UNIT 7 STRINGS

Structure

7.0 INTRODUCTION

In the previous unit, we have discussed numeric arrays, a powerful data storage method that lets you group a number of same-type data items under the same group name. Individual items, or elements, in an array are identified using a subscript after the array name. Computer programming tasks that involve repetitive data processing lend themselves to array storage. Like non-array variables, arrays must be declared before they can be used. Optionally, array elements can be initialized when the array is declared. In the earlier unit, we had just known the concept of *character arrays* which are also called *strings*.

String can be represented as a single-dimensional character type array. C language does not provide the intrinsic string types. Some problems require that the characters within a string be processed individually. However, there are many problems which require that strings be processed as complete entities. Such problems can be manipulated considerably through the use of special string oriented library functions. Most of the C compilers include string library functions that allow string comparison, string copy, concatenation of strings etc. The string functions operate on null-terminated arrays of characters and require the header <string.h>.The use of the some of the string library functions are given as examples in this unit.

7.1 OBJECTIVES

After going through this unit, you will be able to:

- define, declare and initialize a string;
- discuss various formatting techniques to display the strings; and
- discuss various built-in string functions and their use in manipulation of strings.

7.2 DECLARATION AND INITIALIZATION OF STRINGS

Strings in C are group of characters, digits, and symbols enclosed in quotation marks or simply we can say the string is declared as a “character array”. The end of the string is marked with a special character, the ‘\0’ (*Null character)*, which has the decimal value 0. There is a difference between a *character* stored in memory and a

single character string stored in a memory. The character requires only one byte whereas the single character string requires two bytes (one byte for the character and other byte for the delimiter).

Declaration of strings

A string in C is simply a sequence of characters. To declare a string, specify the data type as char and place the number of characters in the array in square brackets after the string name. The syntax is shown as below:

char string-name[size];

For example,

char name[20];
char address[25];
char city[15];

Initialization of strings

The string can be initialized as follows:

char name[8] = {'P', 'R', 'O', 'G', 'R', 'A', 'M', '\0'};

Each character of string occupies 1 byte of memory (on 16 bit computing). The size of character is machine dependent, and varies from 16 bit computers to 64 bit computers. The characters of strings are stored in the contiguous (adjacent) memory locations.

1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte
P	R	O	G	R	A	M	\0
1001	1002	1003	1004	1005	1006	1007	1008

The C compiler inserts the NULL (\0) character automatically at the end of the string. So initialization of the NULL character is not essential.

You can set the initial value of a character array when you declare it by specifying a string literal. If the array is too small for the literal, the literal will be truncated. If the literal (including its null terminator) is smaller than the array, then the final characters in the array will be undefined. If you don't specify the size of the array, but do specify a literal, then C will set the array to the size of the literal, including the null terminator.

```
char str[4] = {'u', 'n', 'i', 'x'};
char str[5] = {'u', 'n', 'i', 'x', '\0'};
char str[3];
char str[ ] =  "UNIX";
char str[4] = "unix";
char str[9] = "unix";
```

All of the above declarations are legal. But which ones don't work? The first one is a valid declaration, but will cause major problems because it is not *null-terminated.* The second example shows a correct null-terminated string. The special escape character **\0** denotes string termination. The fifth example suffers the size problem, the character array ***'str'*** is of size 4 bytes, but it requires an additional space to store ***'\0'***. The fourth example however does not. This is because the compiler will determine the length of the string and automatically initialize the last character to a null-terminator. The strings not terminated by a ***'\0'*** are merely a collection of characters and are called as *character arrays*.

String Constants

String constants have double quote marks around them, and can be assigned to char pointers. Alternatively, you can assign a string constant to a char array - either with no size specified, or you can specify a size, but don't forget to leave a space for the null character! Suppose you create the following two code fragments and run them:

```
/*   Fragment 1   */
{
   char *s;
   s=hello";
   printf("%s\n",s);
}

/*  Fragment  2 */

{
   char s[100];
   strcpy(s, " hello");
   printf("%s\n",s);
}
```

These two fragments produce the same output, but their internal behaviour is quite different. In fragment 2, you cannot say **s = "hello";**. To understand the differences, you have to understand how the *string constant table* works in C. When your program is compiled, the compiler forms the object code file, which contains your machine code and a table of all the string constants declared in the program. In fragment 1, the statement **s = "hello";** causes ***s*** to point to the address of the string **hello** in the string constant table. Since this string is in the string constant table, and therefore technically a part of the executable code, you cannot modify it. You can only point to it and use it in a read-only manner. In fragment 2, the string **hello** also exists in the constant table, so you can copy it into the array of characters named ***s***. Since ***s*** is not an address, the statement **s="hello";** will not work in fragment 2. It will not even compile.

Example 7.1

Write a program to read a name from the keyboard and display message **Hello** onto the monitor

Program 7.1

```
/*Program that reads the name and display the hello along with your name*/
#include <stdio.h>
main()
{
char name[10];
printf("\nEnter Your Name : ");
scanf("%s", name);
printf("Hello %s\n", name);
}
```

OUTPUT

```
Enter Your Name :  Alex
Hello Alex
```

In the above example declaration char name [10] allocates 10 bytes of memory space (on 16 bit computing) to array name [ ]. We are passing the base address to scanf function and scanf() function fills the characters typed at the keyboard into array until enter is pressed. The scanf() places '\0' into array at the end of the input. The printf()

function prints the characters from the array on to monitor, leaving the end of the string '\0'. The *%s* used in the scanf() and printf() functions is a format specification for strings.

7.3 DISPLAY OF STRINGS USING DIFFERENT FORMATTING TECHNIQUES

The ***printf*** function with ***%s*** format is used to display the strings on the screen. For example, the below statement displays entire string:

printf("%s", name);

We can also specify the accuracy with which character array (string) is displayed. For example, if you want to display first 5 characters from a field width of 15 characters, you have to write as:

printf("%15.5s", name);

If you include minus sign in the format (e.g. % –10.5s), the string will be printed left justified.

printf("% -10.5s", name);

Example 7.2

Write a program to display the string "UNIX" in the following format.

```
U
UN
UNI
UNIX
UNIX
UNI
UN
U
```

```
/* Program to display the string in the above shown format*/

# include <stdio.h>
main()
{
int  x, y;
static char string[ ] = "UNIX";
printf("\n");
for( x=0; x<4; x++)
{
        y = x + 1;
  /* reserves 4 character of space on to the monitor and minus sign is for left
justified*/
        printf("%-4.*s \n", y, string);

 /* and for every loop the * is replaced by value of y */
/* y value starts with 1 and for every time it is incremented by 1 until it reaches to 4*/
}

for( x=3; x>=0; x- -)
        {
         y = x + 1;
         printf("%-4.*s \n", y, string);
```

```
/*  y value starts with 4 and for every time it is decrements by 1 until it reaches to 1*/
        }
}
```

OUTPUT

```
U
UN
UNI
UNIX
UNIX
UNI
UN
U
```

7.4 ARRAY OF STRINGS

Array of strings are multiple strings, stored in the form of table. Declaring array of strings is same as strings, except it will have additional dimension to store the number of strings. Syntax is as follows:

char array-name[size][size];

For example,

char names[5][10];

where names is the name of the character array and the constant in first square brackets will gives number of string we are going to store, and the value in second square bracket will gives the maximum length of the string.

Example 7.3

char names [3][10] = {“martin”, “phil”, “collins”};

It can be represented by a two-dimensional array of size[3][10] as shown below:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
m	a	r	t	i	n	\0			
p	h	i	l	\0					
c	o	l	l	i	n	s	\0		

Example 7.4

Write a program to initializes 3 names in an array of strings and display them on to monitor

```
/* Program that initializes 3 names in an array of strings and display them on to
monitor.*/

#include <stdio.h>
main()
{
        int   n;
        char   names[3][10] = {“Alex”, “Phillip”, “Collins” };
        for(n=0; n<3; n++)
        printf(“%s \n”,names[n] );    }
```

OUTPUT

Alex
Phillip
Collins

Check Your Progress 1

1. Which of the following is a static string?

   A. Static String;
   B. “Static String”;
   C. ‘Static String’;
   D. char string[100];

……

……

……

2. Which character ends all strings?

   A. ‘.’
   B. ‘ ‘
   C. ‘0’
   D. ‘n’

……

……

……

3. What is the Output of the following programs?

(a)
```
main()
{
    char name[10] = “IGNOU”;
    printf(“\n %c”, name[0]);
    printf(“\n %s”, name);
}
```

(b)
```
main()
{
  char  s[ ] = “hello”;
  int j = 0;
  while ( s[j] != ‘\0’ )
      printf(“ %c”,s[j++]);
}
```

(c)
```
main()
{
  char  str[ ] = “hello”;
  printf(“%10.2s”, str);
  printf(“%-10.2s”, str);
}
```

……

……

……

4 Write a program to read 'n' number of lines from the keyboard using a two-dimensional character array (ie., strings).

……

……

……

7.5 BUILT IN STRING FUNCTIONS AND APPLICATIONS

The header file <string.h> contains some string manipulation functions. The following is a list of the common string managing functions in C.

7.5.1 Strlen Function

The **strlen** function returns the length of a string. It takes the string name as argument. The syntax is as follows:

n = strlen (str);

where **str** is name of the string and **n** is the length of the string, returned by **strlen** function.

Example 7. 5

Write a program to read a string from the keyboard and to display the length of the string on to the monitor by using strlen( ) function.

```
/*  Program to illustrate the strlen function to determine the length of a string */

#include <stdio.h>
#include <string.h>
main()
{
char name[80];
int length;
printf("Enter your name: ");
gets(name);
length = strlen(name);
printf("Your name has %d characters\n", length);
}
```

OUTPUT

```
Enter your name: TYRAN
Your name has 5 characters
```

7.5.2 Strcpy Function

In C, you cannot simply assign one character array to another. You have to copy element by element. The string library <string.h> contains a function called **strcpy** for this purpose. The **strcpy** function is used to copy one string to another. The syntax is as follows:

strcpy(str1, str2);

where str1, str2 are two strings. The content of string str2 is copied on to string str1.

Example 7.6

Write a program to read a string from the keyboard and copy the string onto the second string and display the strings on to the monitor by using strcpy( ) function.

```
/* Program to illustrate strcpy function*/

#include <stdio.h>
#include <string.h>
main()
{
char first[80], second[80];
printf("Enter a string: ");
gets(first);
strcpy(second, first);
printf("\n First string is : %s, and second string is: %s\n", first, second);
}
```

OUTPUT

```
Enter a string: ADAMS
First string is: ADAMS, and second string is: ADAMS
```

7.5.3 Strcmp Function

The **strcmp** function in the string library function which compares two strings, character by character and stops comparison when there is a difference in the ASCII value or the end of any one string and returns ASCII difference of the characters that is integer. If the return value ***zero*** means the two strings are equal, a negative value means that first is less than second, and a positive value means first is greater than second**.** The syntax is as follows:

n = strcmp(str1, str2);
where **str1** and **str2** are two strings to be compared and **n** is returned value of differed characters.

Example 7.7

Write a program to compare two strings using string compare function.

```
/* The following program uses the strcmp function to compare two strings. */

#include <stdio.h>
#include <string.h>
main()
{
 char first[80], second[80];
 int value;
printf("Enter a string: ");
 gets(first);
 printf("Enter another string: ");
 gets(second);
 value = strcmp(first,second);
  if(value == 0)
       puts("The two strings are equal");
    else if(value < 0)
         puts("The first string is smaller ");
        else if(value > 0)
```

```
        puts("the first string is bigger");
}
```

OUTPUT

```
Enter a string: MOND
Enter another string: MOHANT
The first string is smaller
```

7.5.4 Strcat Function

The **strcat** function is used to join one string to another. It takes two strings as arguments; the characters of the second string will be appended to the first string. The syntax is as follows:

strcat(str1, str2);
where str1 and str2 are two string arguments, string *str2* is appended to string *str1*.

Example 7.8

Write a program to read two strings and append the second string to the first string.

```
/* Program for string concatenation*/

#include <stdio.h>
#include <string.h>
main()
{
char first[80], second[80];
printf("Enter a string:");
gets(first);
printf("Enter another string: ");
gets(second);
strcat(first, second);
printf("\nThe two strings joined together: %s\n", first);
}
```

OUTPUT

```
Enter a string: BOREX
Enter another string: BANKS
The two strings joined together: BOREX BANKS
```

7.5.5 Strlwr Function

The **strlwr** function converts upper case characters of string to lower case characters. The syntax is as follows:

strlwr(str1);
where str1 is string to be converted into lower case characters.

Example 7.9

Write a program to convert the string into lower case characters using in-built function.

```
/* Program that converts input string to lower case characters */

#include <stdio.h>
#include <string.h>
```

```
main()
{
char first[80];
printf("Enter a string: ");
gets(first);
printf("Lower case of the string is %s", strlwr(first));
}
```

OUTPUT

```
Enter a string: BROOKES
Lower case of the string is brookes
```

7.5.6 Strrev Function

The **strrev** funtion reverses the given string. The syntax is as follows:

strrev(str);
where string **str** will be reversed.

Example 7.9

Write a program to reverse a given string.

```
/* Program to reverse a given string */

#include <stdio.h>
#include <string.h>
main()
{
char first[80];
printf("Enter a string:");
gets(first);
printf("\n Reverse of the given string is :  %s ", strrev(first));
}
```

OUTPUT

```
Enter a string: ADANY
Reverse of the given string is:  YNADA
```

7.5.7 Strspn Function

The **strspn** function returns the position of the string, where first string mismatches with second string. The syntax is as follows:

n = strspn (first, second);
where **first** and **second** are two strings to be compared, **n** is the number of character from which first string does not match with second string.

Example 7.10

Write a program, which returns the position of the string from where first string does not match with second string.

```
/*Program which returns the position of the string from where first string does not
match with second string*/

#include <stdio.h>
#include <string.h>
main()
```

```
{
char first[80], second[80];
printf("Enter  first string: “);
gets(first);
printf(“\n Enter  second string: “);
gets(second);
printf(“\n After %d characters there is no match”,strspn(first, second));
}
```

OUTPUT

```
Enter first string: ALEXANDER
Enter second string: ALEXSMITH
After 4 characters there is no match
```

7.6 OTHER STRING FUNCTIONS

strncpy function

The **strncpy** function same as *strcpy*. It copies characters of one string to another string up to the specified length. The syntax is as follows:

strncpy(str1, str2, 10);
where str1 and str2 are two strings. The **10** characters of string **str2** are copied onto string **str1.**

stricmp function

The **stricmp** function is same as *strcmp*, except it compares two strings ignoring the case (lower and upper case). The syntax is as follows:

n = stricmp(str1, str2);

strncmp function

The **strncmp** function is same as *strcmp*, except it compares two strings up to a specified length. The syntax is as follows:

n = strncmp(str1, str2, 10);
where **10** characters of **str1** and **str2** are compared and **n** is returned value of differed characters.

strchr function

The **strchr** funtion takes two arguments (the string and the character whose address is to be specified) and returns the address of first occurrence of the character in the given string. The syntax is as follows:

cp = strchr (str, c);
where **str** is string and **c** is character and **cp** is character pointer.

strset function

The **strset** funtion replaces the string with the given character**.** It takes two arguments the string and the character. The syntax is as follows:

strset (first, ch);
where string **first** will be replaced by character **ch**.

strchr function

The **strchr** function takes two arguments (the string and the character whose address is to be specified) and returns the address of first occurrence of the character in the given string. The syntax is as follows:

cp = strchr (str, c);
where **str** is string and **c** is character and **cp** is character pointer.

strncat function

The **strncat** function is the same as *strcat*, except that it appends upto specified length. The syntax is as follows:

strncat(str1, str2,10);
where 10 character of the str2 string is added into str1 string.

strupr function

The strupr function converts lower case characters of the string to upper case characters. The syntax is as follows:

strupr(str1);
where str1 is string to be converted into upper case characters.

strstr function

The **strstr** function takes two arguments address of the string and second string as inputs. And returns the address from where the second string starts in the first string. The syntax is as follows:

cp = strstr (first, second);
where **first** and s**econd** are two strings, **cp** is character pointer.

Check Your Progress 2

1. Which of the following functions compares two strings?
   A. compare();
   B. stringcompare();
   C. cmp();
   D. strcmp();

……

……

……

2. Which of the following appends one string to the end of another?
   A. append();
   B. stringadd();
   C. strcat();
   D. stradd();

……

……

……

3. Write a program to concatenate two strings without using the *strcat()* function.

……

……

……

4. Write a program to find string length without using the *strlen()* function.

……

……

……

5. Write a program to convert lower case letters to upper case letters in a given string without using strupp().

……

……

……

7.7 SUMMARY

Strings are sequence of characters. Strings are to be null-terminated if you want to use them properly. Remember to take into account null-terminators when using dynamic memory allocation. The string.h library has many useful functions. Losing the ' **\0'** character can lead to some very considerable bugs. Make sure you copy \0 when you copy strings. If you create a new string, make sure you put \0 in it. And if you copy one string to another, make sure the receiving string is big enough to hold the source string, including \0. Finally, if you point a character pointer to some characters, make sure they end with \0.

String Functions	Its Use
strlen	Returns number of characters in string.
strlwr	Converts all the characters in the string into lower case characters
strcat	Adds one string at the end of another string
strcpy	Copies a string into another
strcmp	Compares two strings and returns zero if both are equal.
strdup	Duplicates a string
strchr	Finds the first occurrence of given character in a string
strstr	Finds the first occurrence of given string in another string
strset	Sets all the characters of string to given character or symbol
strrev	Reverse a string

7.8 SOLUTIONS / ANSWERS

Check Your Progress 1

1. B

2. C

3. (a) I
IGNOU

(b) hello

(c) hehe

Check Your Progress 2

1. D

2. C

3. /* Program to concatenate two strings without using the strcat() function*/

```
# include<string.h>
# include <stdio.h>
 main()
 {
  char str1[10];
  char str2[10];
  char output_str[20];
  int i=0, j=0, k=0;
 printf(" Input the first string: ");
           gets(str1);
           printf("\nInput the second string: ");
           gets(str2);
          while(str1[i] != '\0')
  output_str[k++] = str1[i++];
 while(str2[j] != '\0')
  output_str[k++] = str2[j++];
 output_str[k] = '\0';
          puts(output_str);
 }
```

4. /* Program to find the string length without using the strlen() funtion */

```
# include<stdio.h>
# include<string.h>
main()
{
char  string[60];
int len=0, i=0;
printf(" Input the string : ");
gets(string);
while(string[i++] != '\0')
              len ++;
printf("Length of Input String = %d", len);
getchar();
}
```

5. /* Program to convert the lower case letters to upper case in a given string without using strupp() function*/

```
#include<stdio.h>
main()
{
int i= 0;   char source[10], destination[10];
gets(source);
while( source[i] != '\0')
{
  if((source[i]>=97) && (source[i]<=122))
```

```
            destination[i]=source[i]-32;
        else
            destination[i]=source[i];
    i++;
    }
    destination[i]= ' \0 ';
    puts(destination);
}
```

7.9 FURTHER READINGS

1. The C programming language, *Brain W. Kernighan, Dennis M. Ritchie*, PHI.
2. Programming with ANSI and Turbo C, *Ashok N. Kamthane*, Pearson Education, 2002.
3. Computer Programming in C, *Raja Raman. V*, 2002, PHI.
4. C,The Complete Reference, Fourth Edition, *Herbert Schildt*, Tata McGraw Hill, 2002.
5. Computer Science A structured Programming Approach Using C, *Behrouz A. Forouzan, Richard F. Gilberg*, Brooks/Cole Thomas Learning, Second Edition, 2001.