A First Book of ANSI C

Fourth Edition

Chapter 9
Character Strings
Objectives

- String Fundamentals
- Library Functions
- Input Data Validation
- Formatting Strings (Optional)
- Case Study: Character and Word Counting
- Common Programming and Compiler Errors
• On a fundamental level, strings are simply arrays of characters that can be manipulated using standard element-by-element array-processing techniques.
• On a higher level, string library functions are available for treating strings as complete entities.
• This chapter explores the input, manipulation, and output of strings using both approaches.
• We will also examine the particularly close connection between string-handling functions and pointers.
9.1 String Fundamentals

• A string literal is any sequence of characters enclosed in double quotes.
• A string literal is also referred to as a string constant and string value, and more conventionally as a string.
• For example, “This is a string”, “HelloWord!”, and “xyz123*!#@&” are all strings.
• Because a string is stored as an array of characters, the individual characters in the array can be input, manipulated, or output using standard array-handling techniques utilizing either subscript or pointer notations.

• The end-of-string null character is useful for detecting the end of the string when handing strings in this fashion.
String Input and Output

- Table 9.1 lists the commonly available library functions for both character-by-character and complete string input and output.

<table>
<thead>
<tr>
<th>Table 9.1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td>gets( )</td>
</tr>
<tr>
<td>scanf( )</td>
</tr>
<tr>
<td>getchar( )</td>
</tr>
</tbody>
</table>
Example of String Input and Output

Program 9.1 illustrates the use of `gets()` and `puts()` to input and output a string entered at the user’s terminal.

```c
#include <stdio.h>

int main()
{
    #define MSIZE 81
    char message[MSIZE]; /* enough storage for 80 characters plus '\0' */

    printf("Enter a string:\n");
    gets(message);
    printf("The string just entered is:\n");
    puts(message);

    return 0;
}
```

Sample run:
Enter a string:
This is a test input of a string of characters.
The string just entered is:
This is a test input of a string of characters.
• The `gets()` function used in Program 9.1 continuously accepts and stores the characters typed at the terminal into the character array named `message`.

• Pressing the Enter key at the terminal generates a newline character, `\n`, which is interpreted by `gets()` as the end-of-character entry.

• All the characters encountered by `gets()`, except the newline character, are stored in the `message` array.
• Before returning, the `gets()` function appends the null character to the stored set of characters, as illustrated in Figure 9.2a.
• The `puts()` function is then used to display the string.
• The `scanf()` function reads a set of characters up to either a blank space or a newline character, whereas `gets()` stops accepting characters only when a newline is detected.
• Trying to enter the characters *This is a string* using the statement `scanf("\%s", message);` results in the word *This* being assigned to the message array.

• Entering the complete line using a `scanf()` function call would require a statement such as

• `scanf("\%s \%s \%s \%s", message1, message2, message3, message4);`
• This allows us to understand how the standard library functions are constructed and to create our own library functions.

• For a specific example, consider the function `strcopy()`, which copies the contents of string2 to string1.

```c
void strcopy(char string1[ ],
            char string2[ ] )
{
    // i will be used as a subscript
    int i=0;
    while (string2[i]!="\0")
    {
        string1[i]=string2[i];
        i++;
    }
    // terminate the first string
    string1[i]="\0";
}
```
Program 9.2: includes the \texttt{strcpy()} function in a complete program

```c
#include<stdio.h>

/*@ expects two arrays of chars */
void strcpy(char[ ],char[ ]); 

int main( ) 
{
    // enough storage for a complete line 
    char message[81];
    // enough storage for a copy of message 
    char newMessage[81];
    int i;
    printf(" Enter a sentence: ");
    gets(message);
    strcpy(newMessage, message);
    // pass two array addresses*
    puts(newMessage);
    return 0;
}

/* copy string2 to string1 */
/* two arrays are passed */

void strcpy(char string1[ ],
            char string2[ ]); 

{
    int i=0; // i will be used as a subscript 
    /* check for the end-of-string */
    while( string2[i]!='\0' )
    {
        /* copy the element to string1*/
        string1[i]=string2[i];
        i++;
    }
    /* terminate the first string */
    string1[i]=\0;
}
```

Example Output

Enter a sentence: I am a CS CMU student.
I am a CS CMU student.
Character-by-Character Input

Program 9.3

```c
#include <stdio.h>
int main()
{
    #define LSIZE 81
    char message[LSIZE]; /* enough storage for 80 characters plus '\0' */
    char c;
    int i;

    printf("Enter a string:\n");
    i = 0;
    while(i < (LSIZE-1) && (c = getchar()) != '\n')
    {
        message[i] = c; /* store the character entered */
        i++;
    }
    message[i] = '\0'; /* terminate the string */

    printf("The string just entered is: \n");
    puts(message);
    return 0;
}
```

ข้อควรระวัง: ถ้าลืมใส่วงเล็บ จะมีค่าเท่ากับ

```
c = (getchar() != '\n')
```
String Processing (continued)

Program 9.4

```c
#include <stdio.h>

void getline(char []); /* function prototype */
#define LSIZE 81

int main()
{
    char message[LSIZE]; /* enough storage for 80 characters plus '\0' */

    printf("Enter a string: \n");
    getline(message);
    printf("The string just entered is: \n");
    puts(message);

    return 0;
}

void getline(char strng[])
{
    int i = 0;
    char c;

    while(i < (LSIZE-1) && (c = getchar()) != '\n')
    {
        strng[i] = c; /* store the character entered */
        i++;
    }
    strng[i] = '\0'; /* terminate the string */
}
```

สร้าง function ชื่อ `getline()` เพื่อใช้รับค่า string ที่ใช้ character ต่าง function ชื่อ `getchar()` จนกระทั่ง user เคาะ `enter`. 
# Library Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcpy(str1, str2)</code></td>
<td>Copies <code>str2</code> to <code>str1</code>, including the '\0'</td>
<td><code>strcpy(test, &quot;efgh&quot;)</code></td>
</tr>
<tr>
<td><code>strcat(str1, str2)</code></td>
<td>Appends <code>str2</code> to the end of <code>str1</code></td>
<td><code>strcat(test, &quot;there&quot;)</code></td>
</tr>
<tr>
<td><code>strlen(string)</code></td>
<td>Returns the length of string. Does not include the '\0' in the length count.</td>
<td><code>strlen(&quot;Hello World!&quot;)</code></td>
</tr>
<tr>
<td><code>strcmp(str1, str2)</code></td>
<td>Compares <code>str1</code> to <code>str2</code>. Returns a negative integer if <code>str1 &lt; str2</code>, 0 if <code>str1 == str2</code>, and a positive integer if <code>str1 &gt; str2</code>.</td>
<td><code>strcmp(&quot;Beb&quot;, &quot;Bee&quot;)</code></td>
</tr>
</tbody>
</table>

Note: Attempting to copy a larger string into a smaller string causes the copy to overflow the destination array beginning with the memory area immediately following the last array element.
Library Functions (continued)

• When comparing strings, their individual characters are evaluated in pairs; if a difference is found, the string with the first lower character is the smaller one
  – "Good Bye" is less than "Hello" because the first 'G' in Good Bye is less than the first 'H' in Hello
  – "Hello" is less than "Hello  " because the '\0' terminating the first string is less than the ' ' in the second string
  – "123" is greater than "122" because '3' in 123 is greater than '2' in 122
  – "1237" is greater than "123" because '7' in 1237 is greater than '\0' in 123
```c
#include <stdio.h>
#include <string.h> /* required for the string function library */

int main()
{
    #define MAXELS 50
    char string1[MAXELS] = "Hello";
    char string2[MAXELS] = "Hello there";
    int n;

    n = strcmp(string1, string2);

    if (n < 0)
        printf("%s is less than %s\n\n", string1, string2);
    else if (n == 0)
        printf("%s is equal to %s\n\n", string1, string2);
    else
        printf("%s is greater than %s\n\n", string1, string2);

    printf("The length of string1 is %d characters\n", strlen(string1));
}```
The length of string2 is %d characters

strcat(string1," there World!");

After concatenation, string1 contains the string value

%\n", string1);

The length of this string is %d characters

strlen(string1));

Type in a sequence of characters for string2:

gets(string2);

strcpy(string1, string2);

After copying string2 to string1"

the string value in string1 is:

%\n", string1);

The length of this string is %d characters

strlen(string1));

The starting address of the string1 string is: %d

(void *) string1); return 0;

}
Library Functions (continued)

• Sample output:
  Hello is less than Hello there

  The length of string1 is 5 characters
  The length of string2 is 11 characters

  After concatenation, string1 contains the string value
  Hello there World!
  The length of this string is 18 characters

  Type in a sequence of characters for string2:
  It's a wonderful day
  After copying string2 to string1, the string value in
  string1 is:
  It's a wonderful day
  The length of this string is 20 characters

  The starting address of the string1 string is: 1244836
# Character Routines

<table>
<thead>
<tr>
<th>Required Prototype</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int isalpha(char)</td>
<td>Returns a non-0 number if the character is a letter; otherwise, it returns 0.</td>
<td>isalpha('a')</td>
</tr>
<tr>
<td>int isupper(char)</td>
<td>Returns a non-0 number if the character is uppercase; otherwise, it returns 0.</td>
<td>isupper('a')</td>
</tr>
<tr>
<td>int islower(char)</td>
<td>Returns a non-0 number if the character is lowercase; otherwise, it returns 0.</td>
<td>islower('a')</td>
</tr>
<tr>
<td>int isdigit(char)</td>
<td>Returns a non-0 number if the character is a digit (0 through 9); otherwise, it returns 0.</td>
<td>isdigit('a')</td>
</tr>
<tr>
<td>int isascii(char)</td>
<td>Returns a non-0 number if the character is an ASCII character; otherwise, it returns 0.</td>
<td>isascii('a')</td>
</tr>
<tr>
<td>int isspace(char)</td>
<td>Returns a non-0 number if the character is a space; otherwise, it returns 0.</td>
<td>isspace(' ')</td>
</tr>
<tr>
<td>int isprint(char)</td>
<td>Returns a non-0 number if the character is a printable character; otherwise, it returns 0.</td>
<td>isprint('a')</td>
</tr>
<tr>
<td>int iscntrl(char)</td>
<td>Returns a non-0 number if the character is a control character; otherwise, it returns 0.</td>
<td>iscntrl('a')</td>
</tr>
<tr>
<td>int ispunct(char)</td>
<td>Returns a non-0 number if the character is a punctuation character; otherwise, it returns 0.</td>
<td>ispunct('!')</td>
</tr>
<tr>
<td>int toupper(char)</td>
<td>Returns the uppercase equivalent if the character is lowercase; otherwise, it returns the character unchanged.</td>
<td>toupper('a')</td>
</tr>
<tr>
<td>int tolower(char)</td>
<td>Returns the lowercase equivalent if the character is uppercase; otherwise, it returns the character unchanged.</td>
<td>tolower('A')</td>
</tr>
</tbody>
</table>
Character Routines (continued)

Program 9.6

```c
#include <stdio.h>
#include <ctype.h> /* required for the character function library */

int main()
{
    #define MAXCHARS 100
    char message[MAXCHARS];
    void convertToUpper(char []); /* function prototype */

    printf("\nType in any sequence of characters:\n");
    gets(message);
    convertToUpper(message);

    printf("The characters just entered, in uppercase are:\n%sn", message);
    return 0;
}

// this function converts all lowercase characters to uppercase
void convertToUpper(char message[])
{
    int i;
    for(i = 0; message[i] != '\0'; i++)
        message[i] = toupper(message[i]);
}
```
## Conversion Routines

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int atoi(string)</td>
<td>Converts an ASCII string to an integer. Conversion stops at the first noninteger character.</td>
<td>atoi(&quot;1234&quot;)</td>
</tr>
<tr>
<td>double atof(string)</td>
<td>Converts an ASCII string to a double-precision number. Conversion stops at the first character that cannot be interpreted as a double.</td>
<td>atof(&quot;12.34&quot;)</td>
</tr>
<tr>
<td>char[] itoa(string)</td>
<td>Converts an integer to an ASCII string. The space allocated for the returned string must be large enough for the converted value.</td>
<td>itoa(1234)</td>
</tr>
</tbody>
</table>
Program 9.7

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h> // required for test conversion function library

int main()
{
    #define MAXELS 20
    char test[MAXELS] = "1234";
    int num;
    double dnum;

    num = atoi(test);
    printf("The string %s as an integer number is %d\n", test, num);
    printf("This number divided by 3 is: %d\n", num/3);

    strcat(test, ".96");

    dnum = atof(test);
    printf("\nThe string %s as a double number is: %f\n", test, dnum);
    printf("This number divided by 3 is: %f\n", dnum/3);

    return 0;
}
```
Input Data Validation

• Successful programs always try to anticipate invalid data and isolate such data from being accepted and processed
  – First validate that the data is of the correct type; if not, request the user to re-enter the data
  – Explain why the entered data was invalid

• One of the most common methods of validating input data is to accept all numbers as strings
  – Each character can then be checked to ensure that it complies with the data type being requested
**Program 9.8**

```c
#include <stdio.h>
#include <stdlib.h> /* needed to convert a string to an integer */
#define MAXCHARS 40
#define TRUE 1
#define FALSE 0

int isvalidInt(char []); /* function prototype */

int main()
{

    char value[MAXCHARS];
    int number;

    printf("Enter an integer: ");
    gets(value);

    if (isvalidInt(value)== TRUE)
    {
        number = atoi(value);
        printf("The number you entered is %d\n", number);
    }
    else
        printf("The number you entered is not a valid integer.\n");

    return 0;
}
```
```c
int isvalidInt(char val[])
{
    int start = 0;
    int i;
    int valid = TRUE;
    int sign = FALSE;

    /* check for an empty string */
    if (val[0] == '\0') valid = FALSE;

    /* check for a leading sign */
    if (val[0] == '-' || val[0] == '+')
    {
        sign = TRUE;
        start = 1; /* start checking for digits after the sign */
    }

    /* check that there is at least one character after the sign */
    if (sign == TRUE && val[1] == '\0') valid = FALSE;

    /* now check the string, which we know has at least one non-sign char */
    i = start;
    while(valid == TRUE && val[i] != '\0')
    {
        if (val[i] < '0' || val[i] > '9') /* check for a non-digit */
            valid = FALSE;
        i++;
    }

    return valid;
}
```
Input Data Validation (continued)

• We can use `isvalidInt()` in a loop that continually requests an integer until a valid integer value is entered.

  *Set an integer variable named `isanInt` to 0*

  `do`

  Accept a string value

  *If the string value does not correspond to an integer*

    Display the error message "Invalid integer - Please re-enter: "

    Send control back to expression being tested by the do-while statement

  *Set `isanInt` to 1 (this causes the loop to terminate)*

  `while(isanInt is 0)`

  *Return the integer corresponding to the entered string*
Program 9.9

...  

17    #define TRUE 1  
18    #define FALSE 0 
19    #define MAXCHARS 40 
20    int getanInt() 
21    { 
22    int isValidInt(char []); /* function prototype */ 
23               
24    int isanInt = FALSE; 
25    char value[MAXCHARS]; 
26 
27    do 
28    { 
29        gets(value); 
30        if (isValidInt(value) == FALSE) 
31           { 
32            printf("Invalid integer - Please re-enter: "); 
33            continue; /* send control to the do-while expression test */ 
34           } 
35        isanInt = TRUE; 
36    }while (isanInt == FALSE); 
37  
38    return (atoi(value)); /* convert to an integer */ 
39    }

A First Book of ANSI C, Fourth Edition 28
Creating a Personal Library

• Programmers create their own libraries of functions
  – This permits the functions to be incorporated in any program without further expenditure of coding time
• Each file in a library contains related functions
  – #include <C:\mylibrary\dataChecks.h>
  – #include "C:\mylibrary\dataChecks.h"
  • The #include statement for dataChecks.h must be placed after the #include statements for the stdio.h and stdlib.h header files (the functions in dataChecks.h require stdio.h and stdlib.h functions to correctly compile)
Formatting Strings

• Examples:
  − `printf("|%25s|","Have a Happy Day");`
    • | Have a Happy Day |
  − `printf("|%-25s|","Have a Happy Day");`
    • |Have a Happy Day | |
  − `printf("|%25.12s|","Have a Happy Day");`
    • | Have a Happy | |
  − `printf("|%.12s|","Have a Happy Day");`
    • |Have a Happy|
In-Memory String Conversions

• The `sprintf()` and `sscanf()` functions provide capabilities for writing and scanning strings to and from memory variables
  - `sprintf(disStrn,"%d %d", num1, num2);`
  - `sscanf(data,"%c%lf %d", &dol,&price,&units);`
    • "$23.45 10"
  - `sscanf(date,"%d/%d/%d", &month, &day, &year);`
    • "07/01/94"
Format Strings

• The control string containing the conversion control sequences need not be explicitly contained within the function
  – `printf("$%5.2f %d",num1,num2);`
  – Or,
    ```
    char fmat[] = "$%5.2f %d";
    printf(fmat,num1,num2);
    ```
• Useful for listing format strings with other variable declarations at the beginning of a function
  – If you need to change a format, it is easy to find the desired control string without searching to locate the appropriate `printf()` or `scanf()` function calls
Case Study: Character and Word Counting

• We construct two string-processing functions
  – Count the number of characters in a string
  – Count words in a string
    • What constitutes a word?
Program Requirement: Character Counting

• Pass a string to a function and have the function return the number of characters in the string
• Any character in the string (blank, printable, or nonprintable character) is to be counted
• The end-of-string NULL character is not to be included in the final count
Analyze the Problem

- Determine the input data
- Determine the required outputs
- List the algorithm(s) relating the inputs to the outputs
Analyze the Problem (continued)

Figure 9.5  Counting characters in a string
Code the Function

```c
int countchar(char list[])
{
    int i, count = 0;
    for(i = 0; list[i] != '\0'; i++)
        count++;
    return(count);
}
```
Test and Debug the Function

Program 9.10

```c
#include <stdio.h>
#define MAXNUM 1000

int countchar(char []); /* function prototype */

int main()
{
    char message[MAXNUM];
    int numchar;

    printf("\nType in any number of characters: ");
    gets(message);
    numchar = countchar(message);
    printf("The number of characters just entered is %d\n", numchar);

    return 0;
}
...```

A First Book of ANSI C, Fourth Edition 38
Requirement Specification: Word Counting

• The last word does not have a trailing blank
• More than one blank may be used between words
• Leading blanks may be used before the first word
Analyze the Problem

- Determine the input data
- Determine the required outputs
- Algorithm:
  
  Set an integer variable named inaword to the symbolic constant NO
  Set the word count to 0
  For all the characters in the array
    If the current character is a blank
      set inaword to NO
    Else if (inaword equals NO)
      set inaword to the symbolic constant YES
      increment the word count
    EndIf
  EndFor
  EndFor
Return the count
int countword(char list[]) 
#define YES 1
#define NO 0
{
    int i, inaword, count = 0;
inaword = NO;
    for(i = 0; list[i] != '\0'; i++)
    {
        if (list[i] == ' ')
            inaword = NO;
        else if (inaword == NO)
        {
            inaword = YES;
            count++;
        }
    }
    return(count);
}
Test and Debug the Function

Program 9.11

```c
#include <stdio.h>
#define MAXNUM 1000

int countword(char []); /* function prototype */

int main()
{
    char message[MAXNUM];
    int numchar;

    printf("\nType in any number of words: ");
    gets(message);
    numchar = countword(message);
    printf("The number of words just entered is %d\n", numchar);
    return 0;
}
```

Test and Debug the Function (continued)

• A sample run using Program 9.11 follows:
  Type in any number of words: This is a test line with a bunch of words
  The number of words just entered is 10

• Further tests that should be performed are
  – Enter words with multiple spaces between them
  – Enter words with leading spaces before the first word
  – Enter words with trailing spaces after the last word
  – Enter a sentence that ends in a period or question mark
Common Programming Errors

• Forgetting the terminating NULL character, '\0', when processing existing strings in a character-by-character manner
• Forgetting to terminate a newly created character string with the NULL character
• Forgetting that the newline character, '\n', is a valid data input character
• Forgetting to include the `string.h`, `ctype.h`, and `stdlib.h` header files when using the string library, character library, and conversion library functions, respectively
## Common Compiler Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Typical Unix-based Compiler Error Message</th>
<th>Typical Windows-based Compiler Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempting to assign a single character into an element of the array using double, rather than single quotes. For example, message[5] = &quot;A&quot;;</td>
<td>(W) Operation between types &quot;unsigned char&quot; and &quot;unsigned char*&quot; is not allowed.</td>
<td>error : cannot convert from 'const char [2]' to 'char'</td>
</tr>
<tr>
<td>Not using a system predefined constant in all capital letters. For example, message[10] = NULL;</td>
<td>(S) Undeclared identifier NULL.</td>
<td>error: 'NULL' : undeclared identifier</td>
</tr>
<tr>
<td>Forgetting to insert a length in the Size of the Array without initializers. For example, char message[];</td>
<td>(S) Explicit dimension specification or initializer required for an auto or static array.</td>
<td>error: 'message' : unknown size</td>
</tr>
</tbody>
</table>
Common Compiler Errors (continued)

<table>
<thead>
<tr>
<th>Error</th>
<th>Typical Unix-based Compiler Error Message</th>
<th>Typical Windows-based Compiler Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparing against an escape sequence that is inside double quotes. For example, while((c = getchar()) != &quot;\n&quot;)</td>
<td>(W) Operation between types &quot;int&quot; and &quot;unsigned char*&quot; is not allowed.</td>
<td>error: '!=' : no conversion from 'const char *' to 'int'</td>
</tr>
<tr>
<td>Providing an incorrect path for including header files. For example, #include &quot;c:\stdio.h&quot;</td>
<td>(S) #include file &quot;c:\stdio.h&quot; not found.</td>
<td>fatal error: Cannot open include file: 'c:\stdio.h': No such file or directory</td>
</tr>
</tbody>
</table>
String & Pointer

#include <stdio.h>
int main()
{
    char *message2 = "this is a string";

    printf("\nThe string is: %s", message2);
    printf("\nThe first address of this string is %p", message2);

    message2 = "A new message";
    printf("\nThe string is now: %s", message2);
    printf("\nThe first address of this string is %p", message2);

    return 0;
}

The string is: this is a string
The first address of this string is 00420094
The string is now: A new message
The first address of this string is 00420038
Press any key to continue
String & Pointer (Cont.)

message2 is a pointer variable

First the address points here

The address of this location is initially stored in message2

Then the address is changed to point here

The address of this location is then stored in message2

```
this is a string \0
```

```
A new message \0
```
Pointer Arrays (Program)

```c
#include <stdio.h>

int main()
{
    int n;
    char *seasons[] = {"Winter",
                       "Spring",
                       "Summer",
                       "Fall"};

    for(n = 0; n < 4; ++n)
        printf("\nThe season is %s.\n", seasons[n]);

    return 0;
}
```

The season is Winter.
The season is Spring.
The season is Summer.
The season is Fall. Press any key to continue
Pointer Arrays

seasons[0] = "Winter";
seasons[1] = "Spring";
seasons[2] = "Summer";
seasons[3] = "Fall";
Scaling a set of numbers into a more useful set

```c
#include <stdio.h>
int main()
{
    int n;
    char *seasons[] = { "Winter",
        "Spring",
        "Summer",
        "Fall"};

    printf("\nEnter a month (use 1 for Jan., 2 for Feb., etc.): ");
    scanf("%d", &n);
    n = (n % 12) / 3; /* create the correct subscript */
    printf("The month entered is a %s month.\n", seasons[n]);

    return 0;
}
```
Scaling a set of numbers into a more useful set (Cont.)

<table>
<thead>
<tr>
<th>Months</th>
<th>Season</th>
</tr>
</thead>
<tbody>
<tr>
<td>December, January, February</td>
<td>Winter</td>
</tr>
<tr>
<td>March, April, May</td>
<td>Spring</td>
</tr>
<tr>
<td>June, July, August</td>
<td>Summer</td>
</tr>
<tr>
<td>September, October, November</td>
<td>Fall</td>
</tr>
</tbody>
</table>

Enter a month (use 1 for Jan., 2 for Feb., etc.): 5
The month entered is a Spring month. Press any key to continue.
Summary

• A string is an array of characters terminated by the NULL (\0) character

• Character arrays can be initialized using a string assignment of the form char arrayName[] = "text";

• Strings can always be processed using standard array-processing techniques

• The gets(), scanf(), and getchar() library functions can be used to input a string

• The puts(), printf(), and putchar() functions can be used to display strings
Summary (continued)

• Many standard library functions exist for processing strings as a complete unit
• The standard C library also includes individual character-handling functions (ctype.h)
• One of the major uses of strings is validating user input, which is an essential part of any program
• The conversion routines atoi() and atof() are provided in the stdlib.h header file for converting strings to integer and double-precision numeric values