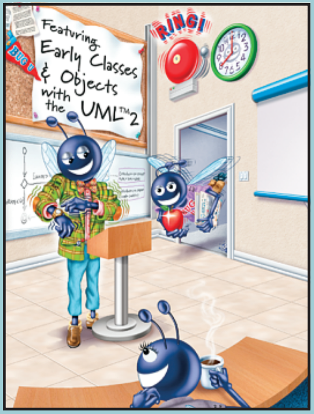
8



Pointers and Pointer-Based Strings

OBJECTIVES

In this chapter you will learn:

- What pointers are.
- The similarities and differences between pointers and references and when to use each.
- To use pointers to pass arguments to functions by reference.
- To use pointer-based C-style strings.
- The close relationships among pointers, arrays and C-style strings.
- To use pointers to functions.
- To declare and use arrays of C-style strings.

Assignment Checklist

Name: ____________________ Date: ____________________

Section: ____________________

Exercises	Assigned: Circle assignments	Date Due
Prelab Activities Matching	YES NO	
Fill in the Blank	11, 12, 13, 14, 15, 16, 17, 18	
Short Answer	19, 20, 21, 22	
Programming Output	23, 24, 25, 26, 27, 28, 29	
Correct the Code	30, 31, 32, 33, 34, 35	
Lab Exercises Lab Exercise 1 — String-Comparison Functions	YES NO	
Follow-Up Question and Activity	1	
Lab Exercise 2 — Shuffling and Dealing	YES NO	
Follow-Up Question and Activity	1	
Debugging	YES NO	
Labs Provided by Instructor 1.		
2.		
3.		
Postlab Activities Coding Exercises	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	
Programming Challenges	1, 2, 3, 4	

Prelab Activities

Matching

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

After reading Chapter 8 of *C++ How to Program: Fifth Edition*, answer the given questions. These questions are intended to test and reinforce your understanding of key concepts and may be done either before the lab or during the lab.

For each term in the column on the left, write the corresponding letter for the description that best matches it from the column on the right.

.

	Term	Description
___	1. `&`	a) `ptr[ 3 ]` (`ptr` is a pointer).
___	2. *rvalue*	b) A character in single quotes that corresponds to an integer value.
___	3. Pointer-offset notation	c) Can be used to determine the size of an array in bytes.
___	4. Dereferencing operator	d) A series of characters enclosed in double quotation marks.
___	5. lvalue	e) `ptr + 3` (`ptr` is a pointer).
___	6. Pointer-subscript notation	f) Series of characters delimited by characters such as spaces or punctuation marks.
___	7. `sizeof` operator	g) Can be used on the left side of an assignment operator (e.g., variables).
___	8. Character constant	h) *.
___	9. String literal	i) Unary operator that returns the address of its operand.
___	10. Token	j) Can be used on the right side of an assignment operator (e.g., constants) only.

Prelab Activities

Fill in the Blank

Name: ___________________________ **Date:** ___________________________

Section: ___________________________

Fill in the blanks in each of the following statements:

11. Pointers are variables that contain other variables' __________.

12. All elements of a(n) __________ are stored contiguously in memory.

13. In many cases, pointers can be accessed exactly like arrays (known as pointer __________ notation).

14. The only integer value that can be assigned to a pointer without casting is __________.

15. It is not necessary to include names of pointers in function prototypes; it is necessary only to include the pointer __________.

16. The * operator is referred to as the __________, or __________, operator.

17. Subtracting or comparing two pointers that do not point to elements of the same __________ is usually a logic error.

18. Function __________ copies its second argument—a string—into its first argument—a character array.

Prelab Activities

Name:

Short Answer

Name: ____________________ **Date:** ____________________

Section: ____________________

In the space provided, answer each of the given questions. Your answers should be as concise as possible; aim for two or three sentences.

19. What are the three kinds of things that can be assigned to a pointer?

20. What happens when a programmer tries to dereference a pointer that has not been properly initialized or has not been assigned to point to a specific location in memory? What sort of error(s) will this dereferencing cause?

21. What is pointer arithmetic? Why is it applicable to arrays only?

Prelab Activities

Name:

Short Answer

22. An array name is like a constant pointer to the beginning of an array. Can programmers modify array names in the same way as they modify pointers (i.e., in arithmetic expressions)? What about operators ++, --. +=, -=, and =? Can these be used with an array name?

Prelab Activities

Name:

Programming Output

Name: ______________________________ **Date:** ______________________________

Section: ______________________________

For each of the given program segments, read the code and write the output in the space provided below each program. [*Note:* Do not execute these programs on a computer.]

23. What is output by the following program segment?

```
int number = 99;
int *ptr = &number;  // address of number is 0012FF7C

cout << number << " " << *ptr << " " << ptr;
```

Your answer:

24. What is output by the following code?

```
char c[] = "Hello you";
char *sPtr = c;

for ( ; *sPtr != 'u'; sPtr++ )
   cout << *sPtr;
```

Your answer:

Prelab Activities

Name:

Programming Output

25. What is output by the following program segment?

```
int a[] = { 1, 2, 3, 4, 5 };
int *ptr = a;

cout << a[ 3 ] << " " << *( ptr + 3 ) << " " << ptr[ 3 ];
```

Your answer:

26. What is output by the following program segment?

```
int main()
{
   char s1[] = "Boston, Massachusetts";
   char *sPtr2 = "New York, NY";

   cout << s1 << setw( 4 ) << sPtr2;

   for ( int i = 0; ( s1[ i ] = sPtr2[ i ] ) != '\0'; i++ )
      ;

   cout << endl << s1 << setw( 4 ) << sPtr2 << endl;

   return 0;

} // end main
```

Your answer:

Prelab Activities

Name:

Programming Output

27. What is output by the following program segment?

```
char x[] = "It is a nice day";
char y[ 25 ];
char z[ 15 ];

cout << "The string in array x is: " << x
     << "\nThe string in array y is: " << strcpy( y, x )
     << '\n';

strncpy( z, x, 9 );
z[ 9 ] = '\0';
cout << "The string in array z is: " << z << endl;
```

Your answer::

28. What is output by the following program segment?

```
char *s1Ptr = "C++ is fun";
char *s2Ptr = "C++ is fun everyday";

if ( strncmp( s1Ptr, s2Ptr, 10 ) == 0 )
   cout << "Equal ";
else
   cout << "Not Equal";
```

Your answer:

Prelab Activities

Name:

Programming Output

29. What is output by the following program segment?

```
char *sPtr = "Boston, MA";

cout << strlen( sPtr );
```

Your answer::

Prelab Activities

Name:

Correct the Code

Name: ______________________________ **Date:** ______________________________

Section: ______________________________

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic, syntax or compilation error, circle the error in the program, and write the corrected code in the space provided after each problem. If the code does not contain an error, write “no error.” [*Note:* It is possible that a program segment may contain multiple errors.]

30. The following declarations should declare three pointers:

```
int *ptr1;
int &ptr2;
int ptr2;
```

Your answer:

31. The following code should display a and the contents of a via the pointer:

```
int a = 7;
int *aptr = &a;

cout << *a << aptr
```

Your answer:

Prelab Activities

Name:

Correct the Code

32. The following code should print the elements of the array, using pointer arithmetic:

```
int a[ 5 ] = { 1, 2, 3, 4, 5 };
int *aptr = &a[ 0 ];

for ( int i = 0; i <= 5; i++ )
   cout << *( aptr + i );
```

Your answer:

33. The following program segment should copy part of `s1Ptr` into `s2`:

```
char *s1Ptr = "United States of America";
char s2[ 15 ];

for ( int i = 0; *( s1Ptr + i ) != 'o'; i++ )
   s2[ i ] = *( s1Ptr + i );

cout << s1Ptr << endl << s2;
```

Your answer:

Prelab Activities

Name:

Correct the Code

34. The following code should compare two strings and display whether they are equal:

```
char *s1Ptr = "Hello";
char *s2Ptr = "Hello";

if ( strcmp( s1Ptr, s2Ptr ) == true )
   cout << "Equal ";
else
   cout << "Not Equal";
```

Your answer:

35. The following code should display the result of concatenating two strings with `strcat`:

```
char s1[] = "How are you";
char s2[] = "Good night";

cout << strcat( s1, s2 );
```

Your answer:

Lab Exercises

Lab Exercise 1 — String-Comparison Functions

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 8.1)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up question. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 8 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Using pointer-subscript notation to access data stored in an array.
- Using pointer-offset notation to access data stored in an array.

The follow-up question and activity also will give you practice:

- Controlling pointer-subscript and pointer-offset notations.

Description of the Problem

Write two versions of the string-comparison function `strcmp`. The first version should use array subscripting, and the second should use pointers and pointer arithmetic.

Sample Output

```
Enter two strings: keychain keyboard
The value returned from stringCompare1( "keychain", "keyboard" ) is 1
The value returned from stringCompare2( "keychain", "keyboard" ) is 1
```

```
Enter two strings: keychain keyring
The value returned from stringCompare1( "keychain", "keyring" ) is -1
The value returned from stringCompare2( "keychain", "keyring" ) is -1
```

Lab Exercises

Name:

Lab Exercise 1 — String-Comparison Functions

```
Enter two strings: key key
The value returned from stringCompare1( "key", "key" ) is 0
The value returned from stringCompare2( "key", "key" ) is 0
```

Template

```
// Lab 1: stringCompare.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

/* Write a function prototype for stringCompare1 */
/* Write a function prototype for stringCompare2 */

int main()
{
   char string1[ 100 ], string2[ 100 ];

   cout << "Enter two strings: ";
   cin >> string1 >> string2;

   cout << "The value returned from stringCompare1( \"" << string1
      << "\", \"" << string2 << "\" ) is "
      << /* Write a function call for stringCompare1 */
      << "\nThe value returned from stringCompare2( \"" << string1
      << "\", \"" << string2 << "\" ) is "
      << /* Write a function call for stringCompare2 */ << '\n';

   return 0;
} // end main

/* Write a function header for stringCompare1 */
{
   int index;

   // array subscript notation
   /* Write a for loop that iterates through both strings
      using array subscript notation, until index contains the
      first index where the strings do not match or the index
      of the null character */
      ; // empty statement

   if ( s1[ index ] == '\0' && s2[ index ] == '\0' )
      return 0;
   /* Write code to return -1 or 1 based on which
      of s1[ index ] and s2[ index ] is greater */
} // end function stringCompare1

```

Fig. L 8.1 | `stringCompare.cpp` (Part 1 of 2.).

Lab Exercises

Name:

Lab Exercise 1 — String-Comparison Functions

```
/* Write a function header for stringCompare2 */
{
   // pointer notation
   /* Write a for loop that iterates through both strings
      using pointer notation, until both pointers point to
      the first character where the strings do not match
      or the index of the null character */
      ; // empty statement

   if ( *s1 == '\0' && *s2 == '\0' )
      return 0;
   /* Write code to return -1 or 1 based on which
      of *s1 and *s2 is greater */
} // end function stringCompare2
```

Fig. L 8.1 | `stringCompare.cpp` (Part 2 of 2.).

Problem-Solving Tips

1. Remember that pointer-subscript notation uses the pointer name and brackets, `[]`. Pointer-offset notation uses the pointer name and the dereferencing operator. Use each of these methods to step through the strings comparing individual characters.
2. Remember that your string comparison functions should return `-1` if the first string is less than the second, `0` if the two strings are equal and `1` if the first string is greater than the second.
3. Since comparing strings does not require modifying their contents, your string comparison functions should take their parameters as pointers to constant data.

Follow-Up Question and Activity

1. Modify your string compare functions to take a third argument specifying the number of characters in each string to compare. These functions should act like function `strncmp`. Again, one version should use array subscripting, and the other should use pointers and pointer arithmetic.

Lab Exercises

Name:

Lab Exercise 2 — Shuffling and Dealing

Name: __________________________ Date: __________________________

Section: __________________________

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 8.4)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working C++ program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the `/* */` comments with C++ code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up question. The source code for the template is available at `www.deitel.com` and `www.prenhall.com./deitel`.

Lab Objectives

This lab was designed to reinforce programming concepts from Chapter 8 of *C++ How To Program: Fifth Edition*. In this lab, you will practice:

- Using arrays of pointers to create an array of strings.
- Recognizing indefinite postponement and designing mechanisms to avoid it.
- Using `enum` to create an enumerated type `Suits`.

The follow-up question and activity also will give you practice:

- Eliminating unnecessary computation.

Description of the Problem

In the card shuffling and dealing program of Fig. 8.26, we intentionally used in an inefficient shuffling algorithm that introduced the possibility of indefinite postponement. In this problem, you will create a high-performance shuffling algorithm that avoids indefinite postponement.

Modify Fig. 8.26 as follows. Initialize the `deck` array as shown in Fig. L 8.2. 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. Display the resulting array to determine whether the deck is satisfactorily shuffled (as in Fig. L 8.3, for example).

Lab Exercises

Name:

Lab Exercise 2 — Shuffling and Dealing

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. L 8.2 | 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. L 8.3 | Sample shuffled deck array.

Sample Output

```
  Six of Clubs                  Deuce of Hearts
 King of Hearts                 Queen of Clubs
 King of Spades                  Nine of Hearts
  Ten of Hearts                   Six of Diamonds
 Jack of Hearts                  Five of Spades
Seven of Clubs                  Deuce of Clubs
 Jack of Diamonds                Four of Spades
  Ace of Clubs                    Ten of Clubs
 King of Diamonds                Nine of Diamonds
 Jack of Spades                 Three of Clubs
  Ten of Spades                   Ace of Spades
 Five of Clubs                  Seven of Spades
Eight of Spades                 Queen of Diamonds
Deuce of Spades                 Eight of Clubs
Seven of Hearts                   Six of Spades
Deuce of Diamonds               Eight of Hearts
 Four of Clubs                    Six of Hearts
 Four of Hearts                  Nine of Clubs
Three of Diamonds                Four of Diamonds
Three of Hearts                 Queen of Hearts
Three of Spades                   Ace of Diamonds
Eight of Diamonds                 Ace of Hearts
 Jack of Clubs                  Queen of Spades
Seven of Diamonds                 Ten of Diamonds
 Five of Hearts                  King of Clubs
 Nine of Spades                  Five of Diamonds
```

Lab Exercises

Name:

Lab Exercise 2 — Shuffling and Dealing

Template

```cpp
// Lab 2: DeckOfCards.cpp
// Member-function definitions for class DeckOfCards that simulates
// the shuffling and dealing of a deck of playing cards.
#include <iostream>
using std::cout;
using std::left;
using std::right;

#include <iomanip>
using std::setw;

#include <cstdlib> // prototypes for rand and srand
using std::rand;
using std::srand;

#include <ctime> // prototype for time
using std::time;

#include "DeckOfCards.h" // DeckOfCards class definition

// DeckOfCards default constructor initializes deck
DeckOfCards::DeckOfCards()
{
   // loop through rows of deck
   for ( int row = 0; row <= 3; row++ )
   {
      // loop through columns of deck for current row
      for ( int column = 0; column <= 12; column++ )
      {
         // initialize slot of deck
         /* Write code to initialize the slots in the deck to
            the numbers 1 through 52, inclusive */
      } // end inner for
   } // end outer for

   srand( time( 0 ) ); // seed random number generator
} // end DeckOfCards default constructor

// shuffle cards in deck
void DeckOfCards::shuffle()
{
   int row; // represents suit value of card
   int column; // represents face value of card

   // for each of the 52 slots, choose another slot of the deck randomly
   // loop through rows of deck
   /* Write a for header to loop through the rows of deck */
   {
      // loop through columns of deck for current row
      /* Write a for header to loop through the columns of deck */
      {
         row = rand() % 4; // randomly select another row
         column = rand() % 13; // randomly select another column

```

Fig. L 8.4 | `DeckOfCards.cpp`. (Part 1 of 2.)

Lab Exercises

Name:

Lab Exercise 2 — Shuffling and Dealing

```
         // swap the number in slot (r,c) with the number in slot (row,column)
         /* Write code to swap the number in deck[ r ][ c ] with the number
            in deck[ row ][ column ] */
      }

   } // end for
} // end function shuffle

// deal cards in deck
void DeckOfCards::deal()
{
   // initialize suit array
   static const char *suit[ 4 ] =
      { "Hearts", "Diamonds", "Clubs", "Spades" };

   // initialize face array
   static const char *face[ 13 ] =
      { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
        "Eight", "Nine", "Ten", "Jack", "Queen", "King" };

   // for each of the 52 cards
   for ( int card = 1; card <= 52; card++ )
   {
      // loop through rows of deck
      for ( int row = 0; row <= 3; row++ )
      {
         // loop through columns of deck for current row
         for ( int column = 0; column <= 12; column++ )
         {
            // if slot contains current card, display card
            if ( deck[ row ][ column ] == card )
            {
               cout << setw( 5 ) << right << face[ column ]
                  << " of " << setw( 8 ) << left << suit[ row ]
                  << ( card % 2 == 0 ? '\n' : '\t' );
            } // end if
         } // end innermost for
      } // end inner for
   } // end outer for
} // end function deal
```

Fig. L 8.4 | `DeckOfCards.cpp`. (Part 2 of 2.)

Problem-Solving Tips

1. Use counter variables `row` and `column` to calculate the numbers to fill the slots in two-dimensional array `deck`.
2. To swap the numbers associated with two cards, first store one number in a temporary storage variable, then replace the first number with the second number and finally place the temporary value into the second number.

Lab Exercises

Name:

Lab Exercise 2 — Shuffling and Dealing

Follow-Up Question and Activity

1. 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 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.

Debugging

Name: ____________________________ Date: ____________________________

Section: ____________________________

The program (Fig. L 8.5) in this section does not run properly. Fix all the compilation errors so that the program will compile successfully. Once the program compiles, compare the output with the sample output, and eliminate any logic errors that may exist. The sample output demonstrates what the program's output should be once the program's code has been corrected.

Sample Output

```
The value stored in variable name is:
The value stored in variable age is: 0

The value stored in variable name is:
The value stored in variable age is: 0

Enter a name: John
Enter an age: 44

Enter a name: John
Enter an age: 43

The value stored in variable name is: John
The value stored in variable age is: 44

The value stored in variable name is: John
The value stored in variable age is: 43

John has grown one year older

Both people have the same name and age
```

Broken Code

```cpp
// Debugging Problem: age.cpp

#include <iostream>

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

#include <cstring>

void initialize( char [], int * );
void input( const char [], int & );
void print( const char *, const int );
void growOlder( const char [], int * );
```

Fig. L 8.5 | debugging05.cpp. (Part 1 of 3.)

Lab Exercises

Name:

Debugging

```
bool comparePeople( const char *, const int *,
                    const char *, const int * );

int main()
{
   char name1[ 25 ];
   char name2[ 25 ];
   int age1;
   int age2;

   initialize( name1, &age1 );
   initialize( name2, &age2 );

   print( name1, *age1 );
   print( name2, *age2 );

   input( name1, age1 );
   input( name2, age2 );

   print( &name1, &age1Ptr );
   print( &name2, &age1Ptr );

   growOlder( name2, age2 );

   if ( comparePeople( name1, &age1, name2, &age2 ) )
      cout << "Both people have the same name and age"
           << endl;

   return 0;

} // end main

// function input definition
void input( const char name[], int &age )
{
   cout << "Enter a name: ";
   cin >> name;

   cout << "Enter an age: ";
   cin >> age;
   cout << endl;

} // end function input

// function initialize definition
void initialize( char name[], int *age )
{
   name = "";
   age = 0;

} // end function initialize

```

Fig. L 8.5 | debugging05.cpp. (Part 2 of 3.)

Lab Exercises

Name:

Debugging

```
// function print definition
void print( const char name[], const int age )
{
   cout << "The value stored in variable name is: "
        << name << endl
        << "The value stored in variable age is: "
        << age << endl << endl;

} // end function print

// function growOlder definition
void growOlder( const char name[], int *age )
{
   cout << name << " has grown one year older\n\n";
   *age++;

} // end function growOlder

// function comparePeople definition
bool comparePeople( const char *name1, const int *age1,
                    const char *name2, const int *age2 )
{
   return ( age1 == age2 && strcmp( name1, name2 ) );

} // end function comparePeople
```

Fig. L 8.5 | debugging05.cpp. (Part 3 of 3.)

Postlab Activities

Coding Exercises

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have completed the *Prelab Activities* and *Lab Exercises* successfully.

For each of the following problems, write a program or a program segment that performs the specified action.

1. Write three lines of code that declare three pointers to integers. Initialize one to `NULL`, another to `0` and the third to point to the address of `int` variable `x`.

2. Using the pointer you declared in *Coding Exercise 1*, that points to `x`, modify `x` by assigning to it the value of `4`.

3. Write a function prototype for the function `fn` that takes two pointers to integers as arguments and returns nothing.

4. Call function `fn` that you declared in *Coding Exercise 3*, and pass to it two of the pointers from *Coding Exercise 1* as arguments.

5. Apply the `sizeof` operator to all pointers from *Coding Exercise 1*.

Postlab Activities

Name:

Coding Exercises

6. Initialize a character string to a phrase of your choice. Traverse the contents of the string, using pointer-offset notation, and display the characters by `'+'`characters.

7. Traverse the contents of the string from *Coding Exercise 6*, this time using pointer subscript notation. In this exercise, each character output should be separated from adjacent characters by a `'-'` character.

8. Compare the first and third characters in the string from *Coding Exercises 6–7*. If they are equal, print "`equal.`" Use pointer-offset notation.

9. Write a program that inputs two strings—`s1` and `s2`—from the user. Use `strcpy` to copy the contents of `s2` into `s1`. Assume that user input is limited to 50 characters.

10. Use `strncat` to append the first five characters of `"today is a nice day"` to `"how old are you "`.

Postlab Activities

Name:

Programming Challenges

Name: ____________________________ **Date:** ____________________________

Section: ____________________________

The *Programming Challenges* are more involved than the *Coding Exercises* and may require a significant amount of time to complete. Write a C++ program for each of the problems in this section. The answers to these problems are available at `www.deitel.com` and `www.prenhall.com/deitel`. Pseudocode, hints and/or sample outputs are provided to aid you in your programming.

1. Modify the program in Figs. 8.25–8.27 of *C++ How to Program: Fifth Edition* so that the card-dealing function deals a five-card poker hand. Then write functions to accomplish each of the following tasks:

   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).

Hints:

- Allow class `DeckOfCards` to store a 5 card hand in a 5-by-2 array (five ranks and five suits). Implement the functions as member functions of class `DeckOfCards`.
- To determine if two (or more) cards are of the same rank, create an array of 13 elements—one for each rank—and initialize its elements to `0`. Iterate through the hand, incrementing the appropriate rank within your array. Finally, loop through the resulting array, checking for elements greater than or equal to `2` (or `3` or `4`).
- Similarly, check for a flush by creating a four-element array, where each element represents a different suit.
- Check for a straight by using the insertion sort.
- Sample output:

```
The hand is:
Deuce of Spades         King  of Clubs
Nine  of Diamonds       Jack  of Spades
Jack  of Hearts

The hand contains a pair of Jacks.
```

Postlab Activities

Name:

Programming Challenges

2. Write a program that inputs a line of text, tokenizes the line with function `strtok` and outputs the tokens in reverse order.

Hints:

- Allow the maximum amount of text input to be 100 characters long.
- Use `strtok` to tokenize the text, inserting all tokens into an array.
- Print the contents of this array, from the last element to the first.
- Sample output:

```
Enter a line of text:
twinkle twinkle little star

The tokens in reverse order are:
star little twinkle twinkle
```

3. 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 computer 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
```

Hints:

- Handle only amounts from 0 to 100.
- Create arrays of characters to pointers containing the words for all the digits and the tens places (forty, fifty, sixty, etc.) do not forget about the teens!
- Use the `%` operator to isolate certain digits.
- The implementation for amounts larger than 100 is similar.
- Sample output:

```
Enter check amount: 22.88
The protected amount is $****22.88
```

Postlab Activities

Name:

Programming Challenges

4. [*Note*: This problem is intended for those students who have studied recursion in Sections 6.19–6.21 of *C++ How to Program: Fifth Edition*.] The following grid of hashes (#) and dots (.) is a double-subscripted array representation of a maze:

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

In the preceding double-subscripted array, the hashes represent the walls of the maze, and the dots represent squares in the possible paths through the maze. Moves can be made only to a location in the array that contains a dot.

There is a simple algorithm for walking through a maze which guarantees that you will find the exit (assuming that there is an exit). If there is no 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, you will arrive at the exit of the maze eventually. There may be a shorter path than the one you have taken, but you are guaranteed to get out of the maze if you follow the algorithm.

Write recursive function `mazeTraverse` to walk through the maze. The function should receive as arguments a 12-by-12 character array representing the maze, and the starting location of the maze. As `mazeTraverse` attempts to locate the exit of 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.

Hints:

- The `mazeTraverse` function should be passed the array representing the maze, two starting coordinates (`x` and `y`) and the "right" wall.
- Write functions to determine if the coordinates are on the edge of the maze (only the starting and ending points are in this position) and if a move is legal. Also include a function to display the maze.

Postlab Activities

Name:

Programming Challenges

- Sample output:

```
# # # # # # # # # # # #
# . . . # . . . . . . #
x . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #

Hit return to see next move
.
.
.

# # # # # # # # # # # #
# x x x # x x x x x x #
x x # x # x # # # # x #
# # # x # x x x x # x #
# x x x x # # # x # x x
# # # # x # . # x # x #
# x x # x # . # x # x #
# # x # x # . # x # x #
# x x x x x x x x # x #
# # # # # # x # # # x #
# x x x x x x # x x x #
# # # # # # # # # # # #

Hit return to see next move


Maze successfully exited!
```