

# 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 9.1**

<b>Input</b>	<b>Output</b>
gets( )	puts( )
scanf( )	printf( )
getchar( )	putchar( )

# 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.

```
1  #include <stdio.h>
2  int main()
3  {
4      #define MSIZE 81
5      char message[MSIZE]; /* enough storage for 80 characters plus '\0' */
6
7      printf("Enter a string:\n");
8      gets(message);
9      printf("The string just entered is:\n");
10     puts(message);
11
12     return 0;
13 }
```

### 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 `strcpy( )`, which copies the contents of `string2` to `string1`.

```
void strcpy(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 *strcpy( )* function in a complete program

```
#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++;
    }
    /* teminate 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

```
1  #include <stdio.h>
2  int main()
3  {
4      #define LSIZE 81
5      char message[LSIZE]; /* enough storage for 80 characters plus '\0' */
6      char c;
7      int i;
8
9      printf("Enter a string:\n");
10     i = 0;
11     while(i < (LSIZE-1) && (c = getchar()) != '\n')
12     {
13         message[i] = c; /* store the character entered */
14         i++;
15     }
16     message[i] = '\0'; /* terminate the string */
17     printf("The string just entered is: \n");
18     puts(message);
19
20     return 0;
21 }
```

รับค่า *string* ทีละ *character* ผ่าน function ชื่อ *getchar()* จนกระทั่ง *user* เคะะ *enter*.

ข้อควรระวัง: ถ้าลืมใส่วงเล็บ จะมีค่าเท่ากับ *c = (getchar() != '\n')*

# String Processing (continued)



## Program 9.4

```
1  #include <stdio.h>
2  void getline(char []); /* function prototype */
3  #define LSIZE 81
4
5  int main()
6  {
7      char message[LSIZE]; /* enough storage for 80 characters plus '\0' */
8
9      printf("Enter a string: \n");
10     getline(message);
11     printf("The string just entered is:\n");
12     puts(message);
13
14     return 0;
15 }
16
17 void getline(char strng[])
18 {
19     int i = 0;
20     char c;
21
22     while(i < (LSIZE-1) && (c = getchar()) != '\n')
23     {
24         strng[i] = c; /* store the character entered */
25         i++;
26     }
27     strng[i] = '\0'; /* terminate the string */
28 }
```

สร้าง function ชื่อ *getline( )* เพื่อใช้รับค่า *string* ทีละ *character* ผ่าน function ชื่อ *getchar( )* จนกระทั่ง *user* เคาะ *enter*.

# Library Functions

**Table 9.2** String Library Routines (Required Header File is `string.h`)

Name	Description	Example
<code>strcpy(str1, str2)</code>	Copies <code>str2</code> to <code>str1</code> , including the <code>'\0'</code>	<code>strcpy(test, "efgh")</code>
<code>strcat(str1, str2)</code>	Appends <code>str2</code> to the end of <code>str1</code>	<code>strcat(test, "there")</code>
<code>strlen(string)</code>	Returns the length of <code>string</code> . Does not include the <code>'\0'</code> in the length count.	<code>strlen("Hello World!")</code>
<code>strcmp(str1, str2)</code>	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> .	<code>strcmp("Beb", "Bee")</code>

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



# Library Functions (continued)



## Program 9.5

```
1  #include <stdio.h>
2  #include <string.h> /* required for the string function library */
3
4  int main()
5  {
6      #define MAXELS 50
7      char string1[MAXELS] = "Hello";
8      char string2[MAXELS] = "Hello there";
9      int n;
10
11     n = strcmp(string1, string2);
12
13     if (n < 0)
14         printf("%s is less than %s\n\n", string1, string2);
15     else if (n == 0)
16         printf("%s is equal to %s\n\n", string1, string2);
17     else
18         printf("%s is greater than %s\n\n", string1, string2);
19
20     printf("The length of string1 is %d characters\n", strlen(string1));
```

# Library Functions (continued)

```
21  printf("The length of string2 is %d characters\n\n", strlen(string2));
22
23  strcat(string1, " there World!");
24
25  printf("After concatenation, string1 contains the string value\n");
26  printf("%s\n", string1);
27  printf("The length of this string is %d characters\n\n",
28          strlen(string1));
29  printf("Type in a sequence of characters for string2:\n");
30  gets(string2);
31
32  strcpy(string1, string2);
33
34  printf("After copying string2 to string1");
35  printf(" the string value in string1 is:\n");
36  printf("%s\n", string1);
37  printf("The length of this string is %d characters\n\n",
38          strlen(string1));
39  printf("\nThe starting address of the string1 string is: %d\n",
40          (void *) string1);
41  return 0;
42 }
```

# 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 9.3** Character Library Routines (Required Header File is `ctype.h`)

Required Prototype	Description	Example
<code>int isalpha(char)</code>	Returns a non-0 number if the character is a letter; otherwise, it returns 0.	<code>isalpha('a')</code>
<code>int isupper(char)</code>	Returns a non-0 number if the character is uppercase; otherwise, it returns 0.	<code>isupper('A')</code>
<code>int islower(char)</code>	Returns a non-0 number if the character is lowercase; otherwise, it returns 0.	<code>islower('a')</code>
<code>int isdigit(char)</code>	Returns a non-0 number if the character is a digit (0 through 9); otherwise, it returns 0.	<code>isdigit('5')</code>
<code>int isascii(char)</code>	Returns a non-0 number if the character is an ASCII character; otherwise, it returns 0.	<code>isascii('a')</code>
<code>int isspace(char)</code>	Returns a non-0 number if the character is a space; otherwise, it returns 0.	<code>isspace(' ')</code>
<code>int isprint(char)</code>	Returns a non-0 number if the character is a printable character; otherwise, it returns 0.	<code>isprint('a')</code>
<code>int iscntrl(char)</code>	Returns a non-0 number if the character is a control character; otherwise, it returns 0.	<code>iscntrl('\n')</code>
<code>int ispunct(char)</code>	Returns a non-0 number if the character is a punctuation character; otherwise, it returns 0.	<code>ispunct('!')</code>
<code>int toupper(char)</code>	Returns the uppercase equivalent if the character is lowercase; otherwise, it returns the character unchanged.	<code>toupper('a')</code>
<code>int tolower(char)</code>	Returns the lowercase equivalent if the character is uppercase; otherwise, it returns the character unchanged.	<code>tolower('A')</code>

# Character Routines (continued)



## Program 9.6

```
1  #include <stdio.h>
2  #include <ctype.h> /* required for the character function library */
3
4  int main()
5  {
6      #define MAXCHARS 100
7      char message[MAXCHARS];
8      void convertToUpper(char []); /* function prototype */
9
10     printf("\nType in any sequence of characters:\n");
11     gets(message);
12
13     convertToUpper(message);
14
15     printf("The characters just entered, in uppercase are:\n%s\n", message);
16
17     return 0;
18 }
19
20 // this function converts all lowercase characters to uppercase
21 void convertToUpper(char message[])
22 {
23     int i;
24     for(i = 0; message[i] != '\0'; i++)
25         message[i] = toupper(message[i]);
26 }
```

# Conversion Routines

**Table 9.4** Conversion Routines (Required Header File is `stdlib.h`)

Prototype	Description	Example
<code>int atoi(string)</code>	Converts an ASCII string to an integer. Conversion stops at the first noninteger character.	<code>atoi("1234")</code>
<code>double atof(string)</code>	Converts an ASCII string to a double-precision number. Conversion stops at the first character that cannot be interpreted as a double.	<code>atof("12.34")</code>
<code>char[] itoa(string)</code>	Converts an integer to an ASCII string. The space allocated for the returned string must be large enough for the converted value.	<code>itoa(1234)</code>

# Conversion Routines (continued)



## Program 9.7

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h> // required for test conversion function library
4
5  int main()
6  {
7  #define MAXELS 20
8      char test[MAXELS] = "1234";
9      int num;
10     double dnum;
11
12     num = atoi(test);
13     printf("The string %s as an integer number is %d\n", test,num);
14     printf("This number divided by 3 is: %d\n", num/3);
15
16     strcat(test, ".96");
17
18     dnum = atof(test);
19     printf("\nThe string %s as a double number is: %f\n", test,dnum);
20     printf("This number divided by 3 is: %f\n", dnum/3);
21
22     return 0;
23 }
```

# 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



# Input Data Validation (continued)



## Program 9.8

```
1  #include <stdio.h>
2  #include <stdlib.h> /* needed to convert a string to an integer */
3  #define MAXCHARS 40
4  #define TRUE 1
5  #define FALSE 0
6
7  int isValidInt(char []); /* function prototype */
8
9  int main()
10 {
11
12     char value[MAXCHARS];
13     int number;
14
15     printf("Enter an integer: ");
16     gets(value);
17
18     if (isValidInt(value) == TRUE)
19     {
20         number = atoi(value);
21         printf("The number you entered is %d\n", number);
22     }
23     else
24         printf("The number you entered is not a valid integer.\n");
25
26     return 0;
27 }
28
```

# Input Data Validation (continued)

```
29 int isValidInt(char val[])
30 {
31     int start = 0;
32     int i;
33     int valid = TRUE;
34     int sign = FALSE;
35
36     /* check for an empty string */
37     if (val[0] == '\0') valid = FALSE;
38
39     /* check for a leading sign */
40     if (val[0] == '-' || val[0] == '+')
41     {
42         sign = TRUE;
43         start = 1; /* start checking for digits after the sign */
44     }
45
46     /* check that there is at least one character after the sign */
47     if(sign == TRUE && val[1] == '\0') valid = FALSE;
48
49     /*now check the string, which we know has at least one non-sign char */
50     i = start;
51     while(valid == TRUE && val[i] != '\0')
52     {
53         if (val[i] < '0' || val[i] > '9') /* check for a non-digit */
54             valid = FALSE;
55         i++;
56     }
57
58     return valid;
59 }
```

# 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***

# Input Data Validation (continued)



## 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  }
```

# 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?

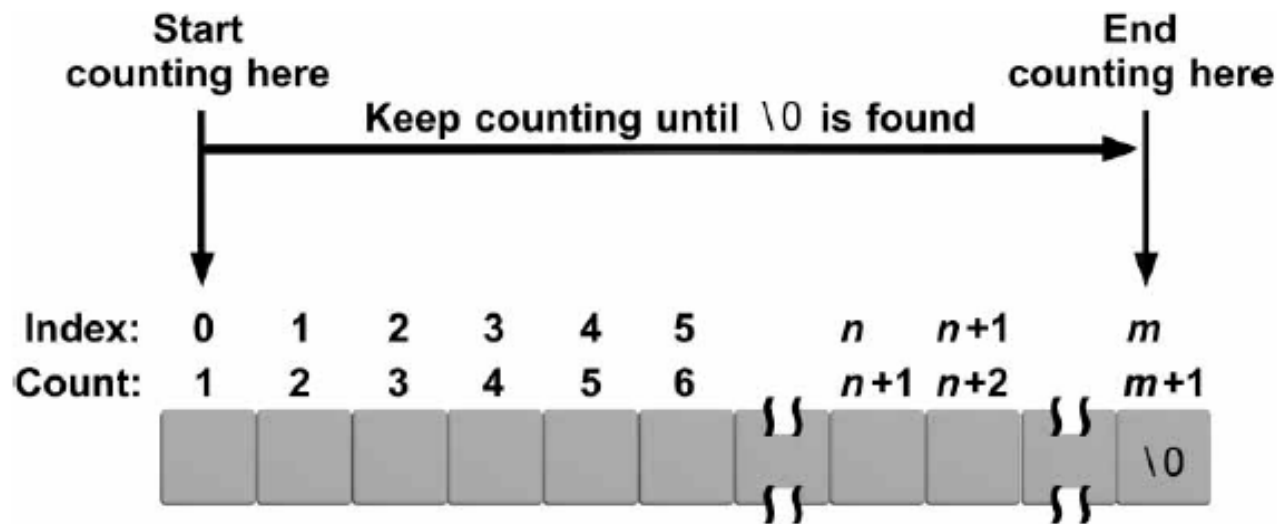
# Program Requirement: Character Counting

- 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

```
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

```
1  #include <stdio.h>
2  #define MAXNUM 1000
3
4  int countchar(char []); /* function prototype */
5
6  int main()
7  {
8      char message[MAXNUM];
9      int numchar;
10
11     printf("\nType in any number of characters: ");
12     gets(message);
13     numchar = countchar(message);
14     printf("The number of characters just entered is %d\n", numchar);
15
16     return 0;
17 }
18 ...
```

# 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*
  - Return the count*



# Code the Function

```
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

```
1  #include <stdio.h>
2  #define MAXNUM 1000
3
4  int countword(char []); /* function prototype */
5
6  int main()
7  {
8      char message[MAXNUM];
9      int numchar;
10
11     printf("\nType in any number of words: ");
12     gets(message);
13     numchar = countword(message);
14     printf("The number of words just entered is %d\n", numchar);
15
16     return 0;
17 }
18 ...
```

# 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

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

# Common Compiler Errors (continued)

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

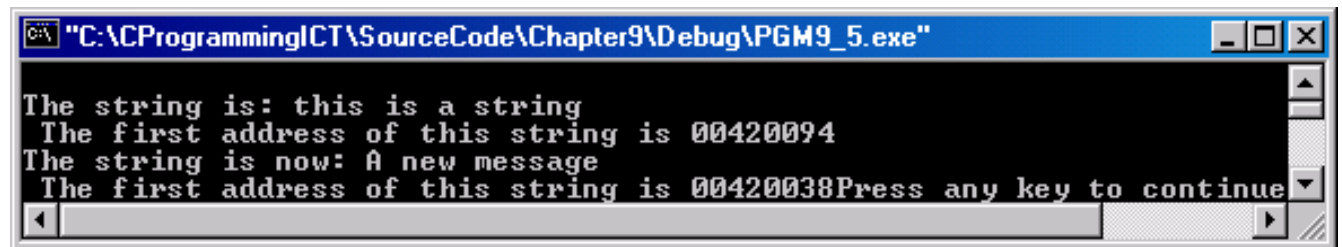
# String & Pointer

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

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

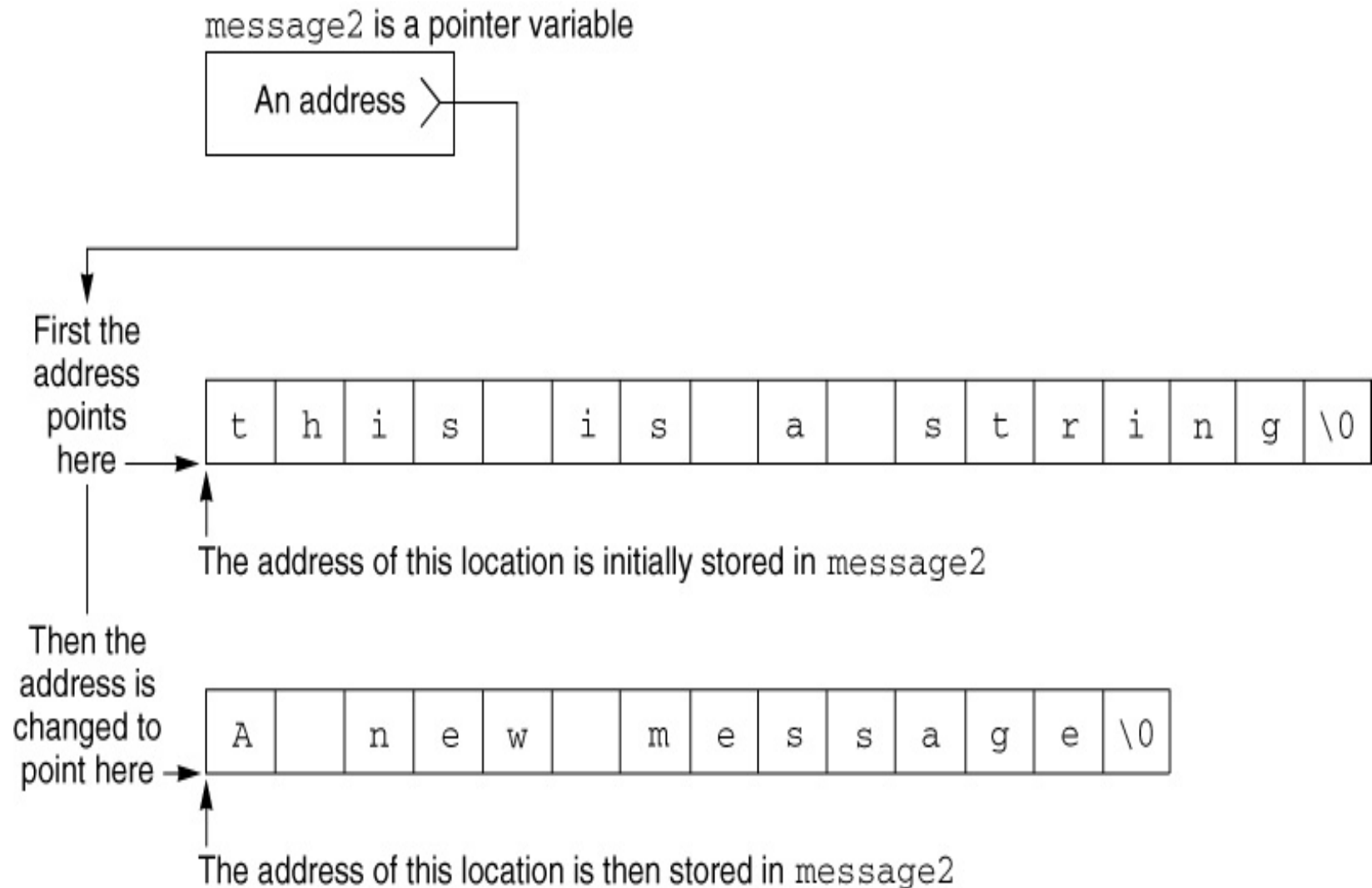
    message2 = "A new message";
    printf("\nThe string is now: %s", message2);
    printf("\n The first address of this string is %p", message2);

    return 0;
}
```

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\CProgramming\CT\SourceCode\Chapter9\Debug\PGM9\_5.exe". The window has standard minimize, maximize, and close buttons. The command prompt area is black with white text. It displays the output of the C program: "The string is: this is a string", "The first address of this string is 00420094", "The string is now: A new message", and "The first address of this string is 00420038". At the bottom, it says "Press any key to continue". A scroll bar is visible on the right side of the window.

```
C:\CProgramming\CT\SourceCode\Chapter9\Debug\PGM9_5.exe
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 00420038Press any key to continue
```

# String & Pointer (Cont.)





# Pointer Arrays (Program)

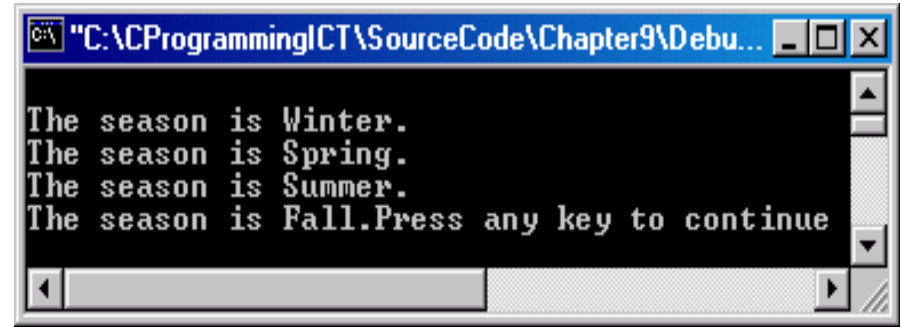
```
#include <stdio.h>

int main()
{
    int n;

    char *seasons[] = { "Winter",
                        "Spring",
                        "Summer",
                        "Fall"};

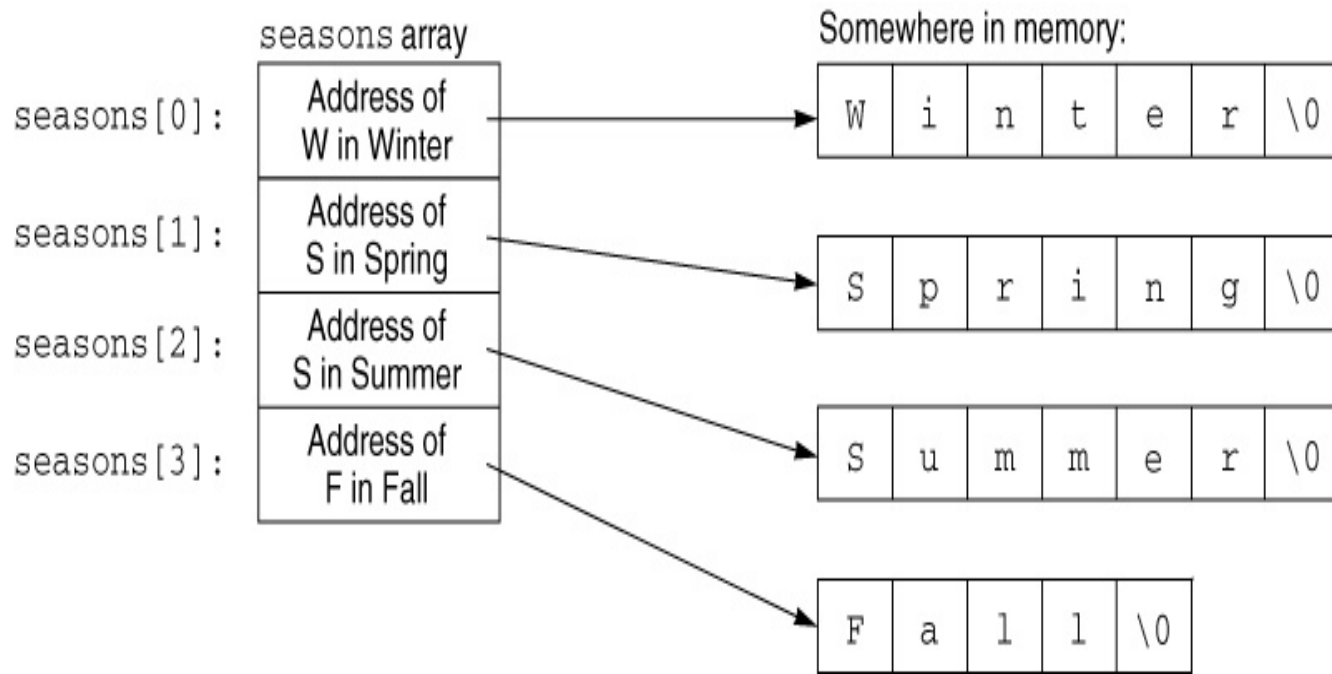
    for(n = 0; n < 4; ++n)
        printf("\nThe season is %s.",seasons[n]);

    return 0;
}
```

A screenshot of a Windows command prompt window. The title bar shows the path "C:\CProgramming\CT\SourceCode\Chapter9\Debu...". The window contains the following text:

```
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

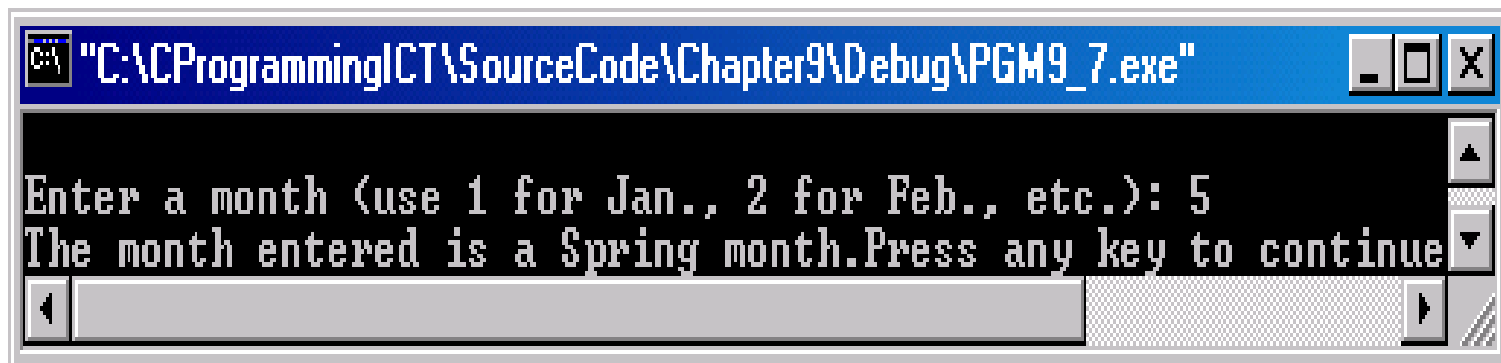
```
#include <stdio.h>
int main()
{
    int n;
    char *seasons[] = { "Winter",
                        "Spring",
                        "Summer",
                        "Fall" };

    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.", seasons[n]);

    return 0;
}
```

# Scaling a set of numbers into a more useful set (Cont.)

Months	Season
December, January, February	Winter
March, April, May	Spring
June, July, August	Summer
September, October, November	Fall



```
"C:\Programming\CT\SourceCode\Chapter9\Debug\PGM9_7.exe"
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