include <iostream>

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

#include <iomanip>

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

#include <cmath>

#include "p16_9.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 p16_9.cpp

#include "p16_9.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

..........................
..........................
..........................
..........................
..........................
............*******.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*?????*.......
............*******.......
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
..........................
```

16.10 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`.

16.11 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 moves. Each 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 three-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!).

ANS:

```
// p16_11.H
#ifndef P16_11_H
#define P16_11_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
```

```
// P16_11M.cpp
// member function definitions for p16_9.cpp
#include <iostream>

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

#include <iomanip>

using std::setw;

#include "p16_11.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 p16_11.cpp
#include "p16_11.h"

int main()
{
   TicTacToe g;
   g.makeMove();

   return 0;
}
```

```
   0    1    2

0     |     |
 _____|_____|_____

1     |     |
 _____|_____|_____

2     |     |

Player X enter move: 0 0
   0    1    2

0  X  |     |
 _____|_____|_____

1     |     |
 _____|_____|_____

2     |     |

...
Player O enter move: 0 2
   0    1    2

0  X  |     |  O
 _____|_____|_____

1  X  |  O  |
 _____|_____|_____

2     |     |

Player X enter move: 2 0
   0    1    2

0  X  |     |  O
 _____|_____|_____

1  X  |  O  |
 _____|_____|_____

2  X  |     |

Player X wins!
```

17

C++ Classes: Part II: Solutions

SOLUTIONS

17.3 Compare and contrast dynamic memory allocation using the C++'s `new` and `delete` operators, 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 objects(s) begin deleted.

17.4 Explain the notion of friendship in C++. Explain the negative aspects of friendship as described in the text.

ANS: Functions that are declared as `friends` 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 implementation details that are supposed to be hidden.

17.5 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 both can be called with no arguments.

17.6 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.

17.7 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 18, 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.

ANS:

```
// P17_07.H
#ifndef p17_07_H
#define p17_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
```

```
// P17_07M.cpp
// member function definitions for p17_07.cpp
#include <iostream>

using std::cout;

#include <ctime>

#include "p17_07.h"

// Date constructor
Date::Date()
{
   long int totalTime;
   double totalYear;
   long double divisor;

   totalTime = time( NULL );      // time in seconds since 1970
   divisor = ( 60.0 * 60.0 * 24.0 * 365.25 );  //number of seconds in a year
   totalYear = totalTime / divisor + 1970;
   year = ( int ) totalYear;
   totalYear -= year;
   day = ( int ) ( 365 * totalYear );
   month = 1;

   while ( day - days( month + 1 ) > 0 )
   day -= days( month++ );
}

```

```
// 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 );

   int m = 1;

   for ( ; 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 p17_07.cpp
#include <iostream>

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

#include "p17_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
7/26/2000
9/1/1998

185 1998
86 1999
207 2000
244 1998

July 4, 1998
March 27, 1999
July 26, 2000
September 1, 1998
```

17.8 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 `static` 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.

ANS:

```
// P17_08.H
#ifndef P17_08_H
#define P17_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
```

```
// P17.08M.cpp
// Member function defintions for p17_08.cpp
#include "p17_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 );
}
```

```cpp
// driver for p17_08.cpp
#include <iostream>

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

#include <iomanip>

using std::setw;

#include "p17_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
```

17.9 It would be perfectly reasonable for the `Time` class of Fig. 17.8 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. 17.8 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.

18

C++ Operator Overloading: Solutions

SOLUTIONS

18.6 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 >>.

18.7 The C++ operators that cannot be overloaded are __________, __________, __________,___________ and __________.
ANS: `sizeof`, `.`, `?:`, `.*`, and `::`.

18.8 *(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++`.

18.9 Overload the subscript operator to return the largest element of a collection, the second largest, the third largest, etc.

18.10 Consider class `Complex` shown in Fig. 18.5. 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. 18.5: complex1.h
// Definition of class Complex
#ifndef COMPLEX1_H
#define COMPLEX1_H

class Complex {
public:
```

Fig. 18.5 A complex number class—`complex1.h`.

```
   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. 18.5 A complex number class—`complex1.h`.

```
// Fig. 18.5: 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. 18.5 A complex number class–`complex1.cpp` .

```
// Fig. 18.5: fig18_05.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;
}
```

```
x: (0, 0)
y: (4.3, 8.2)
z: (3.3, 1.1)

x = y + z:
(7.6, 9.3) = (4.3, 8.2) + (3.3, 1.1)

x = y - z:
(1, 7.1) = (4.3, 8.2) - (3.3, 1.1)
```

Fig. 18.5 A complex number class–`fig18_05.cpp` .

ANS:

```
// P18_10.H
#ifndef P18_10_H
#define P18_10_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
```

```
// P18_10M.cpp
// member function definitions for p18_10.cpp
#include "p18_10.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 p18_10.cpp
#include <iostream>

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

#include "p18_10.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
? 22 + 8i
x: 0 + 0i
y: 4.3 + 8.2i
z: 3.3 + 1.1i
k: 22 + 8i

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 != 22 + 8i

22 + 8i == 22 + 8i
```

18.11 The program of Fig. 18.3 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. 18.3 with the overloaded stream-insertion `operator<<` as a member function and try the two preceding statements in the program to prove that they work.

19

C++ Inheritance: Solutions

SOLUTIONS

19.2 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:
- `HandleBars`
- `Seat`
- `Frame`
- `Wheels` composed of:
  - `Tires`
  - `Rims` composed of:
    - `Spokes`
- `Pedals`
- `Chain` composed of:
  - `Links`
- `Brakes` composed of:
  - `Wires`
  - `Brickbats`
  - `Breadlines`

Classes that can be derived from `Bicycle` are `Unicycle`, `Tricycle`, `Tandem Bicycle`, etc.

19.3 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.

19.4 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 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.

19.5 (True/False) A derived class is often called a subclass because it represents a subset of its base class (i.e., a derived class is 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.

19.6 (True/False) A derived-class object is also an object of that derived class's base class.

ANS: True.

19.7 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 access `private` members. This can result in additional function calls—which can decrease performance.

19.8 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. 19.10 (and the supporting classes) to use composition rather than inheritance. After you do this, reassess the relative merits of the two approaches both for the `Point`, `Circle`, `Cylinder` problem and for object-oriented programs in general.

ANS:

```cpp
// P19_08.H
#ifndef P19_08_H
#define P19_08_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
```

```cpp
// P19_08PM.cpp
// Member functions for class Point
#include <iostream>

using std::cout;
using std::ostream;

#include "p19_08.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() << ']'; }
```

```
// P19_08C.H
#ifndef P19_08C_H
#define P19_08C_H
#include "P19_08.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

```

```
// P19_08CM.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 "P19_08c.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 );
}
```

```
// P19_08CY.H
#ifndef P19_08CY_H
#define P19_08CY_H
#include "P19_08.h"
#include "P19_08c.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
```

```
// P19_08CYM.cpp
// Member function definitions for class Cylinder.
#include <iostream>

using std::cout;
using std::ostream;

#include "p19_08cy.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';
}
```

```
// P19_08.cpp
#include <iostream>

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

#include "P19_08.h"
#include "P19_08c.h"
#include "P19_08cy.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
```

19.9 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.

ANS: No. The "is a" relationship assumes that everything belongs to the base class object belongs to the derived class object and also assumes that all functionality of the base class is present in the derived class object.

19.10 Study the inheritance hierarchy of Fig. 19.2. For each class, indicate some common attributes and behaviors consistent with the hierarchy. Add some other classes (`UndergraduateStudent`, `GraduateStudent`, `Freshman`, `Sophomore`, `Junior`, `Senior`, etc.) to enrich the hierarchy.

ANS:

```
CommunityMember
   Employee
      Staff
         Maintenance
         Janitorial
      Faculty
         Administrator
         Professor
            TenuredProfessor
   Student
      Graduate
         MasterCandidate
         DoctoralCandidate
      Undergraduate
         Freshman
         Sophomore
         Junior
         Senior
```

19.11 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.

ANS:

```
// P19_11.H
#ifndef P19_11_H
#define P19_11_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
```

```
// P19_11PM.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 "p19_11.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 );
}
```

```
// P19_11Q.H
#ifndef P19_11Q_H
#define P19_11Q_H
#include "p19_11.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
```

```
// P19_11QM.cpp
// member functions for class Quadrilateral
#include "p19_11q.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";
}
```

```
// P19_11T.H
#ifndef P19_11T_H
#define P19_11T_H
#include "p19_11q.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
```

```
// P19_11TM.cpp
// member function definitions for class Trapazoid
#include "p19_11t.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";
}
```

```
// P19_11PA_H
#ifndef P19_11PA_H
#define P19_11PA_H
#include "p19_11q.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
```

```
// P19_11PAM.cpp
#include "p19_11q.h"
#include "p19_11pa.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();   }
```

```
// P19_11R.H
#ifndef P19_11R_H
#define P19_11R_H
#include "p19_11pa.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
```

```
// P19_11RM.cpp
#include "p19_11r.h"
#include "p19_11pa.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(); }
```

```
// P19_11RH.H
#ifndef P19_11RH_H
#define P19_11RH_H
#include "p19_11pa.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
```

```
//P19_11HM.cpp
#include "p19_11rh.h"
#include "p19_11pa.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;
}
```

```
// P19_11S.H
#ifndef P19_11S_H
#define P19_11S_H
#include "p19_11pa.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
```

```
// P19_11SM.cpp
#include "p19_11s.h"
#include "p19_11pa.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;
}
```

```
// P19_11.cpp
#include "p19_11.h"
#include "p19_11q.h"
#include "p19_11t.h"
#include "p19_11pa.h"
#include "p19_11rh.h"
#include "p19_11r.h"
#include "p19_11s.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)
```

19.12 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 20 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
         OneSheetedHyperboloid
          TwoSheetedHyperboloid
      Plane
```

20

C++ Virtual Functions and Polymorphism: Solutions

SOLUTIONS

20.2 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.

20.3 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.

20.4 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.

20.5 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 `case`s for a specified type. [*Note*: `switch` logic includes `if/else` structures which are more flexible than the `switch` structure.]

20.6 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.

20.7 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 amount 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 capabilities of the program).

20.8 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 base class must provide definitions for the inherited pure `virtual` functions in order to avoid becoming an abstract base class.

20.9 (True/False) All `virtual` functions in an abstract base class must be declared as pure `virtual` functions.
ANS: False.

20.10 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`).

20.11 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.

20.12 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.

20.13 Develop a basic graphics package. Use the `Shape` class inheritance hierarchy from Chapter 19. 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.

20.14 In Exercise 19.12, 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.

ANS:

```
// 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';
}
```

```
// 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';
}
```

```
// 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
```

```
// 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';
}
```

```
// Exercise 20.14 solution
// Driver to test Shape hierarchy
#include <iostream>

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

#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
```

21

C++ Stream Input/Output: Solutions

EXERCISES

21.6 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 )
     << 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).

21.7 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.

ANS:

```
// Exercise 21.7 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 octal format\n";
   cin >> setiosflags( ios::showbase ) >> 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;

   cout << "\nEnter an integer in hexadecimal format\n";
   cin >> setiosflags( ios::showbase ) >> 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;

   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 octal format
010
As a decimal number 8
As an octal number 010
As a hexadecimal number 0x8

Enter an integer in hexadecimal format
0x10
As a decimal number 16
As an octal number 020
As a hexadecimal number 0x10
```

21.8 Write a program that prints pointer values using casts to all the integer data types. Which ones print strange values? Which ones cause errors?

ANS:

```
// Exercise 21.8 Solution
#include <iostream>

using std::cout;

int main()
{
   char *string = "test";

   cout << "Value of string is                           : "
        << string << '\n';

   cout << "Value of static_cast<void *>( string ) is    : "
        << static_cast<void *>( string ) << '\n';

        // The Following generate errors.
        // reinterpret_cast will allow this type of casting

/* cout << "Value of static_cast<char>(string) is      : "
        << static_cast<char>( string ) << '\n';

   cout << "Value of static_cast<int>(string) is       : "
        << static_cast<int>( string ) << '\n';

   cout << "Value of static_cast<long>(string) is      : "
        << static_cast<long>( string ) << '\n';

   cout << "Value of static_cast<short>(string) is     : "
        << static_cast<short>( string ) << '\n';

   cout << "Value of static_cast<unsigned>(string) is  : "
        << static_cast<unsigned>( string )
*/

   return 0;
}
```

```
Value of string is                           : test
Value of static_cast<void *>( string ) is    : 0041A178
```

21.9 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?

ANS:

```cpp
// Exercise 21.9 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
```

21.10 Write a program that prints the value `100.453627` rounded to the nearest digit, tenth, hundredth, thousandth and ten thousandth.

ANS:

```
// Exercise 21.10 Solution
#include <iostream>

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

#include <iomanip>

using std::setw;
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
```

21.11 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.

ANS:

```
// Exercise 21.11 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
```

21.12 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.

ANS:

```
// Exercise 21.12 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
              5                -15.000
...
            206                +96.667
            207                +97.222
            208                +97.778
            209                +98.333
            210                +98.889
            211                +99.444
            212               +100.000
```

21.13 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?

ANS:

```
// Exercise 21.13 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: "vaccum"
String is "vaccum"
Enter a string: 'grape'
String is 'grape'
Enter a string: water
String is water
```

21.14 In Fig. 18.3, 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.

ANS:

```
// P21_14.H
#ifndef P21_14_H
#define P21_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 <string>
using std::string;

#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
```

```
// P21_14M.cpp
// member function definition definition for p21_14.cpp
#include "p21_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() ) {
      int loop = 0;
      for ( ; 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 p21_14.cpp
#include "p21_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:
abcdefghijk

Invalid phone number entered.
```

21.15 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.

ANS:

```
// P21_15.H
#ifndef P21_15_H
#define P21_15_H
#include <iostream.h>

class Point {
   friend ostream &operator<<( ostream&, Point& );
   friend istream &operator>>( istream&, Point& );
private:
   int xCoordinate;
   int yCoordinate;
};

#endif
```

```
// P21_15M.cpp
// member function definitions for p21_15.cpp
#include "p21_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 p21_15.cpp
#include "p21_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)
```

21.16 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.

ANS:

```
// P21_16.H
#ifndef P21_16_H
#define P21_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
```

```
// P21_16M.cpp
// member function definitions for p21_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 "p21_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 p21_16.cpp
#include <iostream>

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

#include "p21_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
```

21.17 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.

ANS:

```
// Exercise 21.17 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            '
     40      050            0x28            (
     41      051            0x29            )
     42      052            0x2a            *
     43      053            0x2b            +
...
    120     0170            0x78            x
    121     0171            0x79            y
    122     0172            0x7a            z
    123     0173            0x7b            {
    124     0174            0x7c            |
    125     0175            0x7d            }
    126     0176            0x7e            ~
```

21.18 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?

ANS:

```
// Exercise 21.18 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: *
```

21.19 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 non-whitespace 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 preceded by whitespace as input. Print the character that was input to confirm that a whitespace character was not input.

22

C++ Templates: Solutions

SOLUTIONS

22.3 Use a nontype parameter `numberOfElements` and a type parameter `elementType` to help create a template for the `Array` class we developed in Chapter 18, “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.

ANS:

```
#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
{
   int i = 0;
   for ( ; i < size; ++i ) {
      cout << ptr[ i ] << ' ';

      if ( ( i + 1 ) % 10 == 0 )
         cout << '\n';
   }

   if ( i % 10 != 0 )
      cout << '\n';
}

#endif
```

```
// Exercise 22.3 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 doubleArray 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 1.14 1.22 9.11

The values in the doubleArray are:
1.12 1.13 1.14 1.22 9.11
```

22.4 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 >`.

22.5 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.

22.6 What performance problem can result from using class templates?
ANS: There can be a tremendous proliferation of code in the program due to many copies of code generated by the compiler.

22.7 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.

22.8 Explain why you might use the statement

```
Array< Employee > workerList( 100 );
```

in a C++ program.
ANS: Declares an `Array` object to store `Employee` objects and passes `100` to the constructor.

22.9 Review your answer to Exercise 22.8. 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 called.

22.10 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`.

22.11 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.

22.12 Describe how to provide a class for a specific type to override the class template for that type.

22.13 Describe the relationship between class templates and inheritance.

22.14 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: Function `f1` is a `friend` of all template classes instantiated from class template `C1`.
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 > & );
```

is a `friend` of the class `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 >`.

22.15 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)?
ANS: For `static` members of a class template, each template class instantiated receives its own copy of all the `static` members. Then all objects instantiated for a given template class can access that particular template class' `static` members

23

C++ Exception Handling: Solution

SOLUTIONS

23.20 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.

23.21 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.

23.22 Under what circumstances would you use the following statement?

```
catch(...) { throw; }
```

ANS: The proceeding statement is used to `catch` any exception and rethrow it for handling by an exception handler in a function within the call stack.

23.23 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.

23.24 List the advantages of exception handling over conventional means of error processing.

23.25 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.

ANS:

```
// Exercise 23.25 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;
}
```

```
DerivedException
```

23.26 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`.

ANS:

```
// Exercise 23.26 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[ 1000000 ];
         cout << "Allocated 1000000 long doubles in ptr[ "
              << i << " ]\n";
      }
   }
   catch ( bad_alloc ex ) {
      cerr << "Memory Allocation Failed.\n";
      exit( EXIT_FAILURE );
   }

   return 0;
}
```

```
Allocated 1000000 long doubles in ptr[ 0 ]
Allocated 1000000 long doubles in ptr[ 1 ]
Allocated 1000000 long doubles in ptr[ 2 ]
Allocated 1000000 long doubles in ptr[ 3 ]
Allocated 1000000 long doubles in ptr[ 4 ]
Allocated 1000000 long doubles in ptr[ 5 ]
Allocated 1000000 long doubles in ptr[ 6 ]
Allocated 1000000 long doubles in ptr[ 7 ]
Allocated 1000000 long doubles in ptr[ 8 ]
Memory Allocation Failed.
```

24

Introduction to Java Applications and Applets: Solutions

SOLUTIONS

24.8 Fill in the blanks in each of the following:

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

ANS: Comments.

b) An input dialog capable of receiving input from the user is displayed with method __________ of class __________.

ANS: `showInputDialog`, `JOptionPane`.

24.9 Write Java statements that accomplish each of the following:

a) Display the message `"Enter two numbers"` using class `JOptionPane`.

ANS: `JoptionPane.showMessageDialog( null, “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: `// This program performs a simple payroll calculation.`

24.10 What displays in the message dialog when each of the following Java statements is performed? Assume `x = 2` and `y = 3`.

a) `JOptionPane.showMessageDialog( null, "x = " + x );`

ANS: `x = 2`

b) `JOptionPane.showMessageDialog( null, "The value of x + x is " + ( x + x ) );`

ANS: `The value of x + x is 4`

c) `JOptionPane.showMessageDialog( null, "x =" );`

ANS: `x =`

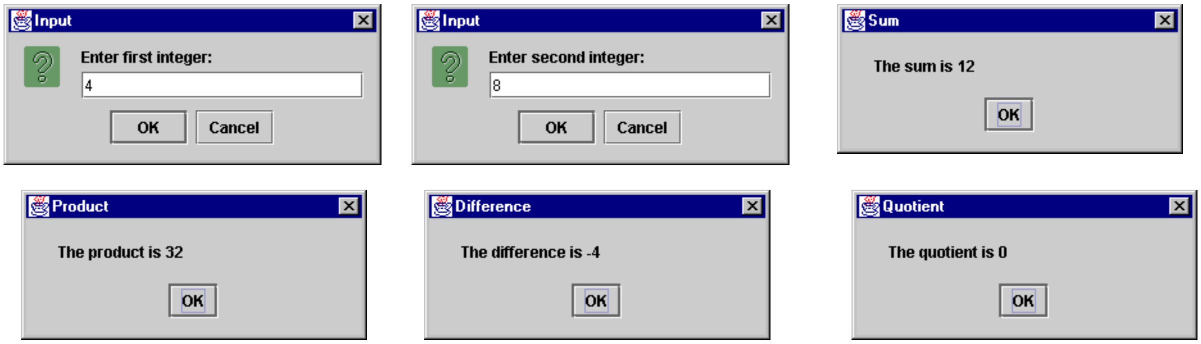
d) `JOptionPane.showMessageDialog(null, ( x + y ) + " = " + ( y + x ) );`

ANS: `5 = 5`

24.11 Write an application 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. Use the techniques shown in Fig. 24.6.

ANS:

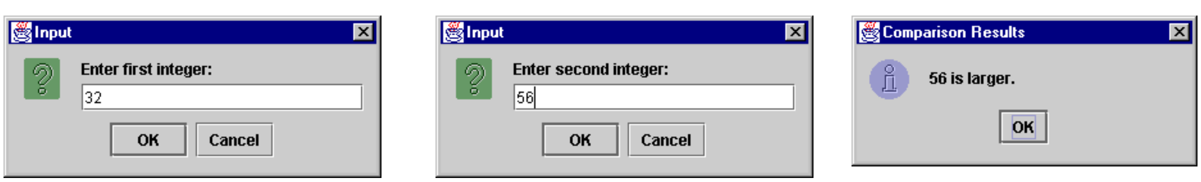
```java
// Exercise 24.11 Solution
// Calculate.java
// Program prints the sum, product, difference and quotient of
// the two numbers.

import javax.swing.JOptionPane;

public class Calculate {
   public static void main( String args[] )
   {
       String firstNumber,    // first string entered by user
             secondNumber;    // second string entered by user
       int number1,           // first number
           number2;           // second number
       int sum;
       int product;
       int difference;
       int quotient;

       // read first number from user as a string
       firstNumber =
          JOptionPane.showInputDialog( "Enter first integer:" );

       // read second number from user as a string
       secondNumber =
          JOptionPane.showInputDialog( "Enter second integer:" );

       // convert numbers from type String to type int
       number1 = Integer.parseInt( firstNumber );
       number2 = Integer.parseInt( secondNumber );

       // calculate
       sum        = number1 + number2;
       product    = number1 * number2;
       difference = number1 - number2;
       quotient   = number1 / number2;

       // Display results
       JOptionPane.showMessageDialog(
          null, "The sum is " + sum, "Sum",
       JOptionPane.PLAIN_MESSAGE );
       JOptionPane.showMessageDialog(
          null, "The product is " + product, "Product",
       JOptionPane.PLAIN_MESSAGE );
       JOptionPane.showMessageDialog(
          null, "The difference is " + difference, "Difference",
       JOptionPane.PLAIN_MESSAGE );
       JOptionPane.showMessageDialog(
          null, "The quotient is " + quotient, "Quotient",
       JOptionPane.PLAIN_MESSAGE );

       System.exit( 0 );
   }
}
```


Input
Enter first integer:
4
OK
Cancel
Input
Enter second integer:
8
OK
Cancel
Sum
The sum is 12
OK
Product
The product is 32
OK
Difference
The difference is -4
OK
Quotient
The quotient is 0
OK

24.12 Write an application that asks the user to enter two integers, obtains the numbers from the user and displays the larger number followed by the words "`is larger`" in an information message dialog. If the numbers are equal, print the message "`These numbers are equal`." Use the techniques shown in Fig. 24.6.

ANS:

```java
// Exercise 24.12 Solution
// Larger.java
// Program determines the larger of two numbers
import javax.swing.JOptionPane;

public class Larger {
   public static void main( String args[] )
   {
      String firstNumber,    // first string entered by user
             secondNumber;   // second string entered by user
       int number1,          // first number to compare
           number2;          // second number to compare

       // read first number from user as a string
       firstNumber =
          JOptionPane.showInputDialog( "Enter first integer:" );

       // read second number from user as a string
       secondNumber =
          JOptionPane.showInputDialog( "Enter second integer:" );

       // convert numbers from type String to type int
       number1 = Integer.parseInt( firstNumber );
       number2 = Integer.parseInt( secondNumber );

       String result;         // a string containing the output
       if ( number1 > number2 )
          result = number1 + " is larger.";
       else if ( number1 < number2 )
          result = number2 + " is larger.";
       else
          result = "These numbers are equal.";

       // Display results
       JOptionPane.showMessageDialog(
          null, result, "Comparison Results",
          JOptionPane.INFORMATION_MESSAGE );

       System.exit( 0 );
   }
}
```

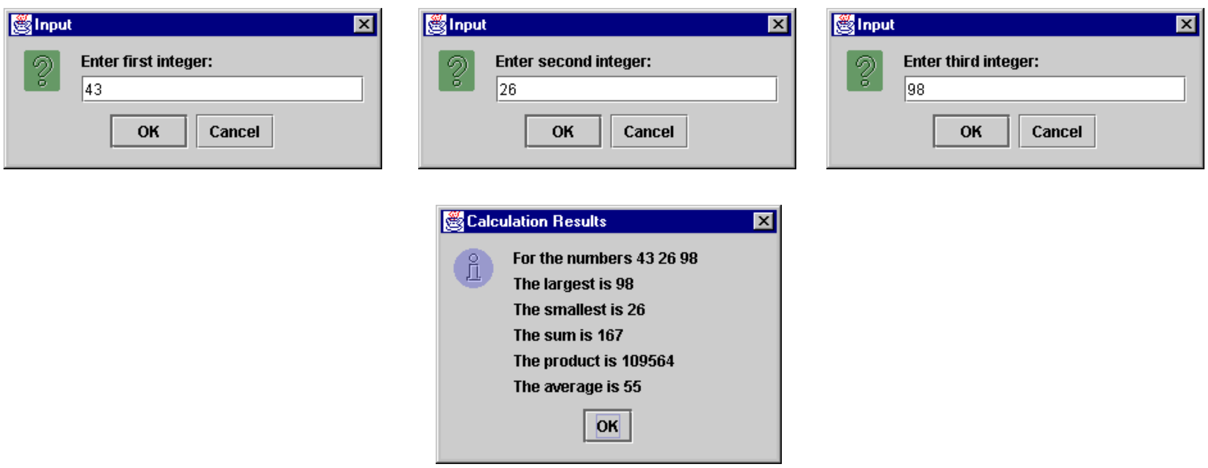


24.13 Write an application that inputs three integers from the user and displays the sum, average, product, smallest and largest of these numbers in an information message dialog. Use the GUI techniques shown in Fig. 24.6. Note: The average calculation in this exercise should result in an integer representation of the average. So, if the sum of the values is 7, the average will be 2 not 2.3333...

ANS:

```
// Exercise 24.13 Solution
// Calculate.java
// Program make simple calculations on three integers.
import javax.swing.JOptionPane;

public class Calculate {
   public static void main( String args[] )
   {
      String firstNumber,    // first string entered by user
             secondNumber,   // second string entered by user
             thirdNumber;    // third string entered by user
      String result;
      int number1,          // first number
          number2,          // second number
          number3;          // third number

      int sum;
      int largest;
      int smallest;
      int product;
      int average;

       // read first number from user as a string
       firstNumber =
          JOptionPane.showInputDialog( "Enter first integer:" );

       // read second number from user as a string
       secondNumber =
          JOptionPane.showInputDialog( "Enter second integer:" );

       // read third number from user as a string
       thirdNumber =
          JOptionPane.showInputDialog( "Enter third integer:" );

       // convert numbers from type String to type int
       number1 = Integer.parseInt( firstNumber );
       number2 = Integer.parseInt( secondNumber );
       number3 = Integer.parseInt( thirdNumber );

       if ( number1 > number2 ) {
           largest = number1;
           smallest = number2;
       }
       else {
           largest = number2;
           smallest = number1;
       }

       if ( number3 > largest )
           largest = number3;

       if ( number3 < smallest )
           smallest = number3;

       sum = number1 + number2 + number3;
       product = number1 * number2 * number3;
```

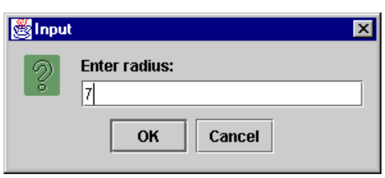
```
       average = sum / 3;

       result = "For the numbers " + number1 + " "
                + number2 + " "
                + number3 + "\n" +
                "The largest is " + largest + "\n" +
                "The smallest is " + smallest + "\n" +
                "The sum is " + sum + "\n" +
                "The product is " + product + "\n" +
                "The average is " + average + "\n";

       // Display results
       JOptionPane.showMessageDialog(
          null, result, "Calculation Results",
          JOptionPane.INFORMATION_MESSAGE );

       System.exit( 0 );
   }
}
```

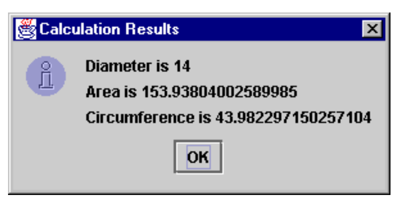


24.14 Write an application that inputs from the user the radius of a circle and prints the circle's diameter, circumference and area. Use the constant value 3.14159 for π. Use the GUI techniques shown in Fig. 24.6. [*Note:* You may also use the predefined constant `Math.PI` for the value of π. This constant is more precise than the value 3.14159. Class `Math` is defined in the `java.lang` package, so you do not need to `import` it.] Use the following formulas (r is the radius): $diameter = 2r$, $circumference = 2\pi r$, $area = \pi r^2$.

ANS:

```
// Exercise 24.14 Solution
// Circle.java
// Program calculate the area, circumference, and diameter for a circle
import javax.swing.JOptionPane;

public class Circle {
   public static void main( String args[] )
   {
      String input,       // string entered by user
             result;      // output display string
      int radius;         // radius of circle

      // read from user as a string
      input =
         JOptionPane.showInputDialog( "Enter radius:" );

      // convert number from type String to type int
      radius = Integer.parseInt( input );

      result = "Diameter is " + ( 2 * radius ) +
               "\nArea is " + ( Math.PI * radius * radius ) +
               "\nCircumference is " + ( 2 * Math.PI * radius );

     // Display results
      JOptionPane.showMessageDialog(
         null, result, "Calculation Results",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```





24.15 Write an application that displays in the command window a box, an oval, an arrow and a diamond using asterisks (*) as follows:

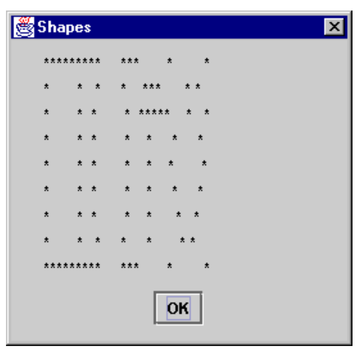
```
*********          ***              *              *
*       *        *     *           ***            * *
*       *       *       *         *****          *   *
*       *       *       *           *           *     *
*       *       *       *           *          *       *
*       *       *    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

ANS:

```java
// Exercise 24.15 Solution
// Shapes.java
// Program draws four shapes to the command window.

public class Shapes {
   public static void main( String args[] )
   {
      System.out.println( "*********     ***       *        *    " );
      System.out.println( "*       *   *     *    ***      * *   " );
      System.out.println( "*       *  *       *  *****    *   *  " );
      System.out.println( "*       *  *       *    *     *     * " );
      System.out.println( "*       *  *       *    *    *       *" );
      System.out.println( "*       *  *       *    *     *     * " );
      System.out.println( "*       *  *       *    *      *   *  " );
      System.out.println( "*       *   *     *     *       * *   " );
      System.out.println( "*********     ***       *        *    " );
   }
}
```

24.16 Modify the program you created in Exercise 24.15 to display the shapes in a `JOptionPane.PLAIN_MESSAGE` dialog.
ANS:

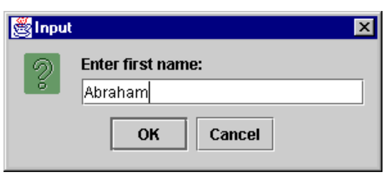
```
// Exercise 24.16 Solution
// Shapes.java
// Program draws four shapes in a Plain Message Dialog
import javax.swing.JOptionPane;

public class Shapes {
   public static void main( String args[] )
   {
       String shapeString;
       shapeString =
             "*********     ***       *        *    " + "\n"
          +  "*       *   *     *    ***      * *   " + "\n"
          +  "*       *  *       *  *****    *   *  " + "\n"
          +  "*       *  *       *    *     *     * " + "\n"
          +  "*       *  *       *    *    *       *" + "\n"
          +  "*       *  *       *    *     *     * " + "\n"
          +  "*       *  *       *    *      *   *  " + "\n"
          +  "*       *   *     *     *       * *   " + "\n"
          +  "*********     ***       *        *    " + "\n" ;

      JOptionPane.showMessageDialog(
        null, shapeString, "Shapes",
        JOptionPane.PLAIN_MESSAGE );

      System.exit( 0 );
   }
}
```

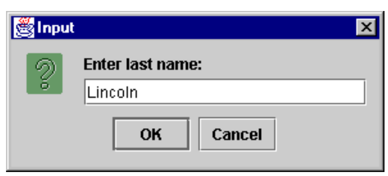


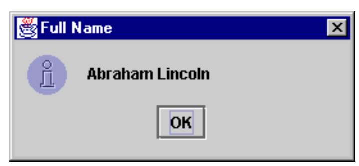
24.17 Write a program that reads a first name and a last name from the user as two separate inputs and concatenates the first name and last name separated by a space. Display in a message dialog the concatenated name.

ANS:

```java
// Exercise 24.17 Solution
// Name.java
// Program puts a first name and a last name together
// as input by the user
import javax.swing.JOptionPane;

public class Name {
   public static void main( String args[] )
   {
      String firstName,    // first string entered by user
             lastName;     // last string entered by user

       // read first name from user
       firstName = JOptionPane.showInputDialog( "Enter first name: " );

       // read last name from user
       lastName = JOptionPane.showInputDialog( "Enter last name: " );

       // Display results
       JOptionPane.showMessageDialog(
          null, firstName + " " + lastName, "Full Name",
          JOptionPane.INFORMATION_MESSAGE );

       System.exit( 0 );
   }
}
```







25

Beyond C & C++: Operators, Methods & Arrays in Java: Solutions

SOLUTIONS

25.5 Answer each of the following questions:

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

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

b) Why is the `Math.random` method useful for simulating games of chance?

ANS: Because it produces a series of random numbers.

c) Why is it often necessary to scale and/or shift the values produced by `Math.random`?

ANS: To produce random numbers in a specific range.

d) 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 toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at a specified time.

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

a) $1 \le n \le 2$

ANS: `n = ( int ) ( 1 + Math.random() * 2 );`

b) $1 \le n \le 100$

ANS: `n = ( int ) ( 1 + Math.random() * 100);`

c) $0 \le n \le 9$

ANS: `n = ( int ) ( Math.random() * 10 );`

d) $1000 \le n \le 1112$

ANS: `n = ( int ) ( 1000 + Math.random() * 113 );`

e) $-1 \le n \le 1$

ANS: `n = ( int ) ( -1 + Math.random() * 3 );`

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

ANS: `n = ( int ) ( -3 + Math.random() * 15 );`

25.7 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: `System.out.print( ( int ) ( ( 1 + Math.random() * 5 ) * 2 ) );`

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

ANS: `System.out.print( ( int ) ( ( 1 + Math.random() * 5 ) * 2 + 1 ) );`

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

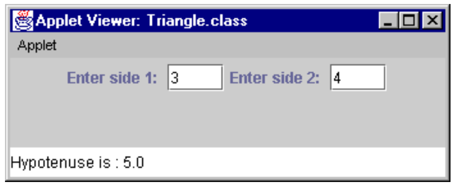
ANS: `System.out.print( ( int ) ( ( Math.random() * 5 ) * 4 + 6 ) );`

25.8 Define a method `hypotenuse` that calculates the length of the hypotenuse of a right triangle when the other two sides are given. The method should take two arguments of type `double` and return the hypotenuse as a `double`. Incorporate this method into an applet that reads integer values for `side1` and `side2` from `JTextField`s and performs the calculation with the `hypotenuse` method. Determine the length of the hypotenuse for each of the following triangles. [*Note*: Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextField`s and pressing *Enter* in the second `JTextField`.]

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

ANS:

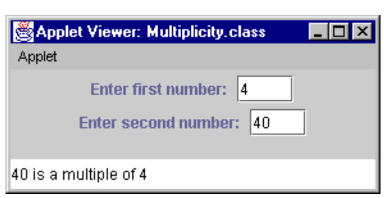
```java
// Exercise 25.8 Solution
// Triangle.java
// Program calculates the hypotenuse of
// a right triangle.
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Triangle extends JApplet
   implements ActionListener {
   JTextField sideInput, side2Input;
   JLabel sidePrompt, sidePrompt2;

   public void init()
   {
      sideInput = new JTextField( 4 );
      side2Input = new JTextField( 4 );
      side2Input.addActionListener( this );
      sidePrompt = new JLabel( "Enter side 1: " );
      sidePrompt2 = new JLabel( "Enter side 2: " );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( sidePrompt );
      c.add( sideInput );
      c.add( sidePrompt2 );
      c.add( side2Input );
   }

   public void actionPerformed( ActionEvent e )
   {
      double side1, side2;

      side1 = Double.parseDouble( side2Input.getText() );
      side2 = Double.parseDouble( sideInput.getText() );

      double h = hypotenuse( side1, side2 );
      showStatus( "Hypotenuse is : " + h );
   }

   public double hypotenuse( double s1, double s2 )
   {
      double hypotSquared = Math.pow( s1, 2 ) + Math.pow( s2, 2 );

      return Math.sqrt( hypotSquared );
   }
}
```


Applet Viewer: Triangle.class
Applet
Enter side 1:
3
Enter side 2:
4
Hypotenuse is : 5.0

25.9 Write a method `multiple` that determines for a pair of integers whether the second integer is a multiple of the first. The method should take two integer arguments and return `true` if the second is a multiple of the first and `false` otherwise. Incorporate this method into an applet that inputs a series of pairs of integers (one pair at a time using `JTextField`s). [*Note*: Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextField`s and pressing *Enter* in the second `JTextField`.]

ANS:

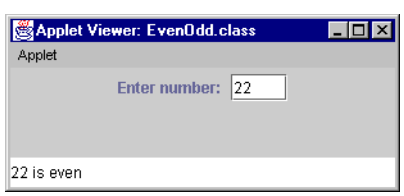
```java
// Exercise 25.9 Solution
// Multiplicity.java
// Determines if the second number entered
// is a multiple of the first.
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class Multiplicity extends JApplet implements ActionListener {
   JTextField input, input2;
   JLabel prompt, prompt2;

   public void init()
   {
      input = new JTextField( 4 );
      input2 = new JTextField( 4 );
      input2.addActionListener( this );
      prompt = new JLabel( "Enter first number: " );
      prompt2 = new JLabel( "Enter second number: " );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( prompt );
      c.add( input );
      c.add( prompt2 );
      c.add( input2 );
   }

   public void actionPerformed( ActionEvent e )
   {
      int first, second;

      first = Integer.parseInt( input.getText() );
      second = Integer.parseInt( input2.getText() );

      if ( multiple( first, second ) == true )
         showStatus( second + " is a multiple of " +
                     first );
      else
         showStatus( second + " is not a multiple of " +
                     first );
   }

   public boolean multiple( int one, int two )
   {
      if ( ( two % one == 0 ) && two != 0 )
         return true;

      return false;
   }
}
```



25.10 Write an applet that inputs integers (one at a time) and passes them one at a time to method `isEven`, which uses the modulus operator to determine if an integer is even. The method should take an integer argument and return `true` if the integer is even and `false` otherwise. Use an input dialog to obtain the data from the user.

ANS:

```
// Exercise 25.10 Solution
// EvenOdd.java
// Determines if a number is odd or even
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class EvenOdd extends JApplet implements ActionListener {
   JTextField input;
   JLabel prompt;

   public void init()
   {
      input = new JTextField( 4 );
      input.addActionListener( this );
      prompt = new JLabel( "Enter number: " );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( prompt );
      c.add( input );
   }

   public void actionPerformed( ActionEvent e )
   {
      int number = Integer.parseInt( input.getText() );
      String result = "";

      if ( isEven( number ) == true )
         result = number + " is even";
      else
         result = number + " is odd ";

      showStatus( result );
   }

   public boolean isEven( int num )
   {
      if ( num % 2 == 0 )
         return true;

      return false;
   }
}
```



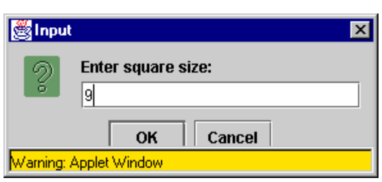
25.11 Write a method `squareOfAsterisks` that displays a solid square of asterisks whose side is specified in integer parameter `side`. For example, if `side` is 4, the method displays

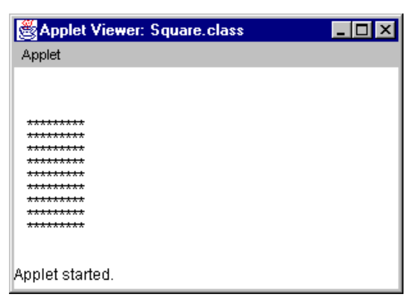
```
****
****
****
****
```

Incorporate this method into an applet that reads an integer value for `side` from the user at the keyboard and performs the drawing with the `squareOfAsterisks` method. Note that this method should be called from the applet's `paint` method and should be passed the `Graphics` object from `paint`.

ANS:

```
// Exercise 25.11 Solution
// Square.java
// Program draws a square of asterisks
import javax.swing.*;
import java.awt.*;

public class Square extends JApplet {
   int size;

   public void init()
   {
      String input = JOptionPane.showInputDialog(
         "Enter square size:" );

      size = Integer.parseInt( input );
   }

   public void squareOfAsterisks( Graphics g )
   {
      int y = 50, x = 5;

      for ( int a = 1; a <= size * size; a++ ) {
         g.drawString( "*", x += 5, y );

         if ( a % size == 0 ) {
            y += 10;
            x = 5;
         }
      }
   }

   public void paint( Graphics g )
   {
      squareOfAsterisks( g );
   }
}
```





25.12 Implement the following integer methods:

a) Method `celsius` returns the Celsius equivalent of a Fahrenheit temperature using the calculation

```
C = 5.0 / 9.0 * ( F - 32 );
```

b) Method `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature.

```
F = 9.0 / 5.0 * C + 32;
```

c) Use these methods to write an applet that enables the user to enter either a Fahrenheit temperature and display the Celsius equivalent or enter a Celsius temperature and display the Fahrenheit equivalent.

[*Note:* This applet will require that two `JTextField` objects that have registered action events. When `actionPerformed` is invoked, the `ActionEvent` parameter has method `getSource()` to determine the GUI component with which the user interacted. Your `actionPerformed` method should contain an `if/else` structure of the following form:

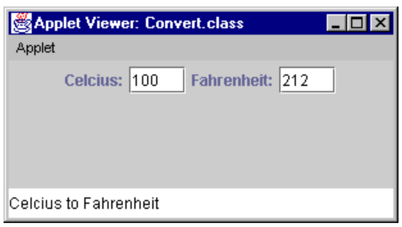
```
if ( e.getSource() == input1 ) {
   // process input1 interaction here
}
else {  // e.getSource() == input2
   // process input2 interaction here
}
```

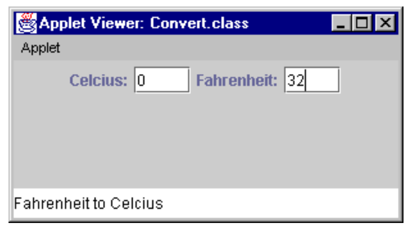
where `input1` and `input2` are `JTextField` references.]

ANS:

```
// Exercise 25.12 Solution
// Convert.java
// Program converts Fahrenheit to Celcius
// and vice versa.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Convert extends JApplet implements ActionListener {
   JTextField cInput, fInput;
   JLabel cLabel, fLabel;

   public void init()
   {
      cInput = new JTextField( 4 );
      fInput = new JTextField( 4 );
      cInput.addActionListener( this );
      fInput.addActionListener( this );
      cLabel = new JLabel( "Celcius:" );
      fLabel = new JLabel( "Fahrenheit:" );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( cLabel );
      c.add( cInput );
      c.add( fLabel );
      c.add( fInput );
   }

   public void actionPerformed( ActionEvent e )
   {
      if ( e.getSource() == cInput ) {
         int c = Integer.parseInt( cInput.getText() );

         fInput.setText( String.valueOf( celcius( c ) ) );
         showStatus( "Celcius to Fahrenheit" );
      }
      else if ( e.getSource() == fInput ) {
         int f = Integer.parseInt( fInput.getText() );

         cInput.setText( String.valueOf( fahrenheit( f ) ) );
```

```
            showStatus( "Fahrenheit to Celcius" );
         }
      }

      public int celcius( int cTemp )
      {
         return ( ( int ) ( 9.0 / 5.0 * cTemp + 32 ) );
      }

      public int fahrenheit( int fTemp )
      {
         return ( ( int ) ( 5.0 / 9.0 * ( fTemp - 32 ) ) );
      }
   }
```





25.13 An integer is said to be *prime* if it is divisible only by 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

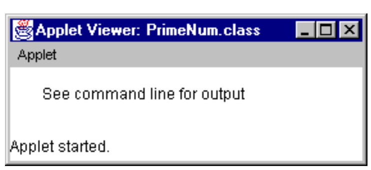
a) Write a method that determines if a number is prime.

b) Use this method in an applet 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? Display the results in a `JTextArea` that has scrolling functionality.

c) Initially you might think that *n*/2 is the upper limit for which you must test to see if 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.

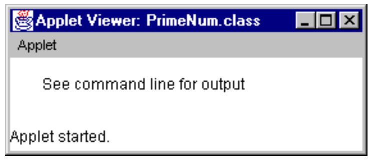
ANS:

```java
// Exercise 25.13 n/2 limit
// PrimeNum.java
// Program calculates prime numbers
import javax.swing.*;
import java.awt.*;

public class PrimeNum extends JApplet {

   public void start()
   {
      int number, count = 0;

      System.out.println( "Prime numbers between 1 and 1000 are: " );

      for ( int m = 1; m <= 10000; m++ )
         if ( prime( m ) == true ) {
            ++count;
            System.out.println( m );
         }
   }

   public void paint( Graphics g )
   {
      g.drawString( "See command line for output", 25, 25 );
   }

   public boolean prime( int n )
   {
      for ( int v = 2; v <= n / 2; v++ )
         if ( n % v == 0 )
            return false;

      return true;
   }
}
```



```
Prime numbers between 1 and 1000 are:
1
2
3
5
7
11
...
9931
9941
9949
9967
9973
```

```java
// Exercise 25.13 sqrt(n) limit
// PrimeNum2.java
// Program calculates prime numbers
import javax.swing.*;
import java.awt.*;

public class PrimeNum2 extends JApplet {

   public void start()
   {
      int number, count = 0;

      System.out.println( "Prime numbers between 1 and 1000 are: " );

      for ( int m = 1; m <= 10000; m++ )
         if ( prime( m ) == true ) {
            ++count;
            System.out.println( m );
         }
   }

   public void paint( Graphics g )
   {
      g.drawString( "See command line for output", 25, 25 );
   }

   public boolean prime( int n )
   {
      for ( int v = 2; v <= ( int ) Math.sqrt( n ); v++ )
         if ( n % v == 0 )
            return false;

      return true;
   }
}
```



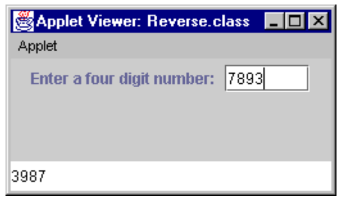

```
Prime numbers between 1 and 1000 are:
1
2
3
5
7
11
...
9931
9941
9949
9967
9973
```

25.14 Write a method that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the method should return 1367. Incorporate the method into an applet that reads a value from the user. Display the result of the method in the status bar.

ANS:

```
// Exercise 25.14 Solution
// Reverse.java
// Program takes a four digit number
// and prints out its digits reversed
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Reverse extends JApplet implements ActionListener {
   JTextField input;
   JLabel prompt;
   int number;

   public void init()
   {
      input = new JTextField( 6 );
      input.addActionListener( this );
      prompt = new JLabel( "Enter a four digit number: " );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( prompt );
      c.add( input );
   }

   public void actionPerformed( ActionEvent e )
   {
      number = Integer.parseInt( input.getText() );
      reverseDigits();
   }

   public void reverseDigits()
   {
      int digit1 = 0, digit2 = 0, digit3 = 0,
          digit4 = 0, factor = 1000, value = 0;

      while ( factor >= 1 ) {
         int temp = number / factor;

         switch ( factor ) {
            case 1000:
               digit4 = temp;
               break;
            case 100:
               digit3 = temp * 10;
               break;
            case 10:
               digit2 = temp * 100;
               break;
            case 1:
               digit1 = temp * 1000;
               break;
         }

         number %= factor;
         factor /= 10;
      }

      if ( digit1 == 0 )  // special case when last digit initially is 0
         showStatus( String.valueOf( 0 ) + String.valueOf( digit2 / 100 )
            + String.valueOf( digit3 / 10 ) +
            String.valueOf( digit4 ) );
```

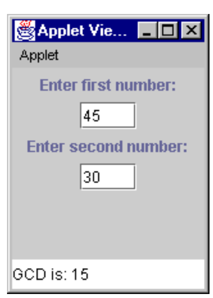
```
      else
         showStatus( String.valueOf(digit1 + digit2 + digit3 + digit4) );
   }
}
```



25.15 The *greatest common divisor (GCD)* of two integers is the largest integer that evenly divides each of the two numbers. Write a method `gcd` that returns the greatest common divisor of two integers. Incorporate the method into an applet that reads two values from the user. Display the result of the method in the status bar.

ANS:

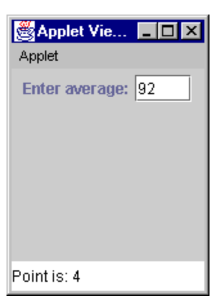
```java
// Exercise 25.15 Solution
// Divisor.java
// Program finds the greatest
// common divisor of two numbers.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Divisor extends JApplet implements ActionListener {
   JTextField input1, input2;
   JLabel label1, label2;

   public void init()
   {
      input1 = new JTextField( 4 );
      input2 = new JTextField( 4 );
      input2.addActionListener( this );
      label1 = new JLabel( "Enter first number:" );
      label2 = new JLabel( "Enter second number:" );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( label1 );
      c.add( input1 );
      c.add( label2 );
      c.add( input2 );
   }

   public void actionPerformed( ActionEvent e )
   {
      int num1, num2;

      num1 = Integer.parseInt( input1.getText() );
      num2 = Integer.parseInt( input2.getText() );

      showStatus( "GCD is: " + gcd( num1, num2 ) );
   }

   public int gcd( int x, int y )
   {
      int greatest = 1;

      for ( int z = 2; z <= ( ( x < y ) ? x : y ); z++ )
         if ( ( x % z == 0 ) && ( y % z == 0 ) )
            greatest = z;

      return greatest;
   }
}
```



25.16 Write a method `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. Incorporate the method into an applet that reads a value from the user. Display the result of the method in the status bar.

ANS:

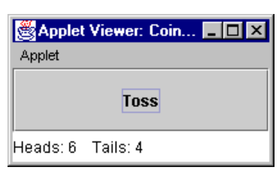
```java
// Exercise 25.16 Solution
// Average.java
// Program displays a number
// representing the student's average
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Average extends JApplet
   implements ActionListener {
   JTextField input;
   JLabel prompt;

   public void init()
   {
      input = new JTextField( 4 );
      input.addActionListener( this );
      prompt = new JLabel( "Enter average:" );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( prompt );
      c.add( input );
   }

   public void actionPerformed( ActionEvent e )
   {
      int number = Integer.parseInt( input.getText() );

      if ( number >= 0 && number <= 100 )
         showStatus( "Point is: " + qualityPoints( number ) );
      else
         showStatus( "Invalid input." );
   }

   public int qualityPoints( int grade )
   {
      if ( grade >= 90 )
         return 4;
      else if ( grade >= 80 )
         return 3;
      else if ( grade >= 70 )
         return 2;
      else if ( grade >= 60 )
         return 1;
      else
         return 0;
   }
}
```



25.17 Write an applet that simulates coin tossing. Let the program toss the coin each time the user presses the "Toss" button. Count the number of times each side of the coin appears. Display the results. The program should call a separate method `flip` that takes no arguments and returns `false` for tails and `true` for heads. [*Note:* If the program realistically simulates the coin tossing, each side of the coin should appear approximately half the time.]

ANS:

```java
// Exercise 25.17 Solution
// Coin.java
// Program simulates tossing a coin.
import javax.swing.*;
import java.awt.event.*;

public class Coin extends JApplet
   implements ActionListener {
   int heads, tails;
   JButton b;

   public void init()
   {
      b = new JButton( "Toss" );
      b.addActionListener( this );
      getContentPane().add( b );
   }

   public void actionPerformed( ActionEvent e )
   {
      if ( flip() == true )
         ++heads;
      else
         ++tails;

      showStatus( "Heads: " + heads + "    Tails: " + tails );
   }

   public boolean flip()
   {
      if ( ( int ) ( Math.random() * 2 ) == 1 )
         return true;
      else
         return false;
   }
}
```



25.18 Computers are playing an increasing role in education. Write a program that will help an elementary school student learn multiplication. Use `Math.random` to produce two positive one-digit integers. It should then display a question in the status bar such as

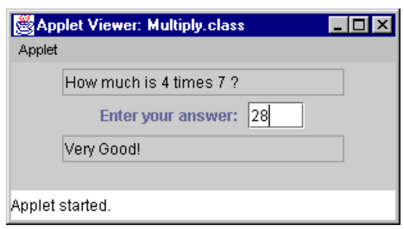
```
How much is 6 times 7?
```

The student then types the answer into a `JTextField`. Your program checks the student's answer. If it is correct, draw the string `"Very good!"` on the applet, then ask another multiplication question. If the answer is wrong, draw the string `"No. Please try again."` on the applet, then let the student try the same question again repeatedly until the student finally gets it right. A separate method should be used to generate each new question. This method should be called once when the applet begins execution and each time the user answers the question correctly. All drawing on the applet should be performed by the `paint` method.

ANS:

```java
// Exercise 25.18 Solution
// Multiply.java
// Program generates single digit multiplication
// problems.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

// Note: Applets have a problem rendering Paint graphics when
// Swing components are attached to the UI.  Consequently, this
// solution does not use paint

public class Multiply extends JApplet implements ActionListener {
   JTextField question;
   JTextField input;
   JTextField response;
   JLabel prompt;
   int answer;
   String questionString;

   public void init()
   {
      input = new JTextField( 4 );
      input.addActionListener( this );
      prompt = new JLabel( "Enter your answer: " );

      response = new JTextField( 20 );
      response.setEditable( false );

      question = new JTextField( 20 );
      question.setEditable( false );

      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( question );
      c.add( prompt );
      c.add( input );
      c.add( response );

      createQuestion();
   }

   public void start()
   {
      question.setText( questionString );
   }

   public void actionPerformed( ActionEvent e )
   {
      int guess = Integer.parseInt( input.getText() );

      input.setText( "" );

      if ( guess != answer )
         response.setText( "No. Please try again." );
```

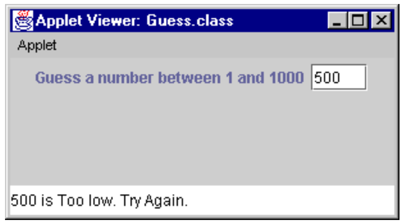
```
      else {
         response.setText( "Very Good!" );
         createQuestion();
      }

      question.setText(questionString);
   }

   // Create a new question, and
   // the corresponding answer
   public void createQuestion()
   {
      int digit1 = getNumber();
      int digit2 = getNumber();

      answer = digit1 * digit2;
      questionString = "How much is " + digit1 + " times " + digit2 + " ?";
   }

   public int getNumber()
   {
      return ( ( int ) ( Math.random() * 10 ) );
   }
}
```



25.19 Write an applet that plays the "guess the number" game as follows: Your program chooses the number to be guessed by selecting a random integer in the range 1 to 1000. The applet displays the prompt `Guess a number between 1 and 1000` next to a `JTextField`. The player types a first guess into the `JTextField` and presses the *Enter* key. If the player's guess is incorrect, your program should display `Too high. Try again.` or `Too low. Try again.` in the status bar to help the player "zero in" on the correct answer and should clear the `JTextField` so the user can enter the next guess. When the user enters the correct answer, display `Congratulations. You guessed the number!` in the status bar and clear the `JTextField` so the user can play again. [Note: The guessing technique employed in this problem is similar to a *binary search*.]

ANS:

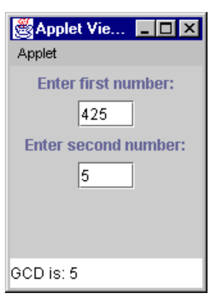
```java
// Exercise 25.19 Solution
// Guess.java
// Program plays guess the number.
// problems.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Guess extends JApplet implements ActionListener {
   JTextField input;
   JLabel prompt;
   int answer;

   public void init()
   {
      input = new JTextField( 4 );
      input.addActionListener( this );
      prompt = new JLabel( "Guess a number between 1 and 1000" );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( prompt );
      c.add( input );
      answer = getNumber();
   }

   public void actionPerformed( ActionEvent e )
   {
      int userGuess = Integer.parseInt( input.getText() );

      checkUserGuess( userGuess );
      input.setText( "" );
   }

   public int getNumber()
   {
      return ( ( int ) ( 1 + Math.random() * 1000 ) );
   }

   public void checkUserGuess( int userGuess )
   {
      if ( userGuess < answer )
         showStatus( userGuess + " is Too low. Try Again." );
      else if ( userGuess > answer )
         showStatus( userGuess + " is Too High. Try Again." );
      else {
         showStatus( "Congratulations. You guessed the number!" );
         input.setText( "" );

         // new search
         answer = getNumber();
      }
   }
}
```


Applet Viewer: Guess.class
Applet
Guess a number between 1 and 1000
500
500 is Too low. Try Again.

25.20 The greatest common divisor of integers `x` and `y` is the largest integer that evenly divides both `x` and `y`. Write a recursive method `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. Use this method to replace the one you wrote in the applet of Exercise 25.15.

ANS:

```java
// Exercise 25.20 Solution
// Divisor.java
// Program recursively finds the greatest
// common divisor of two numbers.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Divisor extends JApplet implements ActionListener {
   JTextField input1, input2;
   JLabel label1, label2;

   public void init()
   {
      input1 = new JTextField( 4 );
      input2 = new JTextField( 4 );
      input2.addActionListener( this );
      label1 = new JLabel( "Enter first number:" );
      label2 = new JLabel( "Enter second number:" );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( label1 );
      c.add( input1 );
      c.add( label2 );
      c.add( input2 );
   }

   public void actionPerformed( ActionEvent e )
   {
      int num1, num2;

      num1 = Integer.parseInt( input1.getText() );
      num2 = Integer.parseInt( input2.getText() );
      showStatus( "GCD is: " + gcd( num1, num2 ) );
   }

   public int gcd( int x, int y )
   {
      if ( y == 0 )
         return x;
      else
         return gcd( y, x % y );
   }
}
```



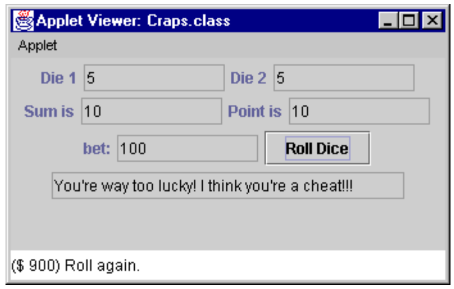
25.21 Modify the craps program of Fig. 25.13 to allow wagering. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a `wager`. Check that `wager` is less than or equal to `bankBalance`, and if not, have the user reenter `wager` until a valid `wager` is entered. After a correct `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 if `bankBalance` 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 c'mon, take a chance!"` or `"You're up big. Now's the time to cash in your chips!"`. Implement the "chatter" as a separate method that randomly chooses the string to display.

ANS:

```java
// Exercise 25.21 Solution
// Craps.java
// Program plays Craps
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class Craps extends JApplet implements ActionListener {
   // constant variables for status of game
   final int WON = 0, LOST = 1, CONTINUE = 2;

   // other variables used in program
   boolean firstRoll = true;   // true if first roll
   int dieSum;             // sum of the dice
   int myPoint;   // point if no win/loss on first roll
   int gameStatus = CONTINUE;  // WON, LOST, CONTINUE
   int bankBalance, wager;

   // graphical user interface components
   JLabel die1Label, die2Label, sumLabel, pointLabel, betLabel;
   JTextField firstDie, secondDie, sum, point, better, chatter;
   JButton roll;

   // setup graphical user interface components
   public void init()
   {
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );

      bankBalance = 1000;
      betLabel = new JLabel( "bet:" );
      better = new JTextField( "100", 10 );
      die1Label = new JLabel( "Die 1" );
      firstDie = new JTextField( 10 );
      firstDie.setEditable( false );
      die2Label = new JLabel( "Die 2" );
      secondDie = new JTextField( 10 );
      secondDie.setEditable( false );
      sumLabel = new JLabel( "Sum is" );
      sum = new JTextField( 10 );
      sum.setEditable( false );
      roll = new JButton( "Roll Dice" );
      roll.addActionListener( this );
      pointLabel = new JLabel( "Point is" );
      point = new JTextField( 10 );
      point.setEditable( false );

      chatter = new JTextField( 25 );
      chatter.setEditable( false );
      c.add( die1Label );
      c.add( firstDie );
      c.add( die2Label );
      c.add( secondDie );
      c.add( sumLabel );
      c.add( sum );
      c.add( pointLabel );
```

```
      c.add( point );
      c.add( betLabel );
      c.add( better );
      c.add( roll );
      c.add( chatter );
   }

   // process one roll of the dice
   public void play()
   {
      if ( firstRoll ) {             // first roll of the dice
         dieSum = rollDice();

         switch ( dieSum ) {
            case 7: case 11:         // win on first roll
               gameStatus = WON;
               point.setText( "" );  // clear point text field
               firstRoll = true;     // allow new game to start
               break;
            case 2: case 3: case 12: // lose on first roll
               gameStatus = LOST;
               point.setText( "" );  // clear point text field
               firstRoll = true;     // allow new game to start
               break;
            default:                 // remember point
               gameStatus = CONTINUE;
               myPoint = dieSum;
               point.setText( Integer.toString( myPoint ) );
               firstRoll = false;
               break;
         }
      }
      else {
         dieSum = rollDice();

         if ( dieSum == myPoint )    // win by making point
            gameStatus = WON;
         else if ( dieSum == 7 )     // lose by rolling 7
            gameStatus = LOST;
      }

      if ( gameStatus == CONTINUE )
         showStatus( "($ " + bankBalance + ") Roll again." );
      else {

         if ( gameStatus == WON ) {
            bankBalance += wager;
            showStatus( "($ " + bankBalance + ") Player wins. " +
                        "Click Roll Dice to play again." );
         }
         else {
            bankBalance -= wager;
            checkBalance();
            showStatus( "($ " + bankBalance + ") Player loses. " +
                        "Click Roll Dice to play again." );
         }

         better.setEditable( true );
         firstRoll = true;
      }
   }

   void checkBalance()
   {
      if ( bankBalance == 0 ) {
         System.out.println( "Sorry. You busted!" );
         System.exit( 0 );
      }
   }
```

```

   // call method play when button is clicked
   public void actionPerformed( ActionEvent e )
   {
      int w = Integer.parseInt( better.getText() );

      if ( w > bankBalance || w < 0 )
         showStatus( "( $" + bankBalance + " ) " +
                     "Enter a valid wager!" );
      else {
         wager = w;
         better.setEditable( false );
         play();
      }

      chatter.setText( chatter() );
   }

   // roll the dice
   int rollDice()
   {
      int die1, die2, workSum;

      die1 = 1 + ( int ) ( Math.random() * 6 );
      die2 = 1 + ( int ) ( Math.random() * 6 );
      workSum = die1 + die2;

      firstDie.setText( Integer.toString( die1 ) );
      secondDie.setText( Integer.toString( die2 ) );
      sum.setText( Integer.toString( workSum ) );

      return workSum;
   }

   public String chatter()
   {
      String s = null;

      switch ( ( int ) ( Math.random() * 5 ) ) {
         case 0:
            s = "Oh, you're going for broke huh?";
            break;
         case 1:
            s = "Aw cmon, take a chance!";
            break;
         case 2:
            s = "You're up big. Now's the " +
                "time to cash in your chips!";
            break;
         case 3:
            s = "You're way too lucky! I think you're " +
                "a cheat!!!";
            break;
         case 4:
            s = "I'm betting all my money on you.";
            break;
      }

      return s;
   }
}
```


Applet Viewer: Craps.class
Applet
Die 1
5
Die 2
5
Sum is
10
Point is
10
bet:
100
Roll Dice
You're way too lucky! I think you're a cheat!!!
($ 900) Roll again.

25.22 Write a program to simulate the rolling of two dice. The program should use `Math.random` to roll the first die and should use `Math.random` 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, the sum of the values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums. Figure 25.24 shows the 36 possible combinations of the two dice. Your program should roll the 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).

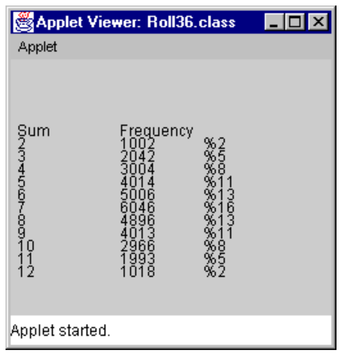
	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 25.24 The 36 possible outcomes of rolling two dice.

ANS:

```java
// Exercise 25.22 Solution
// Roll36.java
// Program simulates rolling two
// six-sided die 36,000 times
// NOTE: this program could take a
// few seconds before displaying the data
import javax.swing.*;
import java.awt.*;

public class Roll36 extends JApplet {
   int total[];

   public void init()
   {
      total = new int[ 13 ];

      for ( int i = 0; i < total.length; i++ )
         total[ i ] = 0;

      roll2Dice();
   }

   public void roll2Dice()
   {
      int face1, face2;

      for ( int x = 1; x <= 36000; x++ ) {
         face1 = ( int ) ( 1 + Math.random() * 6 );
         face2 = ( int ) ( 1 + Math.random() * 6 );

         total[ face1 + face2 ]++;
      }
   }

   public void paint( Graphics g )
   {
      super.paint( g );
      int y = 60;

```

```
      g.drawString( "Sum", 5, 60 );
      g.drawString( "Frequency", 85, 60 );

      // ignore subscripts 0 and 1
      for ( int k = 2; k < total.length; k++ ) {
         g.drawString( String.valueOf( k ), 5, y += 10 );
         g.drawString( String.valueOf( total[ k ] ), 85, y );

         double percent = ( double ) total[ k ] / 360.0;
         g.drawString( "%" + ( int ) percent, 150, y );
      }
   }
}
```



26

Java Object-Based Programming: Solutions

SOLUTIONS

26.2 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` instance variables of the class—the `numerator` and the `denominator`. Provide a constructor method that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form (i.e., the fraction

```
2/4
```

would be stored in the object as 1 in the `numerator` and 2 in the `denominator`). Provide a no-argument constructor that sets default values in case no initializers are provided. Provide `public` methods for each of the following:

a) Addition of two `Rational` numbers. The result of the addition should be stored in reduced form.
b) Subtraction of two `Rational` numbers. The result of the subtraction should be stored in reduced form.
c) Multiplication of two `Rational` numbers. The result of the multiplication should be stored in reduced form.
d) Division of two `Rational` numbers. The result of the division 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 floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

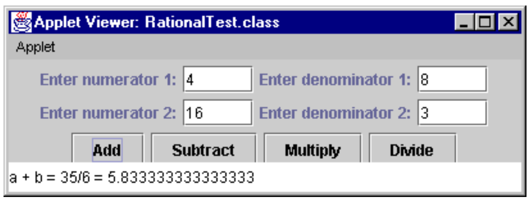
ANS:

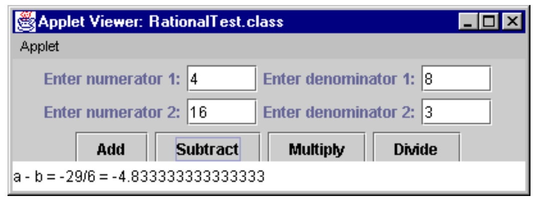
```java
// Exercise 26.2 Solution
// Rational.java
// Definition of class Rational

public class Rational {
   private int numerator;
   private int denominator;

   // Initialize numerator to 0 and denominator to 1
   public Rational() { this( 0, 1 ); }

   // Initialize numerator part to n and denominator part to 1
   public Rational( int n ) { this( n, 1 ); }

   // Initialize numerator part to n and denominator part to d
   public Rational( int n, int d )
   {
```

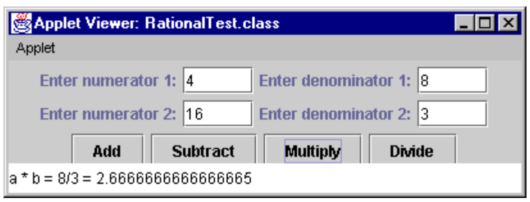
```
      numerator = n;
      denominator = d;
      reduce();
   }

   // Add two Rational numbers
   public Rational sum( Rational right )
   {
      int cd = denominator * right.denominator;
      int numer = numerator * right.denominator +
                  right.numerator * denominator;

      return new Rational( numer, cd );
   }

   // Subtract two Rational numbers
   public Rational subtract( Rational right )
   {
      int cd = denominator * right.denominator;
      int numer = numerator * right.denominator -
                  right.numerator * denominator;

      return new Rational( numer, cd );
   }

   // Multiply two Rational numbers
   public Rational multiply( Rational right )
   {
      return new Rational( numerator * right.numerator,
                           denominator * right.denominator );
   }

   // Divide two Rational numbers
   public Rational divide( Rational right )
   {
      return new Rational( numerator * right.denominator,
                           denominator * right.numerator );
   }

   // Reduce the fraction
   private void reduce()
   {
      int gcd = 0;
      int smaller = Math.min( numerator, denominator );

      for ( int x = 2; x <= smaller; x++ )
         if ( numerator % x == 0 && denominator % x == 0 )
            gcd = x;

      if ( gcd != 0 ) {
         numerator /= gcd;
         denominator /= gcd;
      }
   }

   // Return String representation of a Rational number
   public String toString()
      { return numerator + "/" + denominator; }

   // Return floating-point String representation of
   // a Rational number
   public String toFloatString()
   {
      return Double.toString(
               ( double ) numerator / denominator );
   }
}
```

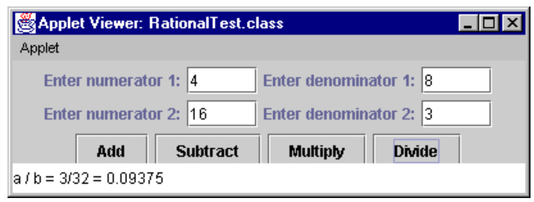
```
// Exercise 26.2: RationalTest.java
// Test the Rational number class
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class RationalTest extends JApplet implements ActionListener {
   private Rational a, b;
   private JLabel nlabel1, nlabel2, dlabel1, dlabel2;
   private JTextField numer1, numer2, denom1, denom2;
   private JButton addit, subtract, multiply, divide;

   public void init()
   {
      nlabel1 = new JLabel( "Enter numerator 1:" );
      nlabel2 = new JLabel( "Enter numerator 2:" );
      dlabel1 = new JLabel( "Enter denominator 1:" );
      dlabel2 = new JLabel( "Enter denominator 2:" );

      numer1 = new JTextField( 5 );
      numer2 = new JTextField( 5 );
      denom1 = new JTextField( 5 );
      denom2 = new JTextField( 5 );

      addit    = new JButton( "Add" );
      subtract = new JButton( "Subtract" );
      multiply = new JButton( "Multiply" );
      divide   = new JButton( "Divide" );

      addit.addActionListener( this );
      subtract.addActionListener( this );
      multiply.addActionListener( this );
      divide.addActionListener( this );

      Container c = getContentPane();
      c.setLayout(new FlowLayout());

      c.add( nlabel1 );
      c.add( numer1 );
      c.add( dlabel1 );
      c.add( denom1 );
      c.add( nlabel2 );
      c.add( numer2 );
      c.add( dlabel2 );
      c.add( denom2 );
      c.add( addit );
      c.add( subtract );
      c.add( multiply );
      c.add( divide );
   }

   public void actionPerformed( ActionEvent e )
   {
      Rational r;
      a = new Rational( Integer.parseInt( numer1.getText() ),
                        Integer.parseInt( denom1.getText() ) );
      b = new Rational( Integer.parseInt( numer2.getText() ),
                        Integer.parseInt( denom2.getText() ) );

      if ( e.getSource() == addit ) {
         r = a.sum( b );
         showStatus( "a + b = " + r + " = " + r.toFloatString() );
      }
      else if ( e.getSource() == subtract ) {
         r = a.subtract( b );
         showStatus( "a - b = " + r + " = " + r.toFloatString() );
      }
      else if ( e.getSource() == multiply ) {
```

```
            r = a.multiply( b );
            showStatus( "a * b = " + r + " = " + r.toFloatString() );
         }
         else if ( e.getSource() == divide ) {
            r = a.divide( b );
            showStatus( "a / b = " + r + " = " + r.toFloatString() );
         }
      }
   }
```









26.3 Modify the `Time2` class of Fig. 26.3 to include the `tick` method that increments the time stored in a `Time2` object by one second. Also provide method `incrementMinute` to increment the minute and method `incrementHour` to increment the hour. The `Time2` object should always remain in a consistent state. Write a driver program that tests the `tick` method, the `incrementMinute` method and the `incrementHour` method to ensure that they work 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).

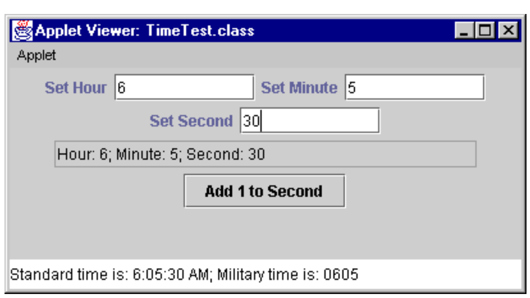
ANS:

```
// Exercise 26.3 Solution
// Time3.java
// Time3 class definition
public class Time3 {
   private int hour;     // 0 - 23
   private int minute;   // 0 - 59
   private int second;   // 0 - 59

   // Time constructor initializes each instance variable
   // to zero. Ensures that Time3 object starts in a
   // consistent state.
   public Time3() { setTime( 0, 0, 0 ); }

   // Time3 constructor: hour supplied, minute and second
   // defaulted to 0.
   public Time3( int h ) { setTime( h, 0, 0 ); }

   // Time3 constructor: hour and minute supplied, second
   // defaulted to 0.
   public Time3( int h, int m ) { setTime( h, m, 0 ); }

   // Time3 constructor: hour, minute and second supplied.
   public Time3( int h, int m, int s ) { setTime( h, m, s ); }

   // Set Methods
   // Set a new Time3 value using military time. Perform
   // validity checks on the data. Set invalid values
   // to zero.
   public void setTime( int h, int m, int s )
   {
      setHour( h );    // set the hour
      setMinute( m );  // set the minute
      setSecond( s );  // set the second
   }

   // set the hour
   public void setHour( int h )
      { hour = ( ( h >= 0 && h < 24 ) ? h : 0 ); }

   // set the minute
   public void setMinute( int m )
      { minute = ( ( m >= 0 && m < 60 ) ? m : 0 ); }

   // set the second
   public void setSecond( int s )
      { second = ( ( s >= 0 && s < 60 ) ? s : 0 ); }

   // Get Methods
   // get the hour
   public int getHour() { return hour; }

   // get the minute
   public int getMinute() { return minute; }

   // get the second
   public int getSecond() { return second; }

```

```
   // Convert to String in military-time format
   public String toMilitaryString()
   {
      return ( hour < 10 ? "0" : "" ) + hour +
             ( minute < 10 ? "0" : "" ) + minute;
   }

   // Convert to String in standard-time format
   public String toString()
   {
      return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
             ":" + ( minute < 10 ? "0" : "" ) + minute +
             ":" + ( second < 10 ? "0" : "" ) + second +
             ( hour < 12 ? " AM" : " PM" );
   }

   // Tick the time by one second
   public void tick()
   {
      setSecond( second + 1 );

      if ( second == 0 )
         incrementMinute();
   }

   // Increment the minute
   public void incrementMinute()
   {
      setMinute( minute + 1 );

      if ( minute == 0 )
         incrementHour();
   }

   // Increment the hour
   public void incrementHour()
   {
      setHour( hour + 1 );
   }
}
```

```
// Exercise 26.3 Solution
// TimeTest.java
// Demonstrating the Time class set and get methods
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class TimeTest extends JApplet implements ActionListener {
   private Time3 t;
   private JLabel hrLabel, minLabel, secLabel;
   private JTextField hrField, minField, secField, display;
   private JButton tickButton;

   public void init()
   {
      t = new Time3();

      hrLabel = new JLabel( "Set Hour" );
      hrField = new JTextField( 10 );
      hrField.addActionListener( this );
      minLabel = new JLabel( "Set Minute" );
      minField = new JTextField( 10 );
      minField.addActionListener( this );
      secLabel = new JLabel( "Set Second" );
      secField = new JTextField( 10 );
```

```java
      secField.addActionListener( this );
      display = new JTextField( 30 );
      display.setEditable( false );
      tickButton = new JButton( "Add 1 to Second" );
      tickButton.addActionListener( this );

      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( hrLabel );
      c.add( hrField );
      c.add( minLabel );
      c.add( minField );
      c.add( secLabel );
      c.add( secField );
      c.add( display );
      c.add( tickButton );
      updateDisplay();
   }

   public void actionPerformed( ActionEvent e )
   {
      if ( e.getSource() == tickButton )
         t.tick();
      else if ( e.getSource() == hrField ) {
         t.setHour( Integer.parseInt( e.getActionCommand().toString() ) );
         hrField.setText( "" );
      }
      else if ( e.getSource() == minField ) {
         t.setMinute( Integer.parseInt( e.getActionCommand().toString() ) );
         minField.setText( "" );
      }
      else if ( e.getSource() == secField ) {
         t.setSecond( Integer.parseInt( e.getActionCommand().toString() ) );
         secField.setText( "" );
      }

      updateDisplay();
   }

   public void updateDisplay()
   {
      display.setText( "Hour: " + t.getHour() +
         "; Minute: " + t.getMinute() +
         "; Second: " + t.getSecond() );
      showStatus( "Standard time is: " + t.toString()+
         "; Military time is: " + t.toMilitaryString() );
   }
}
```



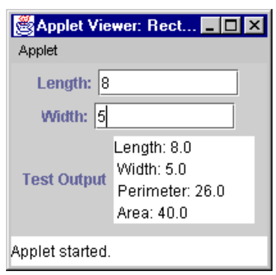
26.4 Create a class `Rectangle`. The class has attributes `length` and `width`, each of which defaults to 1. It has methods that calculate the `perimeter` and the `area` of the rectangle. It has *set* and *get* methods for both `length` and `width`. The *set* methods should verify that `length` and `width` are each floating-point numbers larger than 0.0 and less than 20.0.

ANS:

```java
// Exercise 26.4 Solution
// MyRectangle.java
// Definition of class MyRectangle

public class MyRectangle {
   private double length, width;

   public MyRectangle()  { this( 1.0, 1.0 ); }

   public MyRectangle( double l, double w )
   {
      setLength( l );
      setWidth( w );
   }

   public void setLength( double len )
   { length = ( len >= 0.0 && len <= 20.0 ? len : 1.0 ); }

   public void setWidth( double w )
   { width = ( w >= 0 && w <= 20.0 ? w : 1.0 ); }

   public double getLength() { return length; }

   public double getWidth() { return width; }

   public double perimeter() { return 2 * length + 2 * width; }

   public double area() { return length * width; }

   public String toString (){
       return ("Length: " + length + "\n" +
       " Width: " + width  + "\n" +
       " Perimeter: " + perimeter() + "\n" +
       " Area: " + area() );
   }
}
```

```java
// Exercise 26.4 Solution
// Definition of class RectangleTest
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

public class RectangleTest extends JApplet implements ActionListener {
    private JLabel prompt1, prompt2;
    private JTextField input1, input2;
    private JLabel outputLabel;
    private JTextArea output;
    private MyRectangle r;

    public void init()
    {
       prompt1 = new JLabel( "Length:" );
       prompt2 = new JLabel( "Width:" );
       input1 = new JTextField( 10 );
       input2 = new JTextField( 10 );
       input2.addActionListener( this );

       outputLabel = new JLabel( "Test Output" );
       output = new JTextArea( 4, 10 );
```

```java

      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( prompt1 );
      c.add( input1 );
      c.add( prompt2 );
      c.add( input2 );
      c.add( outputLabel);
      c.add( output);
      r = new MyRectangle();
   }

   public void actionPerformed( ActionEvent e )
   {
      double d1, d2;

      d1 = Double.parseDouble( input1.getText() );
      d2 = Double.parseDouble( input2.getText() );

      r.setLength( d1 );
      r.setWidth( d2 );

      output.setText( r.toString() );
   }
}
```



26.5 Create a more sophisticated `Rectangle` class than the one you created in Exercise 26.4. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* method 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* method also verifies that the supplied coordinates do, in fact, specify a rectangle. Provide methods to calculate the `length`, `width`, `perimeter` and `area`. The length is the larger of the two dimensions. Include a predicate method `isSquare` which determines if the rectangle is a square.

26.6 Modify the `Rectangle` class of Exercise 26.5 to include a `draw` method that displays the rectangle inside a 25-by-25 box enclosing the portion of the first quadrant in which the rectangle resides. Use the methods of the `Graphics` class to help output the `Rectangle`. If you feel ambitious, you might include methods to scale the size of the rectangle, rotate it and move it around within the designated portion of the first quadrant.

26.7 Create a class `HugeInteger` which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods `inputHugeInteger`, `outputHugeInteger`, `add-HugeIntegers` and `subtractHugeIntegers`. For comparing `HugeInteger` objects, provide methods `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `IsGreaterThanOrEqualTo` and `isLessThanOrEqualTo`—each of these is a "predicate" method that simply returns `true` if the relationship holds between the two `HugeIntegers` and returns `false` if the relationship does not hold. Provide a predicate method `isZero`. If you feel ambitious, also provide the method `multiplyHugeIntegers`, the method `divideHugeIntegers` and the method `modulusHugeIntegers`.

26.8 Create class `SavingsAccount`. Use a `static` class variable to store the `annualInterestRate` for all account holders. Each object of the class contains a `private` instance variable `savingsBalance` indicating the amount the saver currently has on deposit. Provide method `calculateMonthlyInterest` to calculate the monthly interest by multiplying the `savingsBalance` by `annualInterestRate` divided by 12; this interest should be added to `savingsBalance`. Provide a `static` method `modifyInterestRate` that sets the `annualInterestRate` to a new value. Write a driver program to test class `SavingsAccount`. Instantiate two `savingsAccount` objects, `saver1` and `saver2`, with balances of $2000.00 and $3000.00, respectively. Set `annualInterestRate` to 4%, then calculate the monthly interest and print the new balances for each of the savers. Then set the `annualInterestRate` to 5% and calculate the next month's interest and print the new balances for each of the savers.

26.9 Create class `IntegerSet`. Each object of the class can hold integers in the range 0 through 100. A set is represented internally as an array of `booleans`. Array element `a[ i ]` is `true` if integer *i* is in the set. Array element `a[ j ]` is `false` if integer *j* is not in the set. The no-argument constructor initializes a set to the so-called "empty set" (i.e., a set whose array representation contains all `false` values).

Provide the following methods: Method `unionOfIntegerSets` 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 `true` if that element is `true` in either or both of the existing sets; otherwise, the element of the third set is set to `false`). Method `intersectionOfIntegerSets` 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 `false` if that element is `false` in either or both of the existing sets; otherwise, the element of the third set is set to `true`). Method `insertElement` inserts a new integer *k* into a set (by setting `a[ k ]` to `true`). Method `deleteElement` deletes integer *m* (by setting `a[m]` to `false`). Method `setPrint` prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set. Print `---` for an empty set. Method `isEqualTo` determines if two sets are equal. Write a program to test your `IntegerSet` class. Instantiate several `IntegerSet` objects. Test that all your methods work properly.

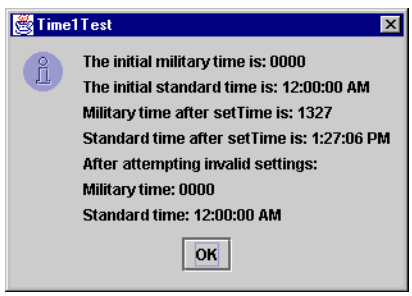
26.10 It would be perfectly reasonable for the `Time1` class of Fig. 26.1 to represent the time internally as the number of seconds since midnight rather than the three integer values `hour`, `minute` and `second`. Clients could use the same `public` methods and get the same results. Modify the `Time1` class of Fig. 26.1 to implement the `Time1` as the number of seconds since midnight and show that there is no visible change to the clients of the class.

ANS:

```java
// Exercise 26.10 Solution
// Time1 class definition

public class Time1 {
   private int totalSeconds;

   public Time1() { setTime( 0, 0, 0 ); }

   public void setTime( int h, int m, int s )
   {
      int hour, minute, second;

      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );

      totalSeconds = hour * 3600 + minute * 60 + second;
   }

   public String toMilitaryString()
   {
      int hour, minute, temp;

      hour = totalSeconds / 3600;
      temp = totalSeconds % 3600;
      minute = temp / 60;

      return ( hour < 10 ? "0" : "" ) + hour +
             ( minute < 10 ? "0" : "" ) + minute;
   }

   public String toString()
   {
      int hour, minute, second, temp;

      hour = totalSeconds / 3600;
      temp = totalSeconds % 3600;
      minute = temp / 60;
      second = temp % 60;

      return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
             ":" + ( minute < 10 ? "0" : "" ) + minute +
             ":" + ( second < 10 ? "0" : "" ) + second +
             ( hour < 12 ? " AM" : " PM" );
   }
}
```

```java
// Exercise 26.10 Solution
// TimeTest.java
// Class TimeTest to exercise class Time
import javax.swing.*;

public class TimeTest {
   public static void main( String args[] )
   {
      Time1 t = new Time1();
      String result = "";

```

```
        result += "The initial military time is: " +
                t.toMilitaryString();
        result += "\nThe initial standard time is: " +
                t.toString();

        t.setTime( 13, 27, 6 );
        result += "\nMilitary time after setTime is: " +
                t.toMilitaryString();
        result += "\nStandard time after setTime is: " +
                t.toString();

        t.setTime( 99, 99, 99 );
        result += "\nAfter attempting invalid settings:";
        result += "\nMilitary time: " + t.toMilitaryString();
        result += "\nStandard time: " + t.toString();

        JOptionPane.showMessageDialog(
              null, result, "Time1Test",
              JOptionPane.INFORMATION_MESSAGE );
        System.exit( 0 );
    }
}
```



26.11 *(Drawing Program)* Create a drawing applet that randomly draws lines, rectangles and ovals. For this purpose, create a set of "smart" shape classes where objects of these classes know how to draw themselves if provided with a `Graphics` object that tells them where to draw (i.e., the applet's `Graphics` object allows a shape to draw on the applet's background). The class names should be `MyLine`, `MyRect` and `MyOval`.

The data for class `MyLine` should include *x1*, *y1*, *x2* and *y2* coordinates. Method `drawLine` method of class `Graphics` will connect the two points supplied with a line. The data for classes `MyRect` and `MyOval` should include an upper-left *x*-coordinate value, an upper-left *y*-coordinate value, a *width* (must be nonnegative) and a *height* (must be nonnegative). All data in each class must be `private`.

In addition to the data, each class should define at least the following `public` methods:

a) A constructor with no arguments that sets the coordinates to 0.
b) A constructor with arguments that sets the coordinates to the supplied values.
c) *Set* methods for each individual piece of data that allow the programmer to independently set any piece of data in a shape (e.g., if you have an instance variable `x1`, you should have a method `setX1`).
d) *Get* methods for each individual piece of data that allow the programmer to independently retrieve any piece of data in a shape (e.g., if you have an instance variable `x1`, you should have a method `getX1`).
e) A `draw` method with the first line

```
public void draw( Graphics g )
```

will be called from the applet's `paint` method to draw a shape onto the screen.

The preceding methods are required. If you would like to provide more methods for flexibility, please do so.

Begin by defining class `MyLine` and an applet to test your classes. The applet should have a `MyLine` instance variable `line` that can refer to one `MyLine` object (created in the applet's `init` method with random coordinates). The applet's `paint` method should draw the shape with a statement like

```
line.draw( g );
```

where `line` is the `MyLine` reference and `g` is the `Graphics` object that the shape will use to draw itself on the applet.

Next, change the single `MyLine` reference into an array of `MyLine` references and hard code several `MyLine` objects into the program for drawing. The applet's `paint` method should walk through the array of `MyLine` objects and draw every one.

After the preceding part is working, you should define the `MyOval` and `MyRect` classes and add objects of these classes into the `MyRect` and `MyOval` arrays. The applet's `paint` method should walk through each array and draw every shape. Create five shapes of each type.

Once the applet is running, select `Reload` from the `appletviewer`'s `Applet` menu to reload the applet. This will cause the applet to choose new random numbers for the shapes and draw the shapes again.

In Chapter 27, we will modify this exercise to take advantage of the similarities between the classes and to avoid reinventing the wheel.

ANS:

```
// Exercise 26.11 Solution
// MyLine.java
// Definition of class MyLine
import java.awt.Graphics;

public class MyLine {
   private int x1, x2;
   private int y1, y2;

   public MyLine()
   {
      x1 = 0;
      y1 = 0;
      x2 = 0;
      y2 = 0;
   }

   public MyLine( int x1, int y1, int x2, int y2 )
   {
      setX1( x1 );
      setX2( x2 );
      setY1( y1 );
```

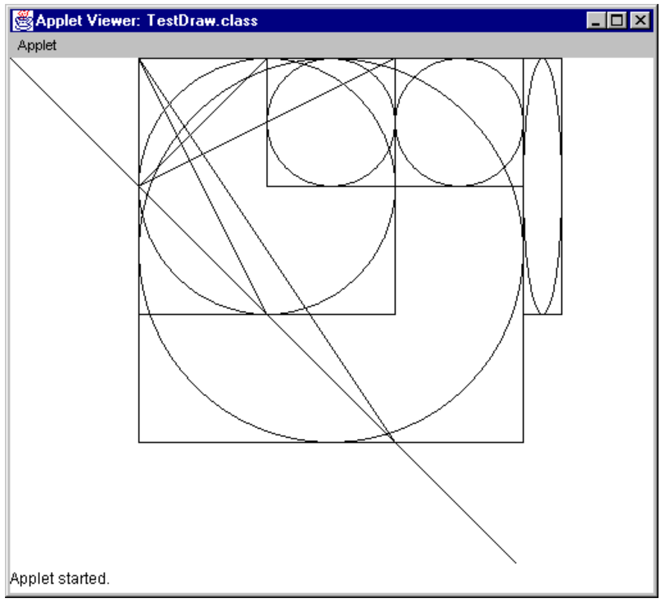
```
      setY2( y2 );
   }

   public void setX1( int x1 )
   { this.x1 = ( x1 >= 0 ? x1 : 0 ); }

   public void setX2( int x2 )
   { this.x2 = ( x2 >= 0 ? x2 : 0 ); }

   public void setY1( int y1 )
   { this.y1 = ( y1 >= 0 ? y1 : 0 ); }

   public void setY2( int x2 )
   { this.y2 = ( y2 >= 0 ? y2 : 0 ); }

   public int getX1() { return x1; }
   public int getX2() { return x2; }
   public int getY1() { return y1; }
   public int getY2() { return y2; }

   public void draw( Graphics g )
   {
      g.drawLine( x1, y1, x2, y2 );
   }
}
```

```
// Exercise 26.11 Solution
// MyOval.java
// Definition of class MyRect
import java.awt.Graphics;

public class MyOval {
   private int length, width;
   private int upperLeftX, upperLeftY;

   public MyOval()
   {
      length =  0;
      width = 0;
      upperLeftX = 0;
      upperLeftY = 0;
   }

   public MyOval( int x, int y, int l, int w )
   {
      setUpperLeftX( x );
      setUpperLeftY( y );
      setLength( l );
      setWidth( w );
   }

   public void setLength( int len )
   { length = ( len >= 0 ? len : 0 ); }

   public void setUpperLeftX( int x )
   { upperLeftX =( x >= 0 ? x : 0 ); }

   public void setUpperLeftY( int y )
   { upperLeftX =( y >= 0 ? y : 0 ); }

   public void setWidth( int w )
   { width = ( w >= 0 ? w : 0 ); }

   public int getLength() { return length; }

   public int getWidth() { return width; }
```

```

   public int getUpperLeftX() { return upperLeftX; }

   public int getUpperLeftY() { return upperLeftY; }

   public void draw( Graphics g )
   {
      g.drawOval( upperLeftX, upperLeftY, length, width );
   }
}
```

```
// Exercise 26.11 Solution
// MyRect.java
// Definition of class MyRect
import java.awt.Graphics;

public class MyRect {
   private int length, width;
   private int upperLeftX, upperLeftY;

   public MyRect()
   {
      length =  0;
      width = 0;
      upperLeftX = 0;
      upperLeftY = 0;
   }

   public MyRect( int x, int y, int l, int w )
   {
      setUpperLeftX( x );
      setUpperLeftY( y );
      setLength( l );
      setWidth( w );
   }

   public void setLength( int len )
   { length = ( len >= 0.0 ? len : 1 ); }

   public void setUpperLeftX( int x )
   { upperLeftX =( x >= 0 ? x : 0 ); }

   public void setUpperLeftY( int y )
   { upperLeftX =( y >= 0  ? y : 0 ); }

   public void setWidth( int w )
   { width = ( w >= 0 ? w : 1 ); }

   public int getLength() { return length; }

   public int getWidth() { return width; }

   public int getUpperLeftX() { return upperLeftX; }

   public int getUpperLeftY() { return upperLeftY; }

   public void draw( Graphics g )
   {
      g.drawRect( upperLeftX, upperLeftY, length, width );
   }
}
```

```
// Exercise 26.11 Solution
// Definition of class RectangleTest
import java.awt.*;
import javax.swing.*;

public class TestDraw extends JApplet {
   private MyLine line[];
   private MyOval oval[];
   private MyRect rect[];

   public void initDraw()
   {
      line = new MyLine[ 5 ];
      line[ 0 ] = new MyLine( 100, 100, 200, 200 );
      line[ 1 ] = new MyLine( 200, 200, 100, 100 );
      line[ 2 ] = new MyLine( 300, 300, 100, 100 );
      line[ 3 ] = new MyLine( 400, 400, 0, 0 );
      line[ 4 ] = new MyLine( 100, 100, 300, 300 );

      oval = new MyOval[ 5 ];
      oval[ 0 ] = new MyOval( 100, 100, 200, 200 );
      oval[ 1 ] = new MyOval( 200, 200, 100, 100 );
      oval[ 2 ] = new MyOval( 300, 300, 100, 100 );
      oval[ 3 ] = new MyOval( 400, 400, 30, 200 );
      oval[ 4 ] = new MyOval( 100, 100, 300, 300 );

      rect = new MyRect[ 5 ];
      rect[ 0 ] = new MyRect( 100, 100, 200, 200 );
      rect[ 1 ] = new MyRect( 200, 200, 100, 100 );
      rect[ 2 ] = new MyRect( 300, 300, 100, 100 );
      rect[ 3 ] = new MyRect( 400, 400, 30, 200 );
      rect[ 4 ] = new MyRect( 100, 100, 300, 300 );
   }

   public void paint( Graphics g )
   {
      initDraw();

      for ( int i = 0; i < line.length; i++ )
 line[ i ].draw( g );

      for ( int i = 0; i < oval.length; i++ )
 oval[ i ].draw( g );

      for ( int i = 0; i < rect.length; i++ )
 rect[ i ].draw( g );
     }
}
```


Applet Viewer: TestDraw.class
Applet
Applet started.

27

Java Object-Oriented Programming: Solutions

SOLUTIONS

27.3 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 subclasses.

ANS: Possible classes are displayed in bold.

`Bicycle` composed of:
- `Handle bars`
- `Seat`
- `Frame`
- `Wheels` composed of:
  - `Tires`
  - `Rims`
  - `Spokes`
- `Pedals`
- `Chain` composed of:
  - `Links`
- Brakes composed of:
  - `Wires`
  - `Brake` Pads
  - `Brake Handles`

27.4 Define each of the following terms: single inheritance, multiple inheritance, interface, superclass and subclass.

ANS:

a) Single inheritance is the process by which a class incorporates the attributes and behaviors of a previously defined class.
b) Multiple inheritance is the process by which a class incorporates the attributes and behaviors of two or more previously defined classes.
c) An interface is a collection of abstract methods that can be implemented to simulate multiple inheritance.
d) A superclass is a class from which other classes inherit attributes and behaviors.
e) A subclass is a class that has inherited attributes and behaviors from a superclass.

27.5 Discuss why casting a superclass reference to a subclass reference is potentially dangerous.

ANS: The reference must refer to an object of the subclass, before being used. When the compiler looks at an object through a subclass reference, it expects to see all the pieces of the subclass. However, if the superclass reference originally referred to a superclass object, the additional pieces added by the subclass do not exist. For this reason, an attempt to cast a subclass reference, that refers to a subclass object, into a superclass reference results in a ClassCastException at execution time.

27.6 Distinguish between single inheritance and multiple inheritance. Why does Java not support multiple inheritance? What feature of Java helps realize the benefits of multiple inheritance?

ANS: Single inheritance inherits from one class only. Multiple inheritance inherits from two or more classes. Java does not support multiple inheritance because of the problems that can be encountered with multiple inheritance. However, Java does support interfaces which provide the benefits of multiple inheritance without the potential problems.

27.7 (*True/False*) A subclass is generally smaller than its superclass.

ANS: False. A subclass is usually larger because it normally adds more data and more functionality.

27.8 (*True/False*) A subclass object is also an object of that subclass's superclass.

ANS: True.

27.9 Rewrite the `Point`, `Circle`, `Cylinder` program of Fig. 27.4 as a `Point`, `Square`, `Cube` program. Do this two ways—once with inheritance and once with composition.

ANS:

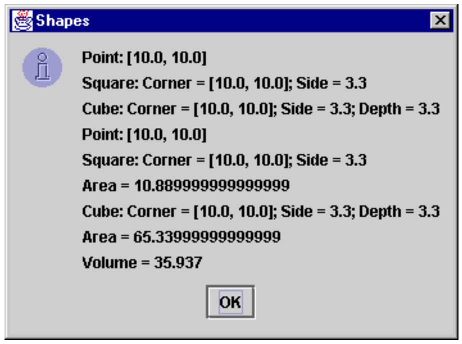
```
// Exercies 27.9 -- Composition
// Point.java
// Definition of class Point

public class Point {
   private double x, y; // coordinates of the Point

   public Point( double a, double b ) { setPoint( a, b ); }

   public void setPoint( double a, double b )
   {
      x = a;
      y = b;
   }

   public double getX() { return x; }

   public double getY() { return y; }

   public String toString()
      { return "[" + x + ", " + y + "]"; }

   public String getName() { return "Point"; }
}
```

```
// Exercies 27.9 -- Composition
// Square.java
// Definition of class Square

public class Square {
   private double side;
   private Point p;       // composition

   public Square()  {  this( 0.0, 0.0, 0.0 );  }

   public Square( double s, double a, double b )
   {
      p = new Point( a, b );   // instantiate point object
      setSide( s );
   }

   public void setSide( double s )
      { side = ( s >= 0 ? s : 0 ); }

   public double getSide() { return side; }

   public double area() { return Math.pow( side, 2 ); }

   public String toString()
      { return "Corner = " + p.toString() + "; Side = " + side; }

   public String getName() { return "Square"; }

   public String getPointName() { return p.getName(); }

   public String getPointString() { return p.toString(); }
}
```

```
// Exercies 27.9 -- Composition
// Cube.java
// Definition of class Cube

public class Cube {
   private double depth;
   private Square s;       // composition

   public Cube( double m, double a, double b )
   {
      s = new Square( m, a, b );
      depth = m;
   }

   public double getDepth() { return depth; }

   public double area() { return s.area() * 6; }

   public double volume() { return s.area() * depth; }

   public String toString()
      { return s.toString() + "; Depth = " + depth; }

   public String getName() { return "Cube"; }

   public double getSquareArea() { return s.area(); }

   public String getSquareName() { return s.getName(); }

   public String getSquareString() { return s.toString(); }

   public String getSPointString() { return s.getPointString(); }

   public String getSPointName() { return s.getPointName(); }
}
```

```
// Exercise 27.9 -- Composition
// Test.java
// Driver for point, square, cube composition program
import javax.swing.*;

public class Test {
   public static void main( String args[] )
   {
      Cube cube = new Cube( 3.3, 10, 10 );
      String result = "";

      result += cube.getSPointName() + ": " +
                     cube.getSPointString();

      result += "\n" + cube.getSquareName() + ": " +
                     cube.getSquareString();

      result += "\n" + cube.getName() + ": " +
                     cube.toString();

      result += "\n" + cube.getSPointName() +
                     ": " + cube.getSPointString();

      result += "\n" + cube.getSquareName() +
                     ": " + cube.getSquareString();
      result += "\n" + "Area = " + cube.getSquareArea();

      result += "\n" + cube.getName() +
                     ": " + cube.toString();
      result += "\n" + "Area = " + cube.area();
      result += "\n" + "Volume = " + cube.volume();
```

```

      JOptionPane.showMessageDialog(
            null, result, "Shapes",
            JOptionPane.INFORMATION_MESSAGE );
      System.exit( 0 );
   }
}
```



ANS:

```
// Exercies 27.9 -- Inheritance
// Point.java
// Definition of class Point

public class Point extends Shape {
   protected double x, y;

   public Point( double a, double b ) { setPoint( a, b ); }

   public void setPoint( double a, double b )
   {
      x = a;
      y = b;
   }

   public double getX() { return x; }

   public double getY() { return y; }

   public String toString()
      { return "[" + x + ", " + y + "]"; }

   public String getName() { return "Point"; }
}
```

```
// Exercies 27.9 -- Inheritance
// Shape.java
// Definition of abstract base class Shape

public abstract class Shape {
   public double area() { return 0.0; }
   public double volume() { return 0.0; }
   public abstract String getName();
}
```

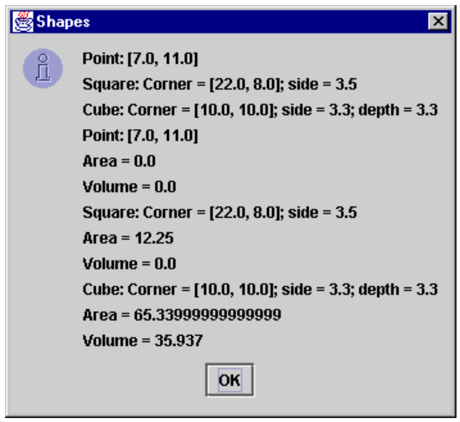
```
// Exercies 27.9 -- Inheritance
// Square.java
// Definition of class Square

public class Square extends Point {
   protected double side;

   public Square()
      { this( 0.0, 0.0, 0.0 ); }

   public Square( double s, double a, double b )
   {
      super( a, b );
      setSide( s );
   }

   public void setSide( double s )
      { side = ( s >= 0 ? s : 0 ); }

   public double getSide() { return side; }

   public double area() { return Math.pow( side, 2 ); }

   public String toString()
      { return "Corner = " + super.toString() +
               "; side = " + side; }

```

```
   public String getName() { return "Square"; }
}
```

```
// Exercies 27.9 -- Inheritance
// Cube.java
// Definition of class Cylinder

public class Cube extends Square {
   private double depth;

   public Cube( double s, double a, double b )
   {
      super( s, a, b );
      depth = s;
   }

   public double area() { return super.area() * 6; }

   public double volume() { return super.area() * depth; }

   public String toString()
      { return super.toString() + "; depth = " + depth; }

   public String getName() { return "Cube"; }
}
```

```
// Exercies 27.9 -- Inheritance
// Test.java
// Driver for point, square, cube hierarchy
import javax.swing.*;

public class Test {
   public static void main( String args[] )
   {
      Point point = new Point( 7, 11 );
      Square square = new Square( 3.5, 22, 8 );
      Cube cube = new Cube( 3.3, 10, 10 );

      Shape[] arrayOfShapes = new Shape[ 3 ];
      String result = "";

      arrayOfShapes[ 0 ] = point;
      arrayOfShapes[ 1 ] = square;
      arrayOfShapes[ 2 ] = cube;

      result += point.getName() + ": " +
                     point.toString();

      result += "\n" + square.getName() + ": " +
                     square.toString();

      result += "\n" + cube.getName() + ": " +
                     cube.toString();

      for ( int i = 0; i < 3; i++ ) {
         result += "\n" + arrayOfShapes[ i ].getName() +
            ": " + arrayOfShapes[ i ].toString();
          result += "\n" + "Area = " +
            arrayOfShapes[ i ].area();
         result += "\n" + "Volume = " +
            arrayOfShapes[ i ].volume();
      }

```

```
        JOptionPane.showMessageDialog(
              null, result, "Shapes",
              JOptionPane.INFORMATION_MESSAGE );
        System.exit( 0 );
     }
  }
```



27.10 In the chapter, we stated, "When a superclass method is inappropriate for a subclass, that method can be overridden in the subclass with an appropriate implementation." If this is done, does the subclass-is-a-superclass-object relationship still hold? Explain your answer.

ANS: Yes, the subclass-is-a-superclass-object relationship still holds. In Java, it is not possible to break this relationship.

27.11 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 a class hierarchy without the knowledge of individual details of each data type. The general processing capabilities are separated from the internal details of each type. Programming in the gernaral enables you to write more maintainable and modifyable systems. New data types can be added into the system as long as they belong to the portion of the class hierarchy being polymorphically processed.

27.12 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 the extensibility and maintainability of the program. A program containing many `switch` structures is difficult to modify. All the structures must be modified to handle the processing of an additional type or of one less type. Polymorphism determines the type of an object automatically, so it is not necessary to determine the type of an object to process the object in a generic manner.

27.13 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 amount of new code that is being written. Such hierarchies are commonly 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 capabilities of the program).

27.14 Distinguish between non-`abstract` methods and `abstract` methods.

ANS: A non-`abstract` method provides implementation. An `abstract` method does not provide any implementation.

27.15 (*True/False*) All methods in an `abstract` superclass must be declared `abstract`.

ANS: False. An `abstract` class must have at least one `abstract` method. Any number of methods in the class can be non-`abstract`.

27.16 Suggest one or more levels of `abstract` superclasses for the `Shape` hierarchy discussed in the beginning of this chapter (the first level is `Shape` and the second level consists of the classes `TwoDimensionalShape` and `ThreeDimensionalShape`).

27.17 How does polymorphism promote extensibility?

ANS: Polymorphism makes programs more extensible by making all method calls generic. When a new class type with the appropriate methods is added to the hierarchy, no changes need to be made to the generic method calls to enable processing of the new data type.

27.18 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.

27.19 *(Drawing Application)* Modify the drawing program of Exercise 26.11 to create a drawing application that draws random lines, rectangles and ovals. [*Note:* Like an applet, a `JFrame` has a `paint` method that you can override to draw on the background of the `JFrame`.]

For this exercise, modify the `MyLine`, `MyOval` and `MyRect` classes of Exercise 26.11 to create the class hierarchy in Fig. 27.8. The classes of the `MyShape` hierarchy should be "smart" shape classes where objects of these classes know how to draw themselves (if provided with a `Graphics` object that tells them where to draw). The only `switch` or `if/else` logic in this program should be to determine the type of shape object to create (use random numbers to pick the shape type and the coordinates of each shape). Once an object from this hierarchy is created, it will be manipulated for the rest of its lifetime as a superclass `MyShape` reference.

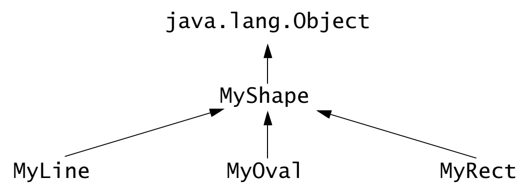


Fig. 27.8 The `MyShape` hierarchy.

Class `MyShape` in Fig. 27.8 *must* be `abstract`. The only data representing the coordinates of the shapes in the hierarchy should be defined in class `MyShape`. Lines, rectangles and ovals can all be drawn if you know two points in space. Lines require *x1*, *y1*, *x2* and *y2* coordinates. The `drawLine` method of the `Graphics` class will connect the two points supplied with a line. If you have the same four coordinate values (*x1*, *y1*, *x2* and *y2*) for ovals and rectangles, you can calculate the four arguments needed to draw them. Each requires an upper-left *x*-coordinate value (minimum of the two *x*-coordinate values), an upper-left *y*-coordinate value (minimum of the two *y*-coordinate values), a *width* (difference between the two *x*-coordinate values; must be nonnegative) and a *height* (difference between the two *y*-coordinate values; must be nonnegative). [*Note*: In Chapter 29, each *x*,*y* pair will be captured using mouse events from mouse interactions between the user and the program's background. These coordinates will be stored in an appropriate shape object as selected by the user. As you begin the exercise, you will use random coordinate values as arguments to the constructor.]

In addition to the data for the hierarchy, class `MyShape` should define at least the following methods:

a) A constructor with no arguments that sets the coordinates to 0.
b) A constructor with arguments that sets the coordinates to the supplied values.
c) *Set* methods for each individual piece of data that allow the programmer to independently set any piece of data for a shape in the hierarchy (e.g., if you have an instance variable `x1`, you should have a method `setX1`).
d) *Get* methods for each individual piece of data that allow the programmer to independently retrieve any piece of data for a shape in the hierarchy (e.g., if you have an instance variable `x1`, you should have a method `getX1`).
e) The `abstract` method

```
public abstract void draw( Graphics g );
```

This method will be called from the program's `paint` method to draw a shape onto the screen.

The preceding methods are required. If you would like to provide more methods for flexibility, please do so. However, be sure that any method you define in this class is a method that would be used by *all* shapes in the hierarchy.

All data *must* be `private` to class `MyShape` in this exercise (this forces you to use proper encapsulation of the data and provide proper *set/get* methods to manipulate the data). You are not allowed to define new data that can be derived from existing information. As explained previously, the upper-left *x*, upper-left *y*, *width* and *height* needed to draw an oval or rectangle can be calculated if you already know two points in space. All subclasses of `MyShape` should provide two constructors that mimic those provided by class `MyShape`.

Objects of the `MyOval` and `MyRect` classes should not calculate their upper-left *x*-coordinate, upper-left *y*-coordinate, *width* and *height* until they are about to draw. Never modify the *x1*, *y1*, *x2* and *y2* coordinates of a `MyOval` or `MyRect` object to prepare to draw them. Instead, use the temporary results of the calculations described above. This will help us enhance the program in Chapter 29 by allowing the user to select each shape's coordinates with the mouse.

There should be no `MyLine`, `MyOval` or `MyRect` references in the program—only `MyShape` references that refer to `MyLine`, `MyOval` and `MyRect` objects are allowed. The program should keep an array of `MyShape` references containing all shapes. The program's `paint` method should walk through the array of `MyShape` references and draw every shape (i.e., call every shape's `draw` method).

Begin by defining class `MyShape`, class `MyLine` and an application to test your classes. The application should have a `MyShape` instance variable that can refer to one `MyLine` object (created in the application's constructor). The `paint` method (for your subclass of `JFrame`) should draw the shape with a statement like

```
currentShape.draw( g );
```

where `currentShape` is the `MyShape` reference and `g` is the `Graphics` object that the shape will use to draw itself on the background of the window.

Next, change the single `MyShape` reference into an array of `MyShape` references and hard code several `MyLine` objects into the program for drawing. The application's `paint` method should walk through the array of shapes and draw every shape.

After the preceding part is working, you should define the `MyOval` and `MyRect` classes and add objects of these classes into the existing array. For now, all the shape objects should be created in the constructor for your subclass of `JFrame`. In Chapter 29, we will create the objects when the user chooses a shape and begins drawing it with the mouse.

ANS:

```
// Exercise 27.19 Solution
// MyShape.java

import java.awt.Graphics;

public abstract class MyShape extends Object {
   private int x1, x2, y1, y2;

   public MyShape()
   {
      setX1( 0 );
      setX2( 0 );
      setY1( 0 );
      setY2( 0 );
   }

   public MyShape( int x1, int y1, int x2, int y2 )
   {
      setX1( x1 );
      setX2( x2 );
      setY1( y1 );
      setY2( y2 );
   }

   public void setX1( int x1 ) { this.x1 = ( x1 >= 0 ? x1 : 0 ); }
   public void setX2( int x2 ) { this.x2 = ( x2 >= 0 ? x2 : 0 ); }
   public void setY1( int y1 ) { this.y1 = ( y1 >= 0 ? y1 : 0 ); }
   public void setY2( int x2 ) { this.y2 = ( y2 >= 0 ? y2 : 0 ); }

   public int getX1() { return x1; }
   public int getX2() { return x2; }
   public int getY1() { return y1; }
   public int getY2() { return y2; }

   public abstract void draw( Graphics g );
}
```

```
// Exercise 27.19 Solution
// MyLine.java
// Definition of class MyLine
import java.awt.Graphics;

public class MyLine extends MyShape {

    public MyLine()
    {
       super ();
    }

    public MyLine( int x1, int y1, int x2, int y2 )
    {
       super (x1, y1, x2, y2);
    }

    public void draw( Graphics g )
    {
       g.drawLine( getX1(), getY1(), getX2(), getY2() );
    }
}
```

```
// Exercise 27.19 Solution
// MyOval.java
import java.awt.Graphics;

public class MyOval extends MyShape {

   public MyOval()
   {
      super();
   }

   public MyOval( int x1, int y1, int x2, int y2 )
   {
      super( x1, y1, x2, y2 );
   }

   public void draw( Graphics g )
   {
      g.drawOval( Math.min( getX1(), getX2() ),
          Math.min( getY1(), getY2() ),
          Math.abs( getY2() - getY1() ),
          Math.abs( getX2() - getX1() ) );
   }
}
```

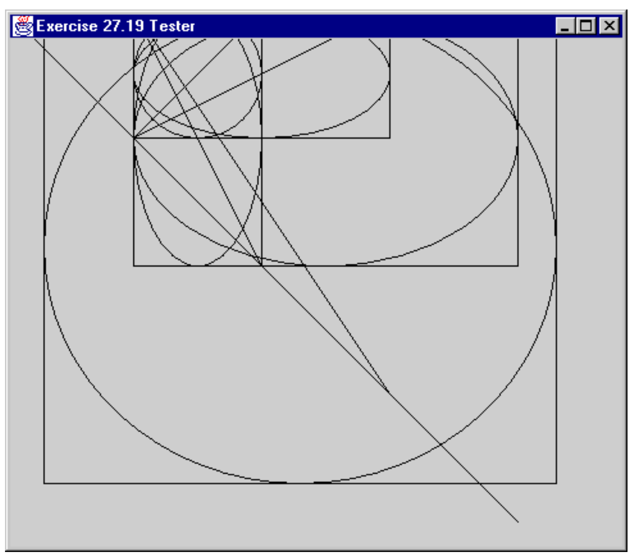
```
// Exercise 27.19 Solution
// MyRect.java
import java.awt.Graphics;

public class MyRect extends MyShape {

   public MyRect()
   {
      super ();
   }

   public MyRect( int x1, int y1, int x2, int y2 )
   {
      super( x1, y1, x2, y2 );
   }

```

```java
   public void draw( Graphics g )
   {
      g.drawRect( Math.min( getX1(),getX2() ),
          Math.min( getY1(), getY2() ),
          Math.abs( getY2() - getY1() ),
          Math.abs( getX2() - getX1() ) );
   }
}
```

```java
// Exercise 27.19 Solution
// MyShape.java

import java.awt.Graphics;

public abstract class MyShape extends Object {
   private int x1, x2, y1, y2;

   public MyShape()
   {
      setX1( 0 );
      setX2( 0 );
      setY1( 0 );
      setY2( 0 );
   }

   public MyShape( int x1, int y1, int x2, int y2 )
   {
      setX1( x1 );
      setX2( x2 );
      setY1( y1 );
      setY2( y2 );
   }

   public void setX1( int x1 ) { this.x1 = ( x1 >= 0 ? x1 : 0 ); }
   public void setX2( int x2 ) { this.x2 = ( x2 >= 0 ? x2 : 0 ); }
   public void setY1( int y1 ) { this.y1 = ( y1 >= 0 ? y1 : 0 ); }
   public void setY2( int x2 ) { this.y2 = ( y2 >= 0 ? y2 : 0 ); }

   public int getX1() { return x1; }
   public int getX2() { return x2; }
   public int getY1() { return y1; }
   public int getY2() { return y2; }

   public abstract void draw( Graphics g );
}
```

```java
// Exercise 27.19 Solution
// TestDrawWindow.java

import java.awt.*;
import javax.swing.*;

public class TestDrawWindow extends JFrame {
   private MyShape shape[];

   public TestDrawWindow()
   {
      super( "Exercise 9.28 Tester" );
      shape = new MyShape[ 15 ];

      shape[ 0 ] = new MyLine( 100, 100, 200, 200 );
      shape[ 1 ] = new MyLine( 200, 200, 100, 100 );
      shape[ 2 ] = new MyLine( 300, 300, 100, 100 );
      shape[ 3 ] = new MyLine( 400, 400, 0, 0 );
```

```
      shape[ 4 ] = new MyLine( 100, 100, 300, 300 );

      shape[ 5 ] = new MyOval( 100, 100, 200, 200 );
      shape[ 6 ] = new MyOval( 200, 200, 100, 100 );
      shape[ 7 ] = new MyOval( 300, 300, 100, 100 );
      shape[ 8 ] = new MyOval( 400, 400, 30, 200 );
      shape[ 9 ] = new MyOval( 100, 100, 300, 300 );

      shape[ 10 ] = new MyRect( 100, 100, 200, 200 );
      shape[ 11 ] = new MyRect( 200, 200, 100, 100 );
      shape[ 12 ] = new MyRect( 300, 300, 100, 100 );
      shape[ 13 ] = new MyRect( 400, 400, 30, 200 );
      shape[ 14 ] = new MyRect( 100, 100, 300, 300 );

      setDefaultCloseOperation( WindowConstants.DISPOSE_ON_CLOSE );
   }

   public static void main( String args[] )
   {
      TestDrawWindow window = new TestDrawWindow();
      window.setSize( 500, 500 );
      window.show();
   }

   public void paint( Graphics g )
   {
      for ( int i = 0; i < shape.length; i++ )
         shape[ i ].draw( g );
   }
}
```



28

Java Graphics and Java2D: Solutions

SOLUTIONS

28.4 Fill in the blanks in each of the following:

a) Class ________ of the Java2D API is used to define ovals.

ANS: `Ellipse2D`

b) Methods `draw` and `fill` of class `Graphics2D` require an object of type ________ as their argument.

ANS: `Shape`

c) The three constants that specify font style are ________, ________ and ________.

ANS: `Font.PLAIN`, `Font.BOLD` and `Font.ITALIC`

d) `Graphics2D` method ________ sets the painting color for Java2D shapes.

ANS: `setColor`

28.5 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The `drawPolygon` method automatically connects the endpoints of the polygon.

ANS: True.

b) The `drawLine` method draws a line between two points.

ANS: True.

c) The `fillArc` method uses degrees to specify the angle.

ANS: True.

d) In the Java coordinate system, *y* values increase from top to bottom.

ANS: True.

e) The `Graphics` class inherits directly from class `Object`.

ANS: True.

f) The `Graphics` class is an `abstract` class.

ANS: True.

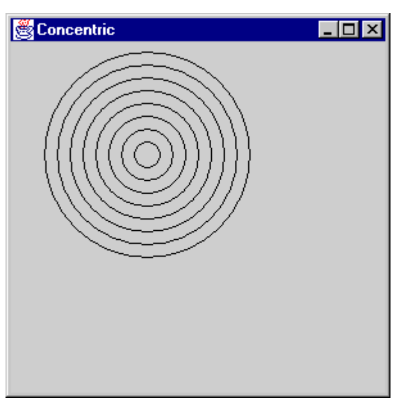
g) The `Font` class inherits directly from class `Graphics`.

ANS: False. Class `Font` inherits directly from class `Object`.

28.6 Write a program that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use the `drawOval` method of class `Graphics`.

ANS:

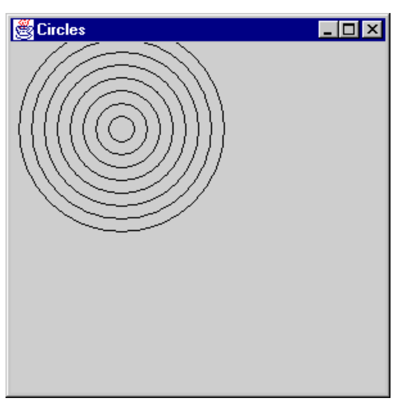
```
// Exercise 28.6 Solution
// Concentric.java
// This program draws concentric circles
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Concentric extends JFrame {

   public Concentric()
   {
      super( "Concentric" );
      setSize( 300, 300 );
      show();
   }

   public void paint( Graphics g )
   {
      for ( int x = 0; x <= 160; x += 10 ) {
         int y = 160 - ( x * 2 );
         g.drawOval( x + 30, x + 30, y, y );
      }
   }

   public static void main( String args[] )
   {
      Concentric app = new Concentric();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```



28.7 Write a program that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use the `drawArc` method.

ANS:

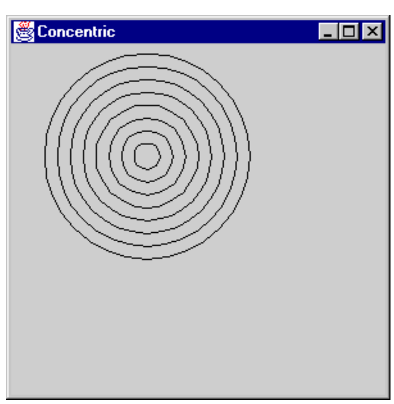
```
// Exercise 28.7 Solution
// Circles.java
// This program draws concentric circles
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Circles extends JFrame {

   public Circles()
   {
      super( "Circles" );
      setSize( 300, 300 );
      show();
   }

   public void paint( Graphics g )
   {
      for ( int x = 0; x <= 160; x += 10 ) {
         int y = 160 - ( x * 2 );

         g.drawArc( x + 10, x + 10, y, y, 0, 360 );
      }
   }
   public static void main( String args[] )
   {
      Circles app = new Circles();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```



28.8 Modify your solution to Exercise 28.6 to draw the ovals using instances of class `Ellipse2D.Double` and method `draw` of class `Graphics2D`.

ANS:

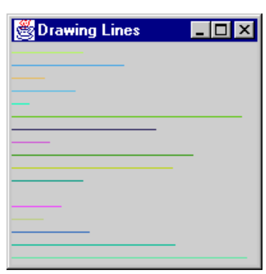
```
// Exercise 28.8 Solution
// Concentric.java
// This program draws concentric circles
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;

public class Concentric extends JFrame {

   public Concentric()
   {
      super( "Concentric" );
      setSize( 300, 300 );
      show();
   }

   public void paint( Graphics g )
   {
       // Create 2D by casting g to Graphics 2D
       Graphics2D g2d = ( Graphics2D ) g;

       for ( int x = 0; x <= 160; x += 10 ) {
           int y = 160 - ( x * 2 );
           g2d.draw( new Ellipse2D.Double( x + 30, x + 30, y, y ) );
       }
   }

   public static void main( String args[] )
   {
      Concentric app = new Concentric();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```



28.9 Write a program that draws lines of random lengths in random colors.
ANS:

```
// Exercise 28.9 Solution
// Lines1.java
// This program draws lines of random sizes and colors
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Lines1 extends JFrame {

   public Lines1()
   {
      super( "Drawing Lines" );
      setSize( 200, 200 );
      show();
   }

   public void paint( Graphics g )
   {
      for ( int y = 10; y < 200; y += 10 ) {
         int x1 = ( int ) ( 1 + Math.random() * 199 );

         g.setColor( new Color( ( float ) Math.random(),
            ( float ) Math.random(), ( float ) Math.random() ) );
         g.drawLine( 1, y, x1, y );
      }
   }

   public static void main( String args[] )
   {
      Lines1 app = new Lines1();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

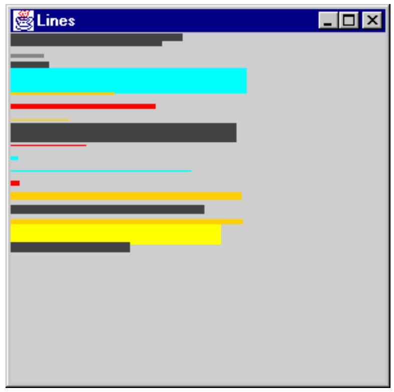
t



28.10 Modify your solution to Exercise 28.9 to draw random lines, in random colors and random line thicknesses. Use class `Line2D.Double` and method `draw` of class `Graphics2D` to draw the lines.

ANS:

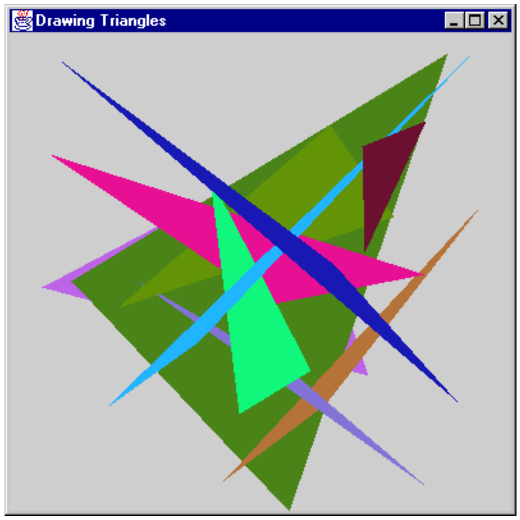
```
// Exercise 28.10 Solution
// Lines.java
// This program draws lines of different colors
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class Lines extends JFrame {
   private Color colors[] = { Color.green, Color.cyan,
                              Color.black, Color.yellow,
                              Color.darkGray, Color.red,
                              Color.orange, Color.gray,
                              Color.pink, Color.magenta };

   public Lines()
   {
      super( "Lines" );
      setSize( 300, 300 );
      show();
   }

   public void paint( Graphics g )
   {
      // Create 2D by casting g to Graphics 2D
      Graphics2D g2d = ( Graphics2D ) g;

      for ( int y = 10; y < 200; y += 10 ) {
         int color = ( int ) ( Math.random() * 9 );

         g2d.setColor( colors[ color ] );
         int thickness = ( int ) ( Math.random() * 20 + 1 );
         g2d.setStroke( new BasicStroke( thickness ) );
         int x1 = ( int ) ( 1 + Math.random() * 199 );
         g2d.draw( new Line2D.Double( 1, y, x1, y ) );
      }
   }

   public static void main( String args[] )
   {
      Lines app = new Lines();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```


Lines

28.11 Write a program that displays randomly generated triangles in different colors. Each triangle should be filled with a different color. Use class `GeneralPath` and method `fill` of class `Graphics2D` to draw the triangles.

ANS:

```
// Exercise 28.11 Solution
// Triangles.java
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.awt.geom.*;

public class Triangles extends JFrame {

   public Triangles()
   {
      super( "Drawing Triangles" );
      setSize( 400, 400 );
      show();
   }

   public void paint( Graphics g )
   {
      GeneralPath triangle = new GeneralPath();

      for ( int i = 0; i < 10; i++ ) {
         // create a triangle from three random points
         int x = ( int ) ( Math.random() * 375 + 25 );
         int y = ( int ) ( Math.random() * 375 + 25 );
         triangle.moveTo( x, y );

         // second point
         x = ( int ) ( Math.random() * 375 + 25 );
         y = ( int ) ( Math.random() * 375 + 25 );
         triangle.lineTo( x, y );

         // third point
         x = ( int ) ( Math.random() * 375 + 25 );
         y = ( int ) ( Math.random() * 375 + 25 );
         triangle.lineTo( x, y );

         Graphics2D g2d = ( Graphics2D ) g;

         // close the shape
         triangle.closePath();

         // choose a random color
         g2d.setColor(new Color( ( int ) ( Math.random() * 256 ),
                                 ( int ) ( Math.random() * 256 ),
                                 ( int ) ( Math.random() * 256 ) ) );
         g2d.fill( triangle );
         triangle.reset();
      }
   }

   public static void main( String args[] )
   {
      Triangles app = new Triangles();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

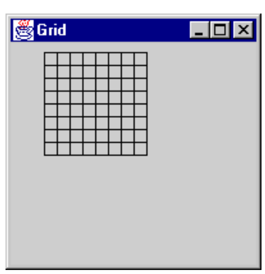

Drawing Triangles

28.12 Write a program that randomly draws characters in different font sizes and colors.
ANS:

```java
// Exercise 28.12 Solution
// Draw.java
// This program randomly draws characters
// Note: cover, resize, or restart the program
// repeatedly to see multiple characters drawn
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw extends JFrame {
   private final int DELAY = 4000000;

   public Draw()
   {
      super( "Drawing Characters" );
      setSize( 380, 150 );
      show();
   }

   public void paint( Graphics g )
   {
      int fontSize = ( int ) ( 10 + Math.random() * 63 );
      int x = ( int ) ( 30 + Math.random() * 341 );
      int y = ( int ) ( 50 + Math.random() * 95 );
      char letters[] = { 'V', 'O', 'L', 'S', '8', '7' };
      Font f = new Font( "Monospaced", Font.BOLD, fontSize );

      g.setColor( new Color( ( float ) Math.random(),
                             ( float ) Math.random(),
                             ( float ) Math.random() ) );
      g.setFont( f );
      g.drawChars( letters, ( int ) ( Math.random() * 6 ), 1, x, y );

      for ( int h = 1; h < DELAY; h++ ) ;  // slow things down
      repaint();
   }

   public static void main( String args[] )
   {
      Draw app = new Draw();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

28.13 Write a program that draws an 8-by-8 grid. Use the `drawLine` method.
ANS:

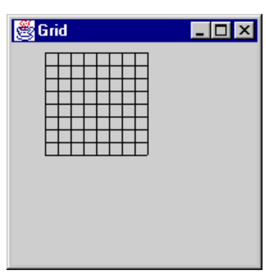
```
// Exercise 28.13 Solution
// Grid.java
// This program draws an 8 x 8 grid
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Grid extends JFrame {

   public Grid()
   {
      super( "Grid" );
      setSize( 200, 200 );
      show();
   }

   public void paint( Graphics g )
   {
      int y = 30, x1 = 30;

      // 9 lines are required for an 8 x 8 grid
      for ( int r = 1; r <= 9; r++, y += 10 )
         g.drawLine( 30, y, 110, y );

      for ( int c = 1; c <= 9; c++, x1 += 10 )
         g.drawLine( x1, 30, x1, 110 );
   }

   public static void main( String args[] )
   {
      Grid app = new Grid();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```



28.14 Modify your solution to Exercise 28.13 to draw the grid using instances of class `Line2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
// Exercise 28.14 Solution
// Grid.java
// This program draws an 8 x 8 grid
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class Grid extends JFrame {

   public Grid()
   {
      super( "Grid" );
      setSize( 200, 200 );
      show();
   }

   public void paint( Graphics g )
   {
      int y = 30, x1 = 30;
      Graphics2D g2d = ( Graphics2D ) g;

      // 9 lines are required for an 8 x 8 grid
      for ( int r = 1; r <= 9; r++, y += 10 )
         g2d.draw( new Line2D.Double( 30, y, 110, y ) );

      for ( int c = 1; c <= 9; c++, x1 += 10 )
         g2d.draw( new Line2D.Double(  x1, 30, x1, 110 ) );
   }

   public static void main( String args[] )
   {
      Grid app = new Grid();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

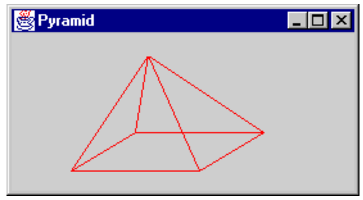


28.15 Write a program that draws a 10-by-10 grid. Use the `drawRect` method.

28.16 Modify your solution to Exercise 28.15 to draw the grid using instances of class `Rectangle2D.Double` and method `draw` of class `Graphics2D`.

28.17 Write a program that draws a tetrahedron (a pyramid). Use class `GeneralPath` and method `draw` of class `Graphics2D`.
ANS:

```java
// Exercise 28.17 Solution
// Pyramid.java
// This program draws a tetrahedron
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;

public class Pyramid extends JFrame {

   public Pyramid()
   {
      super( "Pyramid" );
      setSize( 275, 150 );
      show();
   }

   public void paint( Graphics g )
   {
      int basex[] = { 100, 200, 150, 50, 100 };
      int basey[] = { 100, 100, 130, 130, 100 };
      int x = 110, y = 40;

      Graphics2D g2d = ( Graphics2D ) g;

      GeneralPath tetra = new GeneralPath();

      g2d.setColor( Color.red );

      tetra.moveTo( basex[ 0 ], basey[ 0 ] );

      for ( int i = 1; i < 5; i++ ) {
         tetra.lineTo( x, y );
         tetra.moveTo( basex[ i - 1 ], basey[ i - 1 ] );
         tetra.lineTo( basex[ i ], basey[ i ] );
      }

      tetra.closePath();
      g2d.draw( tetra );
   }

   public static void main( String args[] )
   {
      Pyramid app = new Pyramid();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```



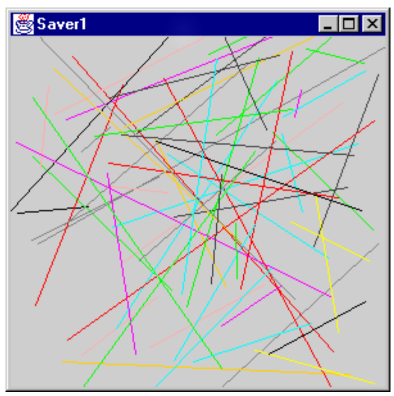
28.18 Write a program that draws a cube. Use class `GeneralPath` and method `draw` of class `Graphics2D`.

28.19 Write an application that simulates a screen saver. The application should randomly draw lines using method `drawLine` of class `Graphics`. After drawing 100 lines, the application should clear itself and start drawing lines again. To allow the program to draw continuously, place a call to `repaint` as the last line in method `paint`. Do you notice any problems with this on your system?

ANS:

```
// Exercise 28.19 Solution
// Saver1.java
// Program simulates a simple screen saver
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class Saver1 extends JFrame {
   private final int DELAY = 4000000;
   private final int XDIM = 300;
   private final int YDIM = 300;
   private int count;

   public Saver1()
   {
      super( "Saver1" );
      setSize( 300, 300 );
      count = 0;
      show();
   }

   public void paint( Graphics g )
   {
      int x, y, x1, y1;
      Color colors[] = { Color.green, Color.cyan,
                         Color.black, Color.yellow,
                         Color.darkGray, Color.red,
                         Color.orange, Color.gray,
                         Color.pink, Color.magenta };

      // assume html size is 200 x 200
      x = ( int ) ( Math.random() * XDIM );
      y = ( int ) ( Math.random() * YDIM );
      x1 = ( int ) ( Math.random() * XDIM );
      y1 = ( int ) ( Math.random() * YDIM );

      g.setColor(  colors[( int ) ( Math.random() * colors.length )] );
      g.drawLine( x, y, x1, y1 );
      ++count;

      // slow the drawing down
      for ( int q = 1; q < DELAY; q++ )
         ;  // do nothing

      if ( count == 100 ) {
         g.setColor( Color.white );
         g.fillRect( 0, 0, XDIM, YDIM );
         count = 0;
      }
```

```

      repaint();
   }

   public static void main( String args[] )
   {
      Saver1 app = new Saver1();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```



28.20 Here is a peek ahead. Package `javax.swing` contains a class called `Timer` that is capable of calling method `actionPerformed` of interface `ActionListener` at a fixed time interval (specified in milliseconds). Modify your solution to Exercise 28.19 to remove the call to `repaint` from method `paint`. Define your class so it implements `ActionListener` (the `actionPerformed` method should simply call `repaint`). Define an instance variable of type `Timer` called `timer` in your class. In the constructor for your class, write the following statements:

```
timer = new Timer( 1000, this );
timer.start();
```

This creates an instance of class `Timer` that will call `this` object's `actionPerformed` method every `1000` milliseconds (i.e., every second).

28.21 Modify your solution to Exercise 28.20 to enable the user to enter the number of random lines that should be drawn before the application clears itself and starts drawing lines again. Use a `JTextField` to obtain the value. The user should be able to type a new number into the `JTextField` at any time during the program's execution. [*Note*: Combining Swing GUI components and drawing leads to interesting problems for which we present solutions in Chapter 29]. For now, the first line of your `paint` method should be

```
super.paint( g );
```

to ensure that the GUI components are displayed properly. You will notice that some of the randomly drawn lines will obscure the `JTextField`. Use an inner class definition to perform event handling for the `JTextField`.

28.22 Modify your solution to Exercise 28.20 to randomly choose different shapes to display (use methods of class `Graphics`).]

28.23 Modify your solution to Exercise 28.22 to use classes and drawing capabilities of the Java2D API. For shapes such as rectangles and ellipses, draw them with randomly generated gradients (use class `GradientPaint` to generate the gradient).

28.24 Write a program that uses method `drawPolyline` to draw a spiral.

28.25 Write a program that inputs four numbers and graphs the numbers as a pie chart. Use class `Arc2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each piece of the pie in a separate color.

28.26 Write an applet that inputs four numbers and graphs the numbers as a bar graph. Use class `Rectangle2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each bar in a different color.

29

Java Graphical User Interface Components: Solutions

SOLUTIONS

29.4 Fill in the blanks in each of the following:

a) The `JTextField` class inherits directly from ________.

ANS: `JTextComponent`.

b) The layout managers discussed in this chapter are ________, ________ and ________.

ANS: `FlowLayout`, `BorderLayout` and `GridLayout`.

c) `Container` method ________ attaches a GUI component to a container.

ANS: `add`.

d) Method ________ is called when a mouse button is released (without moving the mouse).

ANS: `mouseClicked`.

29.5 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Only one layout manager can be used per `Container`.

ANS: True.

b) GUI components can be added to a `Container` in any order in a `BorderLayout`.

ANS: True.

c) `Graphics` method `setFont` is used to set the font for text fields.

ANS: False. `Component` method `setFont` is used.

d) A `Mouse` object contains a method called `mouseDragged`.

ANS: False. A `Mouse` object is not provided by Java.

29.6 State whether each of the following is *true* or *false*. If *false*, explain why.

a) A `JApplet` does not have a content pane.

ANS: False. A `JApplet` does have a content pane.

b) A `JPanel` is a `JComponent`.

ANS: True.

c) A `JPanel` is a `Component`.

ANS: True.

d) A `JLabel` is a `Container`.

ANS: True.

e) An `AbstractButton` is a `JButton`.

ANS: False. A `JButton` is an `AbstractButton`.

f) A `JTextField` is an `Object`.

ANS: True.

29.7 Find any error(s) in each of the following and explain how to correct it (them).

a) `import javax.swing.*       // include swing package`

ANS: Semicolon is missing after the asterick.

b) `panelObject.GridLayout( 8, 8 ); // set GridLayout`

ANS: The GridLayout constructor cannot be used in this manner. The correct statement should be: `panelObject.getContentPane().setLayout( new GridLayout( 8, 8 ) );`

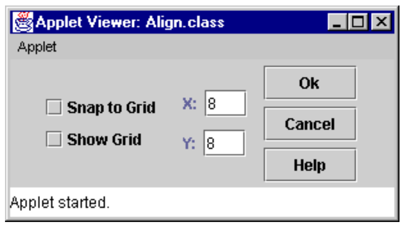
c) `c.setLayout( new FlowLayout( FlowLayout.DEFAULT ) );`

ANS: Class `FlowLayout` does not contain `static` constant `DEFAULT`.

d) `c.add( eastButton, EAST );  // BorderLayout`

ANS: `EAST` should be `BorderLayout.EAST`.

29.8 Create the following GUI. You do not have to provide any functionality.

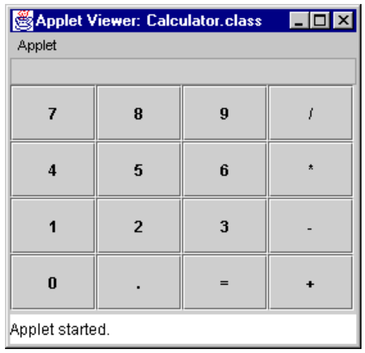


ANS:

```java
// Exercise 29.8 Solution
// Align.java
// This program creates a simple GUI
import javax.swing.*;
import java.awt.*;

public class Align extends JApplet {
   private JButton ok, cancel, help;
   private JTextField xValue, yValue;
   private JCheckBox snap, show;
   private JLabel xLabel, yLabel;
   private JPanel checkPanel, buttonPanel,
                  fieldPanel1, fieldPanel2,
                  fieldPanel;

   public void init()
   {
      // build checkPanel
      snap = new JCheckBox( "Snap to Grid" );
      show = new JCheckBox( "Show Grid" );
      checkPanel = new JPanel();
      checkPanel.setLayout( new GridLayout( 2 , 1 ) );
      checkPanel.add( snap );
      checkPanel.add( show );

      // build field panel1
      xLabel = new JLabel( "X: " );
      xValue = new JTextField( "8", 3 );
      fieldPanel1 = new JPanel();
      fieldPanel1.setLayout( new FlowLayout( FlowLayout.CENTER, 3, 5 ) );
      fieldPanel1.add( xLabel );
      fieldPanel1.add( xValue );

      yLabel = new JLabel( "Y: " );
      yValue = new JTextField( "8", 3 );
      fieldPanel2 = new JPanel();
      fieldPanel2.setLayout( new FlowLayout( FlowLayout.CENTER, 3, 5 ) );
      fieldPanel2.add( yLabel );
      fieldPanel2.add( yValue );

      fieldPanel = new JPanel();
      fieldPanel.setLayout( new BorderLayout() );
```

```
      fieldPanel.add( fieldPanel1, BorderLayout.NORTH );
      fieldPanel.add( fieldPanel2, BorderLayout.SOUTH );

      // build button panel
      ok = new JButton( "Ok" );
      cancel = new JButton( "Cancel" );
      help = new JButton( "Help" );
      buttonPanel = new JPanel();
      buttonPanel.setLayout( new GridLayout( 3, 1, 10, 5 ) );
      buttonPanel.add( ok );
      buttonPanel.add( cancel );
      buttonPanel.add( help );

      // set layout for applet
      getContentPane().setLayout(
            new FlowLayout( FlowLayout.CENTER, 10, 5 ) );
      getContentPane().add( checkPanel );
      getContentPane().add( fieldPanel );
      getContentPane().add( buttonPanel );
   }
}
```

29.9 Create the following GUI. You do not have to provide any functionality.

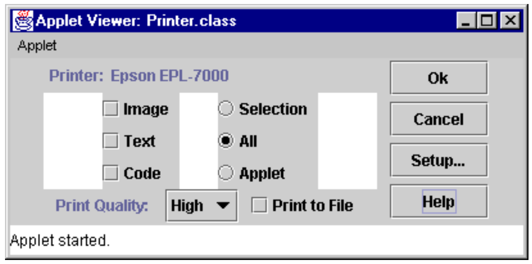


ANS:

```java
// Solution exercise 29.9
// Calculator.java
// This program creates a simple GUI
// html: width = 270 height = 200
import javax.swing.*;
import java.awt.*;

public class Calculator extends JApplet {
   private JButton keys[];
   private JPanel keyPad;
   private JTextField lcd;

   public void init()
   {
      lcd    = new JTextField( 20 );
      keyPad = new JPanel();
      keys   = new JButton[ 16 ];

      lcd.setEditable( false );

      for ( int i = 0; i <= 9; i++ )
         keys[ i ] = new JButton( String.valueOf( i ) );

      keys[ 10 ] = new JButton( "/" );
      keys[ 11 ] = new JButton( "*" );
      keys[ 12 ] = new JButton( "-" );
      keys[ 13 ] = new JButton( "+" );
      keys[ 14 ] = new JButton( "=" );
      keys[ 15 ] = new JButton( "." );

      // set keyPad layout to grid layout
      keyPad.setLayout( new GridLayout( 4, 4 ) );

      for ( int i = 7; i <= 10; i++ ) // 7, 8, 9, 10
         keyPad.add( keys[ i ] );     // divide

      for ( int i = 4; i <= 6; i++ )  // 4, 5, 6
         keyPad.add( keys[ i ] );

      keyPad.add( keys[ 11 ] );       // multiply

      for ( int i = 1; i <= 3; i++ )  // 1, 2, 3
         keyPad.add( keys[ i ] );

      keyPad.add( keys[ 12 ] );       // subtract

      keyPad.add( keys[ 0 ] );        // 0

      for ( int i = 15; i >= 13; i-- )
         keyPad.add( keys[ i ] );     // ., =, add
```

```

      // set applet layout to border layout
      getContentPane().setLayout( new BorderLayout() );
      getContentPane().add( lcd, BorderLayout.NORTH );
      getContentPane().add( keyPad, BorderLayout.CENTER );
   }
}
```

29.10 Create the following GUI. You do not have to provide any functionality.



ANS:

```java
// Exercise 29.10 Solution
// Printer.java
// This program creates a simple GUI
// html: width = 400 height = 130
import javax.swing.*;
import java.awt.*;

public class Printer extends JApplet {
   private JButton b1, b2, b3, b4;
   private JCheckBox c1, c2, c3, c4;
   private JRadioButton rb1, rb2, rb3;
   private ButtonGroup radioGroup;
   private JComboBox q;
   private JLabel label1, label2;
   private JPanel p1, p2, p3, p4, p5, p6, p7, p8;

   public void init()
   {
      // build left north panel
      label1 = new JLabel( "Printer:  Epson EPL-7000" );
      p1 = new JPanel();
      p1.setLayout( new FlowLayout( FlowLayout.LEFT ) );
      p1.add( label1 );

      // build right east panel
      b1 = new JButton( "Ok" );
      b2 = new JButton( "Cancel" );
      b3 = new JButton( "Setup..." );
      b4 = new JButton( "Help" );
      p2 = new JPanel();
      p2.setLayout( new GridLayout( 4, 1, 5, 5 ) );
      p2.add( b1 );
      p2.add( b2 );
      p2.add( b3 );
      p2.add( b4 );

      // build left south panel
      label2 = new JLabel( "Print Quality: " );
      q = new JComboBox();
      q.addItem( "High" );
      c1 = new JCheckBox( "Print to File" );
      p3 = new JPanel();
      p3.setLayout( new FlowLayout( FlowLayout.CENTER, 10, 0 ) );
      p3.add( label2 );
      p3.add( q );
      p3.add( c1 );

      // build left east panel
      c2 = new JCheckBox( "Image" );
      c3 = new JCheckBox( "Text" );
      c4 = new JCheckBox( "Code" );
      p4 = new JPanel();
```

```
      p4.setLayout( new BorderLayout( ) );
      p4.add( c2, BorderLayout.NORTH );
      p4.add( c3, BorderLayout.CENTER );
      p4.add( c4, BorderLayout.SOUTH );

      // build left west panel
      p5 = new JPanel();
      p5.setLayout( new BorderLayout() );
      p5.add( rb1 = new JRadioButton( "Selection", false ),
              BorderLayout.NORTH );
      p5.add( rb2 = new JRadioButton( "All", true ),
              BorderLayout.CENTER );
      p5.add( rb3 = new JRadioButton( "Applet", false ),
              BorderLayout.SOUTH );
      // Group the radio buttons
      radioGroup=new ButtonGroup();
      radioGroup.add( rb1 );
      radioGroup.add( rb2 );
      radioGroup.add( rb3 );

      // build left center
      p8 = new JPanel();
      p8.setLayout( new FlowLayout( FlowLayout.CENTER, 30, 0 ) );
      p8.setBackground( Color.white );
      p8.add( p4 );
      p8.add( p5 );

      // setup left panel
      p6 = new JPanel();
      p6.setLayout( new BorderLayout() );
      p6.add( p1, BorderLayout.NORTH );
      p6.add( p8, BorderLayout.CENTER );
      p6.add( p3, BorderLayout.SOUTH );

      // setup applet layout
      p7 = new JPanel();
      p7.setLayout( new FlowLayout( FlowLayout.CENTER, 10, 0 ) );
      p7.add( p6 );
      p7.add( p2 );

      getContentPane().add( p7 );
   }
}
```

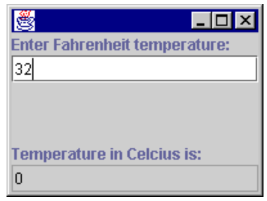
29.11 Write a temperature conversion program that converts from Fahrenheit to Celsius. The Fahrenheit temperature should be entered from the keyboard (via a `JTextField`). A `JLabel` should be used to display the converted temperature. Use the following formula for the conversion:

$$Celsius = \frac{5}{9} \times (Fahrenheit - 32)$$

ANS:

```
// Exercise 29.11 Solution
// Convert.java
// Temperature conversion program
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Convert extends JFrame {
   private JPanel p;
   private JLabel label1, label2;
   private JTextField temperatureF;
   private JTextField temperatureC;

   public Convert()
   {
      label1 = new JLabel( "Enter Fahrenheit temperature:" );
      label2 = new JLabel( "Temperature in Celcius is:" );
      temperatureF = new JTextField( 10 );
      temperatureF.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e)
    {
               int celcius, temp;

       temp = Integer.parseInt( temperatureF.getText() );
       celcius = ( int ) ( 5.0f / 9.0f * ( temp - 32 ) );
       temperatureC.setText( String.valueOf( celcius ) );
    }
 }
      );

      temperatureC = new JTextField( 10 );
      temperatureC.setEditable( false );

      p = new JPanel();
      p.setLayout( new BorderLayout() );
      p.add( temperatureF, BorderLayout.NORTH );
      p.add( label2, BorderLayout.SOUTH );

      Container c = getContentPane();
      c.setLayout( new BorderLayout() );
      c.add( label1, BorderLayout.NORTH );
      c.add( p, BorderLayout.CENTER );
      c.add( temperatureC, BorderLayout.SOUTH );
      setSize( 200, 150 );
      show();
   }

   public static void main ( String args[] )
   {
      Convert app = new Convert();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
```

```
    }
}
```


Enter Fahrenheit temperature:
32
Temperature in Celcius is:
0

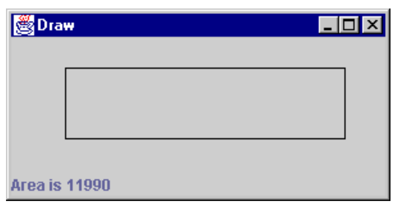
29.12 Write an application that allows the user to draw a rectangle by dragging the mouse on the application window. The upper-left coordinate should be the location where the user presses the mouse button, and the lower-right coordinate should be the location where the user releases the mouse button. Also display the area of the rectangle in a `JLabel` in the `SOUTH` region of a `BorderLayout`. All drawing should be done on a subclass of `JPanel`. Use the following formula for the area:

```
area = width × height
```

ANS:

```
// Exercise 29.12 Solution
// Draw.java
// Program draws a rectangle with the mouse
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw extends JFrame {
   private int topX, topY;
   private int width, height;
   private int bottomX, bottomY;
   protected JLabel status;

   public Draw()
   {
      super( "Draw" );
      topX = topY = 0;
      addMouseListener( new MouseHandler( this ) );

      status = new JLabel();
      getContentPane().add( status, BorderLayout.SOUTH );
      setSize( 300, 150 );
      show();
   }

   public int getTopX() { return topX; }
   public int getTopY() { return topY; }
   public int getWidth() { return width; }
   public int getHeight() { return height; }
   public int getBottomX() { return bottomX; }
   public int getBottomY() { return bottomY; }
   public void setTopX( int x ) { topX = x; }
   public void setTopY( int y ) { topY = y; }
   public void setBottomX( int x ) { bottomX = x; }
   public void setBottomY( int y ) { bottomY = y; }
   public void setWidth( int w ) { width = w; }
   public void setHeight( int h ) { height = h; }

   public void paint( Graphics g )
   {
      super.paint( g );

      g.drawRect( topX, topY, width, height );
   }

   public static void main( String args[] )
   {
      Draw app = new Draw();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

```java

class MouseHandler extends MouseAdapter {
   private Draw draw;

   public MouseHandler( Draw d ) { draw = d; }

   public void mouseReleased( MouseEvent e )
   {
      draw.setBottomX( e.getX() );
      draw.setBottomY( e.getY() );
      draw.setWidth( Math.abs( draw.getTopX() - draw.getBottomX() ) );
      draw.setHeight( Math.abs( draw.getTopY() - draw.getBottomY() ) );
      draw.setTopX( Math.min( draw.getTopX(), draw.getBottomX() ) );
      draw.setTopY( Math.min( draw.getTopY(), draw.getBottomY() ) );
      draw.status.setText( "Area is " + ( draw.getWidth() * draw.getHeight() ) );
      draw.repaint();
   }

   public void mousePressed( MouseEvent e )
   {
      draw.setTopX( e.getX() );
      draw.setTopY( e.getY() );
   }
}
```

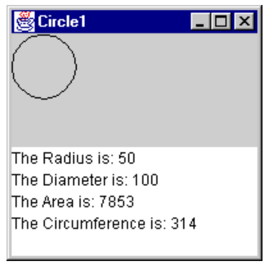


29.13 Write a program that displays a circle of random size and calculates and displays the area, radius, diameter and circumference. Use the following equations: *diameter* = $2 \times radius$, *area* = $\pi \times radius^2$, *circumference* = $2 \times \pi \times radius$. Use the constant `Math.PI` for pi (π). All drawing should be done on a subclass of `JPanel` and the results of the calculations should be displayed in a read-only `JTextArea`.

ANS:

```java
// Exercise 29.13 Solution
// Circle1.java
// Program draws a circle of a random
// diameter and displays the area, diameter,
// and circumference.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Circle1 extends JFrame {
   private CircleCanvas theCanvas;
   private JTextArea display;

   public Circle1()
   {
      super( "Circle1" );
      theCanvas = new CircleCanvas();
      display = new JTextArea( 5, 30 );

      display.setText( "The Radius is: " + theCanvas.getRadius() +
                       "\nThe Diameter is: " + theCanvas.getDiameter()
                       + "\nThe Area is: " + theCanvas.getArea() +
                       "\nThe Circumference is: " +
                       theCanvas.getCircumference() );

      getContentPane().add( theCanvas, BorderLayout.CENTER );
      getContentPane().add( display, BorderLayout.SOUTH );
      setSize( 200, 200 );
      show();
   }

   public static void main( String args[] )
   {
      Circle1 app = new Circle1();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}

class CircleCanvas extends JPanel {
   private int radius;

   public CircleCanvas()
   {
      radius = ( int )( 1 + Math.random() * 100 );
      setSize( 100, 100 );
   }

   public void paintComponent( Graphics g )
   {  g.drawOval( 0, 0, radius, radius );  }

   public int getDiameter()  {  return ( 2 * radius );  }

```

```
    public int getCircumference()
    {  return ( int )( 2 * Math.PI * radius );  }

    public int getArea()
    {  return ( int )( radius * radius * Math.PI );  }

    public int getRadius()  {  return radius;  }
}
```



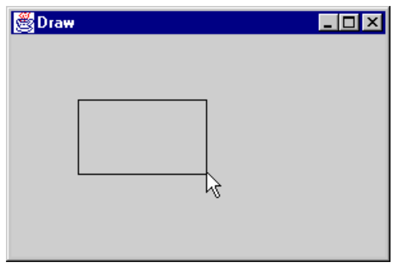
29.14 Write a program that uses `System.out.println` statements to print out events as they occur. Provide a `JComboBox` with a minimum of four items. The user should be able to choose an event to "monitor" from the `JComboBox`. When that particular event occurs, display information about the event in a message dialog box. Use method `toString` on the event object to convert it to a string representation.

29.15 Write a program using methods from interface `MouseListener` that allows the user to press the mouse button, drag the mouse and release the mouse button. When the mouse is released, draw a rectangle with the appropriate upper-left corner, width and height. (*Hint*: The `mousePressed` method should capture the set of coordinates at which the user presses and holds the mouse button initially, and the `mouseReleased` method should capture the set of coordinates at which the user releases the mouse button. Both methods should store the appropriate coordinate values. All drawing should be done on a subclass of `JPanel` and all calculations of the width, height and upper-left corner should be performed by the `paintComponent` method before the shape is drawn).

ANS:

```java
// Exercise 29.15 Solution
// Draw.java
// Program draws a rectangle with the mouse
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw extends JFrame {
   private int topX, topY;
   private int width, height, upperX, upperY;
   private int bottomX, bottomY;

   public Draw()
   {
      super( "Draw" );
      addMouseListener( new MouseHandler() );
      setSize( 300, 200 );
      show();
   }

   public void setTopX( int x ) { topX = x; }
   public void setTopY( int y ) { topY = y; }
   public void setBottomX( int x ) { bottomX = x; }
   public void setBottomY( int y ) { bottomY = y; }

   public void paint( Graphics g )
   {
      super.paint( g );

      width = Math.abs( topX - bottomX );
      height = Math.abs( topY - bottomY );
      upperX = Math.min( topX, bottomX );
      upperY = Math.min( topY, bottomY );

      g.drawRect( upperX, upperY, width, height );
   }

   public static void main( String args[] )
   {
      Draw app = new Draw();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   private class MouseHandler extends MouseAdapter {
      public void mouseReleased( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }
```

```

      public void mousePressed( MouseEvent e )
      {
         setTopX( e.getX() );
         setTopY( e.getY() );
      }
   }
}
```

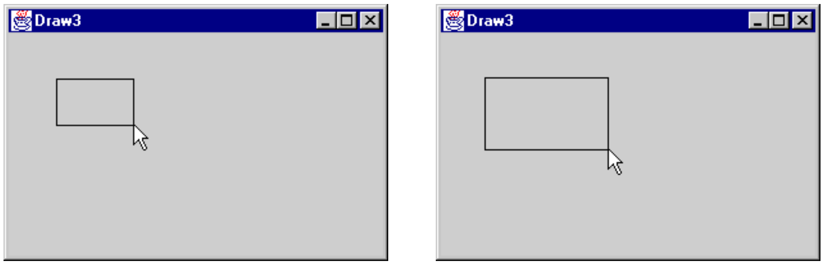


29.16 Modify Exercise 29.15 to provided a "rubber-banding" effect. As the user drags the mouse, the user should be able to see the current size of the rectangle to know exactly what the rectangle will look like when the mouse button is released. (*Hint*: Method `mouseDragged` should perform the same tasks as `mouseReleased`).

ANS:

```
// Exercise 29.16 Solution
// Draw3.java
// Program draws a rectangle with the mouse
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw3 extends JFrame {
   private int topX, topY;
   private int width, height, upperX, upperY;
   private int bottomX, bottomY;

   public Draw3()
   {
      super( "Draw3" );
      addMouseListener( new MouseHandler() );
      addMouseMotionListener( new MouseMotionHandler() );
      setSize( 300, 200 );
      show();
   }

   public void setTopX( int x ) { topX = x; }
   public void setTopY( int y ) { topY = y; }
   public void setBottomX( int x ) { bottomX = x; }
   public void setBottomY( int y ) { bottomY = y; }

   public void paint( Graphics g )
   {
      super.paint( g );

      width = Math.abs( topX - bottomX );
      height = Math.abs( topY - bottomY );
      upperX = Math.min( topX, bottomX );
      upperY = Math.min( topY, bottomY );

      g.drawRect( upperX, upperY, width, height );
   }

   public static void main( String args[] )
   {
      Draw3 app = new Draw3();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   private class MouseHandler extends MouseAdapter {
      public void mouseReleased( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }

      public void mousePressed( MouseEvent e )
      {
```

```
         setTopX( e.getX() );
         setTopY( e.getY() );
      }
   }

   private class MouseMotionHandler extends MouseMotionAdapter {
      public void mouseDragged( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }
   }
}
```



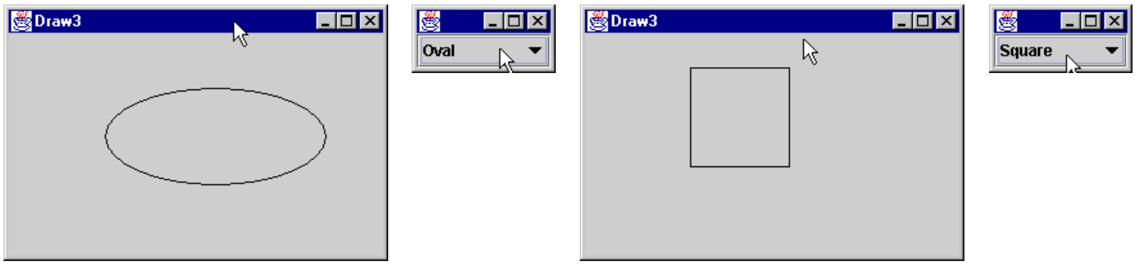
29.17 Modify Exercise 29.16 to allow the user to select which shape to draw. A `JComboBox` should provide options including at least rectangle, oval, line and rounded rectangle.

ANS:

```java
// Exercise 29.17 Solution
// Draw3.java
// Program draws a rectangle with the mouse
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw3 extends JFrame {
   private int topX, topY;
   private int width, height, upperX, upperY;
   private int bottomX, bottomY;
   private final int CIRCLE = 0;
   private final int SQUARE = 1;
   private final int OVAL = 2;
   private final int RECTANGLE = 3;
   private JComboBox choice;
   private int shape;
   private ToolWindow tools;

   public Draw3()
   {
      super( "Draw3" );
      addMouseListener( new MouseHandler() );
      addMouseMotionListener( new MouseMotionHandler() );

      // set default shape to Circle
      shape = CIRCLE;

      setSize( 300, 200 );
      show();
      tools = new ToolWindow();
   }

   public void setTopX( int x ) { topX = x; }
   public void setTopY( int y ) { topY = y; }
   public void setBottomX( int x ) { bottomX = x; }
   public void setBottomY( int y ) { bottomY = y; }
   public void setShape( int s ) { shape = s; }

   public void paint( Graphics g )
   {
      super.paint( g );

      width = Math.abs( topX - bottomX );
      height = Math.abs( topY - bottomY );
      upperX = Math.min( topX, bottomX );
      upperY = Math.min( topY, bottomY );

      switch ( shape ) {
         case CIRCLE:
            g.drawOval( upperX, upperY, width, width );
            break;
         case SQUARE:
            g.drawRect( upperX, upperY, width, width );
            break;
         case OVAL:
            g.drawOval( upperX, upperY, width, height );
            break;
         case RECTANGLE:
            g.drawRect( upperX, upperY, width, height );
            break;
      }
   }
```

```
   public static void main( String args[] )
   {
      Draw3 app = new Draw3();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   private class ToolWindow extends JFrame {
      public ToolWindow()
      {
         choice = new JComboBox();
         choice.addItem( "Circle" );
         choice.addItem( "Square" );
         choice.addItem( "Oval" );
         choice.addItem( "Rectangle" );

         choice.addItemListener(
            new ItemListener() {
               public void itemStateChanged( ItemEvent e )
               {
                  setShape( choice.getSelectedIndex() );
                  repaint();
               }
            }
         );

         // set default shape to Circle
         shape = CIRCLE;

         Container c = getContentPane();
         c.setLayout( new BorderLayout() );
         c.add( choice, BorderLayout.SOUTH );

         pack();
         setLocation( 300, 0 );
         show();
      }
   }

   private class MouseHandler extends MouseAdapter {
      public void mouseReleased( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }

      public void mousePressed( MouseEvent e )
      {
         setTopX( e.getX() );
         setTopY( e.getY() );
      }
   }

   private class MouseMotionHandler extends MouseMotionAdapter {
      public void mouseDragged( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }
```

```
    }
}
```

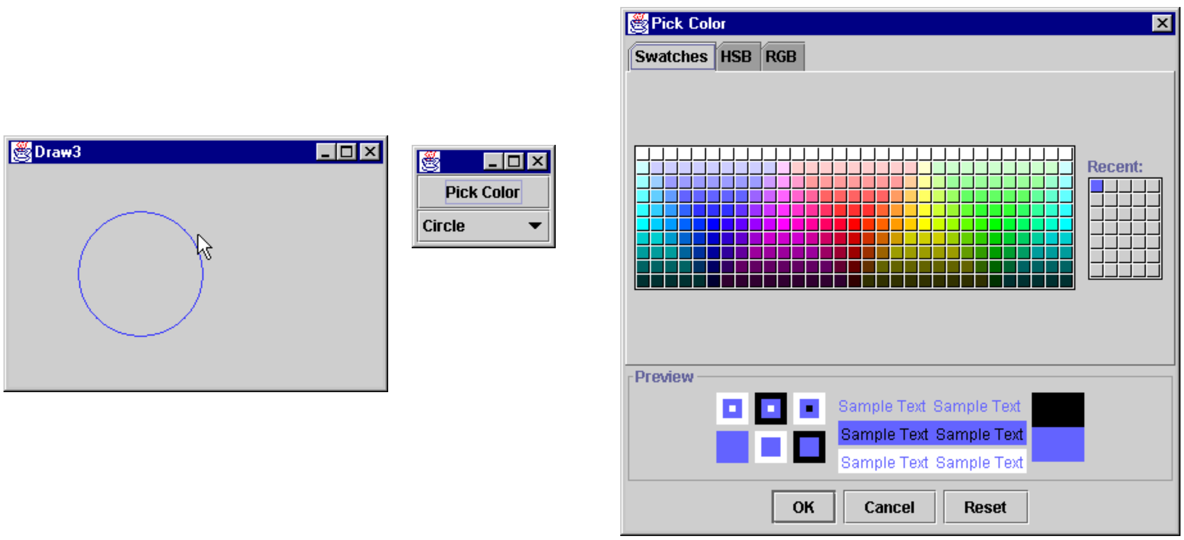

Draw3
Oval
Draw3
Square

29.18 Modify Exercise 29.17 to allow the user to select the drawing color from a `JColorChooser` dialog box.
ANS:

```
// Exercise 29.18 Solution
// Draw3.java
// Program draws a rectangle with the mouse
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw3 extends JFrame {
   private int topX, topY;
   private int width, height, upperX, upperY;
   private int bottomX, bottomY;
   private final int CIRCLE = 0;
   private final int SQUARE = 1;
   private final int OVAL = 2;
   private final int RECTANGLE = 3;
   private JComboBox choice;
   private int shape;
   private ToolWindow tools;
   private JButton chooseColor;
   private Color color;

   public Draw3()
   {
      super( "Draw3" );
      addMouseListener( new MouseHandler() );
      addMouseMotionListener( new MouseMotionHandler() );

      // set default shape to Circle
      shape = CIRCLE;

      setSize( 300, 200 );
      show();
      tools = new ToolWindow();
   }

   public void setTopX( int x ) { topX = x; }
   public void setTopY( int y ) { topY = y; }
   public void setBottomX( int x ) { bottomX = x; }
   public void setBottomY( int y ) { bottomY = y; }
   public void setShape( int s ) { shape = s; }

   public void paint( Graphics g )
   {
      super.paint( g );

      g.setColor( color );

      width = Math.abs( topX - bottomX );
      height = Math.abs( topY - bottomY );
      upperX = Math.min( topX, bottomX );
      upperY = Math.min( topY, bottomY );

      switch ( shape ) {
         case CIRCLE:
            g.drawOval( upperX, upperY, width, width );
            break;
         case SQUARE:
            g.drawRect( upperX, upperY, width, width );
            break;
         case OVAL:
            g.drawOval( upperX, upperY, width, height );
            break;
         case RECTANGLE:
            g.drawRect( upperX, upperY, width, height );
            break;
```

```
      }
   }

   public static void main( String args[] )
   {
      Draw3 app = new Draw3();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   private class ToolWindow extends JFrame {
      public ToolWindow()
      {
         choice = new JComboBox();
         choice.addItem( "Circle" );
         choice.addItem( "Square" );
         choice.addItem( "Oval" );
         choice.addItem( "Rectangle" );

         choice.addItemListener(
            new ItemListener() {
               public void itemStateChanged( ItemEvent e )
               {
                  setShape( choice.getSelectedIndex() );
                  repaint();
               }
            }
         );

         Container c = getContentPane();
         c.add( choice, BorderLayout.SOUTH );

         chooseColor = new JButton( "Pick Color" );
         chooseColor.addActionListener(
            new ActionListener() {
               public void actionPerformed( ActionEvent e )
               {
                  color = JColorChooser.showDialog( null, "Pick Color", Color.black );
               }
            }
         );
         c.add( chooseColor, BorderLayout.NORTH );

         pack();
         setLocation( 300, 0 );
         show();
      }
   }

   private class MouseHandler extends MouseAdapter {
      public void mouseReleased( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }

      public void mousePressed( MouseEvent e )
      {
         setTopX( e.getX() );
         setTopY( e.getY() );
      }
```

```
    }

    private class MouseMotionHandler extends MouseMotionAdapter {
       public void mouseDragged( MouseEvent e )
       {
          setBottomX( e.getX() );
          setBottomY( e.getY() );
          repaint();
       }
    }
 }
```



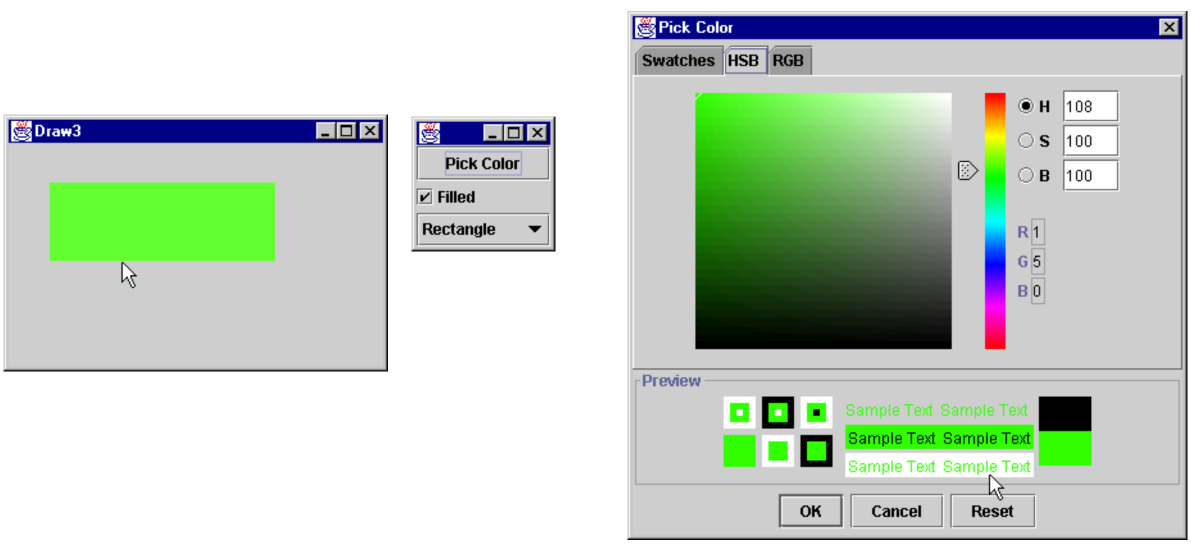
29.19 Modify Exercise 29.18 to allow the user to specify if a shape should be filled or empty when it is drawn. The user should click a `JCheckBox` to indicate filled or empty.

ANS:

```java
// Exercise 29.19 Solution
// Draw3.java
// Program draws a rectangle with the mouse
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Draw3 extends JFrame {
   private int topX, topY;
   private int width, height, upperX, upperY;
   private int bottomX, bottomY;
   private final int CIRCLE = 0;
   private final int SQUARE = 1;
   private final int OVAL = 2;
   private final int RECTANGLE = 3;
   private JComboBox choice;
   private int shape;
   private ToolWindow tools;
   private JButton chooseColor;
   private Color color;
   private JCheckBox filled;

   public Draw3()
   {
      super( "Draw3" );
      addMouseListener( new MouseHandler() );
      addMouseMotionListener( new MouseMotionHandler() );

      // set default shape to Circle
      shape = CIRCLE;

      tools = new ToolWindow();
      setSize( 300, 200 );
      show();
   }

   public void setTopX( int x ) { topX = x; }
   public void setTopY( int y ) { topY = y; }
   public void setBottomX( int x ) { bottomX = x; }
   public void setBottomY( int y ) { bottomY = y; }
   public void setShape( int s ) { shape = s; }

   public void paint( Graphics g )
   {
      super.paint( g );

      g.setColor( color );

      width = Math.abs( topX - bottomX );
      height = Math.abs( topY - bottomY );
      upperX = Math.min( topX, bottomX );
      upperY = Math.min( topY, bottomY );

      if ( filled.isSelected() )
         switch ( shape ) {
            case CIRCLE:
               g.fillOval( upperX, upperY, width, width );
               break;
            case SQUARE:
               g.fillRect( upperX, upperY, width, width );
               break;
            case OVAL:
               g.fillOval( upperX, upperY, width, height );
               break;
```

```
            case RECTANGLE:
               g.fillRect( upperX, upperY, width, height );
               break;
         }
      else
         switch ( shape ) {
            case CIRCLE:
               g.drawOval( upperX, upperY, width, width );
               break;
            case SQUARE:
               g.drawRect( upperX, upperY, width, width );
               break;
            case OVAL:
               g.drawOval( upperX, upperY, width, height );
               break;
            case RECTANGLE:
               g.drawRect( upperX, upperY, width, height );
               break;
         }
   }

   public static void main( String args[] )
   {
      Draw3 app = new Draw3();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }

   private class ToolWindow extends JFrame {
      public ToolWindow()
      {
         choice = new JComboBox();
         choice.addItem( "Circle" );
         choice.addItem( "Square" );
         choice.addItem( "Oval" );
         choice.addItem( "Rectangle" );

         choice.addItemListener(
            new ItemListener() {
               public void itemStateChanged( ItemEvent e )
               {
                  setShape( choice.getSelectedIndex() );
                  repaint();
               }
            }
         );

         Container c = getContentPane();
         c.add( choice, BorderLayout.SOUTH );

         chooseColor = new JButton( "Pick Color" );
         chooseColor.addActionListener(
            new ActionListener() {
               public void actionPerformed( ActionEvent e )
               {
                  color = JColorChooser.showDialog( null, "Pick Color", Color.black );
               }
            }
         );
         c.add( chooseColor, BorderLayout.NORTH );

```

```
         filled = new JCheckBox( "Filled" );
         c.add( filled, BorderLayout.CENTER );

         pack();
         setLocation( 300, 0 );
         show();
      }
   }

   private class MouseHandler extends MouseAdapter {
      public void mouseReleased( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }

      public void mousePressed( MouseEvent e )
      {
         setTopX( e.getX() );
         setTopY( e.getY() );
      }
   }

   private class MouseMotionHandler extends MouseMotionAdapter {
      public void mouseDragged( MouseEvent e )
      {
         setBottomX( e.getX() );
         setBottomY( e.getY() );
         repaint();
      }
   }
}
```



29.20 *(Complete Drawing Application)* Using the techniques developed in Exercises 29.12 through 29.19, create a complete drawing program. The program should use the GUI components of this chapter to enable the user to select the shape, color and fill characteristics. For this program, create your own classes (like those in the class hierarchy described in Exercise 27.19) from which objects will be created to store each shape the user draws. The classes should store the location, dimensions and color of each shape and should indicate if the shape is filled or unfilled. Your classes should all derive from a class called `MyShape` that has all the common features of every shape type. Every subclass of `MyShape` should have its own method `draw`, which returns `void` and receives a `Graphics` object as its argument. Create a subclass of `JPanel` called `DrawPanel` for drawing the shapes. When the `DrawPanel`'s `paintComponent` method is called, it should walk through the array of shapes and display each shape by polymorphically calling the shape's `draw` method (passing the `Graphics` object as an argument). Each shape's `draw` method should know how to draw the shape. As a minimum, your program should provide the following classes: `MyLine`, `MyOval`, `MyRect`, `MyRoundRect`. Design the class hierarchy for maximum software reuse and place all your classes in the package `shapes`. Import this package into your program. Each shape should be stored in an array of `MyShape` objects, where `MyShape` is the superclass in your hierarchy of shape classes (see Exercise 27.19).

29.21 Modify Exercise 29.20 to provide an `Undo` button that can be used repeatedly to undo the last painting operation. If there are no shapes in the array of shapes, the `Undo` button should be disabled.

30

Java Multimedia: Images, Animation, and Audio: Solutions

EXERCISES

30.3 Describe how to make an animation “browser friendly.”

ANS: Begin the animation in the `start` method and suspend/terminate the animation in the `stop` method.

30.4 Discuss the various aspects of flicker elimination in Java.

ANS: Flickering can be reduced or eliminated by overriding the paint method and using graphics double buffering techniques.

30.5 Explain the technique of graphics double buffering.

ANS: Method `createImage` is used to create an empty image. The graphics context of the empty image is retrieved with a call to `getGraphics`. The empty image can then be used to store pixels drawn on the image with the `Graphics` object that was obtained via `getGraphics`. When the image is complete, it can be displayed using method `drawImage`. Swing components such as `JPanel` have built-in double buffering.

30.6 Describe the Java methods for playing and manipulating audio clips.

ANS: The applet `play` method. The `AudioClip` interface methods: `play`, `loop` and `stop`.

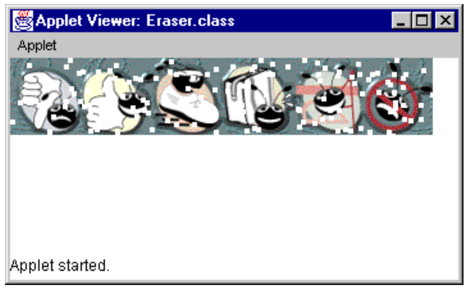
30.7 *(Animation)* Create a a general-purpose Java animation program. Your program should allow the user to specify the sequence of frames to be displayed, the speed at which the images are displayed, audios that should be played while the animation is running and so on.

30.8 *(Screensaver)* Use animation of a series of your favorite images to create a screensaver program. Create various special effects that explode the images, spin the images, fade them in and out, move them off the edge of the screen, and the like.

30.9 *(Randomly Erasing an Image)* Suppose an image is displayed in a rectangular screen area. One way to erase the image is simply to set every pixel to the same color immediately, but this is a dull visual effect. Write a Java program that displays an image then erases it by using random-number generation to select individual pixels to erase. After most of the image is erased, erase all of the remaining pixels at once. You can refer to individual pixels by having a line that starts and ends at the same point. You might try several variants of this problem. For example, you might display lines randomly or you might display shapes randomly to erase regions of the screen.

ANS:

```
// Exercise 30.9 Solution
// Eraser.java
// Program randomly covers up an image.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Eraser extends JApplet implements ActionListener {
   private ImageIcon image;
   private int imageWidth, imageHeight, count;
   private int numberOfTimes;
   private boolean showImage = true;
   private Timer t;

   public void init()
   {
      image = new ImageIcon( "icons2.gif" );
      t = new Timer( 10, this );
      t.start();

      imageWidth = image.getIconWidth();
      imageHeight = image.getIconHeight();
      numberOfTimes = imageWidth * imageHeight / 8;
   }

   public void paint( Graphics g )
   {
      if ( showImage == true ) {
         image.paintIcon( this, getGraphics(), 0, 0 );
         showImage = false;
      }

      g.setColor( getBackground() );
      g.fillRect( ( int ) ( Math.random() * imageWidth ),
                  ( int ) ( Math.random() * imageHeight ), 4, 4 );
   }

   public void actionPerformed( ActionEvent e )
   {
      repaint();

      if ( count == numberOfTimes ) {
         showImage = true;
         count = 0;
      }

      ++count;
   }
}
```

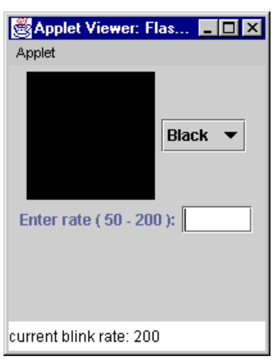

Applet Viewer: Eraser.class
Applet
Applet started.

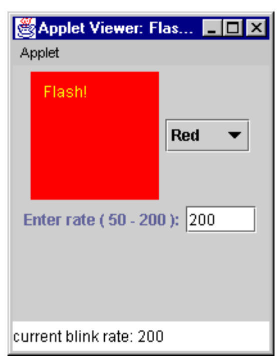
30.10 *(Text Flasher)* Create a Java program that repeatedly flashes text on the screen. Do this by interspersing the text with a plain background color image. Allow the user to control the “blink speed” and the background color or pattern.

ANS:

```
// Exercise 30.10 Solution
// Flash.java
// Program flashes text.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Flash extends JApplet
          implements ActionListener, ItemListener {
   private MyCanvas theCanvas;
   private JComboBox colorSelect;
   private JLabel prompt;
   private JTextField input;

   public void init()
   {
      prompt = new JLabel( "Enter rate ( 50 - 200 ):" );
      input = new JTextField( 5 );
      input.addActionListener( this );
      theCanvas = new MyCanvas();
      String items[] = { "Black", "Red", "Blue", "Green" };
      colorSelect = new JComboBox( items );
      colorSelect.addItemListener( this );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( theCanvas );
      c.add( colorSelect );
      c.add( prompt );
      c.add( input );
   }

   public void itemStateChanged( ItemEvent e )
   {
      Color c;

      if ( e.getItem().equals( "Black" ) )
         c = Color.black;
      else if ( e.getItem().equals( "Red" ) )
         c = Color.red;
      else if ( e.getItem().equals( "Blue" ) )
         c = Color.blue;
      else
         c = Color.green;

      theCanvas.setBackground( c );
   }

   public void actionPerformed( ActionEvent e )
   {
      theCanvas.setSleepTime( Integer.parseInt( input.getText() ) );
      showStatus( "current blink rate: " + theCanvas.getSleepTime() );
   }
}

class MyCanvas extends JPanel implements ActionListener {
   private String text;
   private Timer t;
```

```
   private Color c = Color.black;
   boolean flash = true;

   public MyCanvas()
   {
      setBackground( Color.black );
      t = new Timer( 150, this );
      t.start();
      text = "Flash!";
      setSize( 100, 100 );
      setOpaque( true );
   }

   public synchronized void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      if ( flash ) {
         g.setColor( Color.yellow );
         g.drawString( text, 10, 20 );
      }
   }

   public synchronized void actionPerformed( ActionEvent e )
   {
      flash = !flash;
      repaint();
   }

   public void setSleepTime( int time )
      { t.setDelay( time >= 50 && time <= 200 ? time : 150 ); }

   public int getSleepTime() { return t.getDelay(); }

   public Dimension getPreferredSize()
   {
      return new Dimension( 100, 100 );
   }
}
```



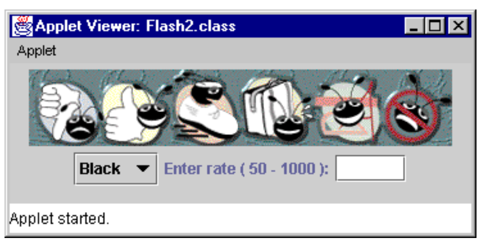


30.11 *(Image Flasher)* Create a Java program that repeatedly flashes an image on the screen. Do this by interspersing the image with a plain background color image.

ANS:

```
// Exercise 30.11 Solution
// Flash2.java
// Program flashes text.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Flash2 extends JApplet
          implements ActionListener, ItemListener {
   private MyCanvas theCanvas;
   private JComboBox colorSelect;
   private JLabel prompt;
   private JTextField input;

   public void init()
   {
      prompt = new JLabel( "Enter rate ( 50 - 1000 ):" );
      input = new JTextField( 5 );
      input.addActionListener( this );
      theCanvas = new MyCanvas();
      String items[] = { "Black", "Red", "Blue", "Green" };
      colorSelect = new JComboBox( items );
      colorSelect.addItemListener( this );
      Container c = getContentPane();
      c.setLayout( new FlowLayout() );
      c.add( theCanvas );
      c.add( colorSelect );
      c.add( prompt );
      c.add( input );
   }

   public void itemStateChanged( ItemEvent e )
   {
      Color c;

      if ( e.getItem().equals( "Black" ) )
         c = Color.black;
      else if ( e.getItem().equals( "Red" ) )
         c = Color.red;
      else if ( e.getItem().equals( "Blue" ) )
         c = Color.blue;
      else
         c = Color.green;

      theCanvas.setBackground( c );
   }

   public void actionPerformed( ActionEvent e )
   {
      theCanvas.setSleepTime( Integer.parseInt( input.getText() ) );
      showStatus( "current blink rate: " + theCanvas.getSleepTime() );
   }
}

class MyCanvas extends JPanel implements ActionListener {
   private ImageIcon image;
   private Timer t;
   boolean flash = true;

   public MyCanvas()
   {
      setBackground( Color.black );
      image = new ImageIcon( "icons2.gif" );
```

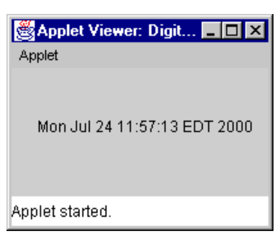
```
      t = new Timer( 500, this );
      t.start();
   }

   public synchronized void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      if ( flash )
         g.drawImage( image.getImage(), 0, 0, this );
   }

   public synchronized void actionPerformed( ActionEvent e )
   {
      flash = !flash;
      repaint();
   }

   public void setSleepTime( int time )
      { t.setDelay( time >= 50 && time <= 1000 ? time : 500 ); }

   public int getSleepTime() { return t.getDelay(); }

   public Dimension getPreferredSize()
   {
      return new Dimension( image.getIconWidth(),
                            image.getIconHeight() );
   }
}
```



30.12 *(Digital Clock)* Implement a program that displays a digital clock on the screen. You might add options to scale the clock; display day, month and year; issue an alarm; play certain audios at designated times and the like.

ANS:

```
// Exercise 30.12 Solution
// DigitalClock.java
// Program creates a digital clock.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class DigitalClock extends JApplet
                          implements ActionListener {
   private String theTime;
   private Timer t;

   public void init()
   {
      theTime = "";
      t = new Timer( 1000, this );
   }

   public void paint( Graphics g )
   {
      super.paint( g );  // clears the background

      g.drawString( theTime, 20, 50 );
   }

   public void start()
   {
      t.start();
   }

   public void stop()
   {
      t.stop();
   }

   public void actionPerformed( ActionEvent e )
   {
      theTime = new Date().toString();
      repaint();
   }
}
```

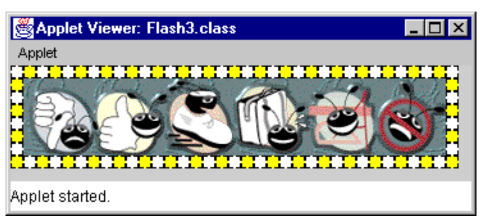


30.13 (*Calling attention to an image*) If you want to emphasize an image, you might place a row of simulated light bulbs around your image. You can let the light bulbs flash in unison or you can let them fire on and off in sequence one after the other.

ANS:

```
// Exercise 30.13 Solution
// Flash3.java
// Program highlights an image.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Flash3 extends JApplet {
   private MyCanvas theCanvas;

   public void init()
   {
      ImageIcon image1 = new ImageIcon( "icons2.gif" );

      int width = image1.getIconWidth() + 20;
      int height = image1.getIconHeight() + 20;
      Image image2 = createImage( width, height );
      Image image3 = createImage( width, height );

      theCanvas = new MyCanvas( image1.getImage(), image2, image3,
                                width, height );
      getContentPane().add( theCanvas, BorderLayout.CENTER );
   }
}

class MyCanvas extends JPanel implements ActionListener {
   private Image img, img2, img3;
   private Graphics graph2, graph3;
   private boolean flashSwitch;
   private Timer t;

   public MyCanvas( Image i, Image i2, Image i3, int w, int h )
   {
      t = new Timer( 300, this );
      t.start();
      flashSwitch = true;
      setSize( w, h );
      img = i;
      img2 = i2;
      img3 = i3;
      createBuffers( w, h );
   }

   public void createBuffers( int w, int h )
   {
      graph2 = img2.getGraphics();
      graph3 = img3.getGraphics();

      graph2.setColor( Color.black );
      graph2.fillRect( 0, 0, w, h );
      graph3.setColor( Color.black );
      graph3.fillRect( 0, 0, w, h );

      int count = 0;

      for ( int x = 0; x < w; x += 10 ) {
         for ( int y = 0; y < h; y += 10 ) {

            // Change ++count to y to get the effect of
            // all the lights "turning off" then "turning on"
            // Also the line below that alternates the lights
            // should be commented out or removed.
```

```java
               if ( ++count % 2 == 0 ) {
                  graph2.setColor( Color.yellow );
                  graph3.setColor( Color.white );
               }
               else {
                  graph2.setColor( Color.white );
                  graph3.setColor( Color.yellow );
               }

               graph2.fillOval( x, y, 10, 10 );
               graph3.fillOval( x, y, 10, 10 );
            }

            // Allow the lights to alternate
            count = ( count % 2 == 0 ? 1 : 0 );
         }

         graph2.drawImage( img, 10, 10, this );
         graph3.drawImage( img, 10, 10, this );
      }

      public void paintComponent( Graphics g )
      {
         super.paintComponent( g );

         if ( flashSwitch )
            g.drawImage( img2, 0, 0, this );
         else
            g.drawImage( img3, 0, 0, this );
      }

      public void actionPerformed( ActionEvent e )
      {
         flashSwitch = !flashSwitch;
         repaint();
      }
   }
```



30.14 *Image Zooming)* Create a program that enables you to zoom in on, or away from, an image.

ANS:

```java
// Exercise 30.14 Solution
// Zoom.java
// Program zooms an image.
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Zoom extends JApplet
         implements ItemListener {
   private MyCanvas theCanvas;
   private JPanel p;
   private JComboBox select;
   private int width, height;

   public void init()
   {
      ImageIcon image1 = new ImageIcon( "icons2.gif" );

      p = new JPanel();
      String items[] = { "50%", "100%", "200%", "300%" };
      select = new JComboBox( items );
      select.addItemListener( this );
      p.add( select );

      width = image1.getIconWidth() / 2;
      height = image1.getIconHeight() / 2;

      theCanvas = new MyCanvas( image1.getImage(), width, height );
      Container c = getContentPane();
      c.add( theCanvas, BorderLayout.CENTER );
      c.add( p, BorderLayout.SOUTH );
   }

   public void itemStateChanged( ItemEvent e )
   {
      if ( e.getItem().equals( "50%" ) )
         theCanvas.setWidthHeight( ( int ) ( width * .5 ), ( int ) ( height * .5 ) );
      else if ( e.getItem().equals( "100%" ) )
         theCanvas.setWidthHeight( width, height );
      else if ( e.getItem().equals( "200%" ) )
         theCanvas.setWidthHeight( width * 2, height * 2 );
      else  // 300%
         theCanvas.setWidthHeight( width * 3, height * 3 );
   }
}

class MyCanvas extends JPanel {
   private Image img;
   private int imgWidth, imgHeight;

   public MyCanvas( Image i, int w, int h )
   {
      setBackground( Color.green );
      setSize( w, h );
      img = i;
      setWidthHeight( w, h );
   }

   public void setWidthHeight( int w, int h )
   {
      imgWidth = w;
      imgHeight = h;
      repaint();
   }
```

```

    public void paintComponent( Graphics g )
    {
       super.paintComponent( g );
       g.drawImage( img, 0, 0, imgWidth, imgHeight, this );
    }
}
```