Programming Fundamentals (CT-175)
Lab 08

Strings in C - Using string library functions

Objectives

The objective of this lab is to implementing strings in C and to utilize them in an efficient and suitable manner.

Tools Required

DevC++ IDE

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Strings in C

In C, a string is not a formal data type; instead, it is an array of type char. Most languages internally treat strings as character arrays, but somehow conceal this fact from the programmer. Character arrays or strings are used by programming languages to manipulate text as words and sentences.

A string is one dimensional array of characters terminated by a null character ('\0') which has a numerical value of 0.

Declaration and initialization

A string can be declared as follow.

```
char string [8];
```

This will occupy eighty bytes in memory for the string. C concedes the fact that you would use strings very often and hence provides a shortcut for initializing strings. For Example,

```
char string [] = {'H', 'E', 'L', 'L', 'O', '\0',}
```

Each character in the array occupies one byte of memory and the last character is always '\0'. Ascii value of '\0' is 0 whereas value of '0' is 48.
The above string can also be initialized as
char [] = "HELLO";

Note that in this declaration '\0' is not necessary. C inserts the null character automatically.

Example:

```
int main (void)
        {
        char str[80] = "anonymous";
        printf("%s", str);                  //this will print HELLO
        printf("%c",str[2]);                //this will print the L
        printf ("%c",str[4]);               //this will print O
        }
```

Traversing String

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which can be done by traversing the text. Traversing string is somewhat different from the traversing an integer array. We need to know the length of the array to traverse an integer array, whereas we may use the null character in the case of string to identify the end the string and terminate the loop.
Hence, there are two ways to traverse a string.

- By using the length of string
- By using the null character.

Using the length of string

Let's see an example of counting the number of vowels in a string.

```
#include<stdio.h>
void main ()
{
   char s[11] = "hello world";
   int i = 0;
   int count = 0;
   while(i<11)
   {
      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
                        {
                           count ++;
                        }
                        i++;
                     }
                     printf("The number of vowels %d",count);
                  }
```

Let's see the same example of counting the number of vowels by using the null character.

```
#include<stdio.h>
void main ()
{
   char s[11] = "javatpoint";
   int i = 0;
   int count = 0;
   while(s[i] != NULL)
   {
      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
      {
         count ++;
      }
      i++;
   }
   printf("The number of vowels %d",count);
}
```

Input a String in C

Scanf can also be used in the case of strings but with a different scenario. Consider the below code which stores the string while space is encountered.

```
#include<stdio.h>
void main ()
{
   char s[20];
   printf("Enter the string?");
   scanf("%s",s);
   printf("You entered %s",s);
}
```

Above code will not work for space separated strings. To make this code working for the space separated strings, the minor changed required in the scanf function, i.e., instead of writing scanf("%s",s), we must write: scanf("%[^\n]s",s) which instructs the compiler to

store the string s while the new line (\n) is encountered. Let's consider the following example to store the space-separated strings.

```
#include<stdio.h>
void main ()
{
   char s[20];
   printf("Enter the string?");
   scanf("%[^\n]s",s);
   printf("You entered %s",s);
}
```

We do not need to use address of (&) operator in scanf to store a string since string s is an array of characters and the name of the array, i.e., s indicates the base address of the string (character array) therefore we need not use & with it.

C gets() and puts() functions

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

C gets() function

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

```
#include<stdio.h>
void main ()
{
   char s[30];
   printf("Enter the string? ");
   gets(s);
   printf("You entered %s",s);
}
```

The gets() function is risky to use since it doesn't perform any array bound checking and keep reading the characters until the new line (enter) is encountered. It suffers from buffer overflow, which can be avoided by using fgets(). The fgets() makes sure that not more than the maximum limit of characters are read. Consider the following example.

```
#include<stdio.h>
void main()
{
  char str[20];
  printf("Enter the string? ");
  fgets(str, 20, stdin);
  printf("%s", str);
}
```

C puts() function

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string,

which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

```
#include<stdio.h>
#include <string.h>
int main(){
char name[50];
printf("Enter your name: ");
gets(name); //reads string from user
printf("Your name is: ");
puts(name);  //displays string
return 0;
}
```

String Library Function: (use header <string.h>)

1. strlen( )

Purpose: use length to find the length of a string, not counting the terminating null character.

Returns: returns the length of string.

strlen (string);

Example:

```
int main (void)
{
char string [50];
int len;

printf("Enter the string:");
gets(string);
len = strlen(string);
printf("%d",len);
}
```

2. strcpy( )

Purpose: use strcpy() to copy one string to another.

Returns: The strcpy() function returns a pointer to the copied string (i.e., it returns string.)

char strcpy (char dest , const char src)

Example:

```
int main(void)  {
   char string [ 10 ] ;
   char str2 [ ] = "Pakistan";
   printf ("%s \n",  str2 );
   strcpy ( string , str2 ) ;
   printf ("%s \n",  string)  ;
 }
```

Exercise

1. As a programmer, you are required to create a program that takes the first and last name from a user. The program then combines both the inputs taken and prints the string backwards.
2. Each student is required to find out the maximum frequency of characters occurring in their name and the courses offered in Fall 2021. To find it, the student enters their name, courses offered and the program finds the maximum occurrences of a character in the name and course. Course names should be used like Programming Fundamentals, Applied Physics, Pakistan Studies and so on.
3. Students are grouped in two to complete a lab task. Each student is required to enter a string of their own choice as an input to the program. The program will then display as a result whether both the strings are equal. If the strings are not equal, the program will display which of the string is greater.

   Test cases:

   Enter two strings that are same.

   Enter two different strings.
4. Write down the output of the following program.

```
int main (void)
{
char a[11] = "hello world";
int i;
for(i = 0; i <= 9; i++)
        {
        a[i] = a[i + 1];
        printf("%d  \t  %s  \n", i, a);
        }
Printf("\n %d",  a);}
```

Lab 08 Evaluation		
Student Name:	**Student ID:**	**Date:**
Task No.	**Marks**	**Remarks by teacher in accordance with the rubrics**
1		
2		
3		
4		
5		