Strings

Definition: A string constant is a one-dimensional array of characters terminated by a null (‘\0’).

Example: char name[ ] = {‘D’, 'A', 'D', 'A' , '\0' } ;

name	D	A	D	A	\0
	100	101	102	103	104

Note:

- Each character in the array occupies one byte of memory and the last character is always ‘\0’which is called null character.
- Note that ‘\0’ and ‘0’ are not same. ASCII value of ‘\0’ is 0, whereas ASCII value of ‘0’ is 48.
- The terminating null (‘\0’) is important, because it is the only way the functions that work with a string can know where the string ends. In fact, a string not terminated by a ‘\0’ is not really a string, but merely a collection of characters.
- While initializing ‘\0’ is not necessary, C inserts the null character automatically. Ex: char name[ ] = "GANGULY" ;
- The **%s** is used as a format specification for printing out a string. The same specification can be used to receive a string from the keyboard.
- The **scanf( )** function fills in the characters typed at keyboard into this array until the enter key is hit. Once enter is hit, **scanf( )** places a ‘\0’ in the array. Naturally, we should pass the base address of the array to the **scanf( )** function.
- **scanf( )** is not capable of receiving multi-word strings.
- The standard library functions **gets()** and **puts()** are generally used with strings. The **gets( )** can receive a multi-word string. The **puts( )** can display only one string at a time. Also, on displaying a string, unlike **printf( )**, **puts( )** places the cursor on the next line.

Standard library string functions: The most commonly used string handling library functions are **strlen( )**, **strcpy( )**, **strcat( )** and **strcmp( )**. Let us study these functions one by one.

1. strlen( ) : This function counts the number of characters present in a string. Its usage is illustrated in the following program-

```
void main( )
 {
  char name[ ] = "Sachin" ;
  int length1, length2 ;
  length1= strlen (name) ;
  length2=strlen(“Sachin”);
  printf(“%d %d”, length1, length2);
 }
```

Output: 6 6

Note: In the first call to the function **strlen( )**, we are passing the base address of the string, and the function in turn returns the length of the string. While calculating the length it doesn't count '\0'. Even in the second call we are passing the address of the string and not the string itself.

❖ The personalized **strlen( )** function **mystrlen( ):**

```
void main( )
 {
  char name[ ] = "Sachin" ;
  int len1, len2 ;
  len1 = mystrlen (name) ;
  len2 = mystrlen ( "Sachin" ) ;
  printf ( "\n%d %d", len1,len2) ;
 }

 mystrlen ( char *str )
 {
   int length = 0 ;
   while ( *str != '\0' )
    {
      length++ ;
      str++ ;
     }
    return ( length ) ;
 }
```

Output: 6 6

Note: The function **mystrlen( )** is fairly simple. All that it does is keep counting the characters till the end of string is not met. Or in other words keep counting characters till the pointer s doesn't point to '\0'.

2. strcpy( ): This function copies the contents of one string into another. The base addresses of the source and target strings should be supplied to this function. Consider the following example:

```
void main( )
 {
   char source_str[ ] = "Sourav" ;
   char target_str[20] ;
   strcpy ( target_str, source_str) ;
   printf ( "\nsource string = %s", source_str) ;
   printf ( "\ntarget string = %s", target_str) ;
  }
```

Output:

```
source string = Sourav
target string = Sourav
```

Note: On supplying the base addresses, **strcpy( )** goes on copying the characters in source string into the target string till it doesn't encounter the end of source string ('\0').

❖ The personalized my**strcpy( )** function **mystrcpy( ):**

```
void main( )
 {
  char source_str[ ] = "Sourav" ;
  char target_str[20] ;
  mystrcpy ( target_str, source_str) ;
  printf ( "\nsource string = %s", source_str) ;
  printf ( "\ntarget string = %s", target_str) ;
 }

 mystrcpy ( char *t, char *s )
 {
  while ( *s != '\0' )
    {
     *t = *s ;
      s++ ;
      t++ ;
    }
   *t = '\0' ;
 }
```

Output:

```
source string = Sourav
target string = Sourav
```

Note: Having copied the entire source string into the target string, it is necessary to place a '\0' into the target string, to mark its end.

3. strcat( ) : This function concatenates the source string at the end of the target string. Consider the following example-

```
void main( )
 {
  char source_str[ ] = "Students!" ;
  char target_str[30] = "Hello" ;
  strcat ( target_str, source_str) ;
  printf ( "\nsource string = %s", source_str) ;
  printf ( "\ntarget string = %s", target_str) ;
 }
```

Output:

```
source string = students!
target string = Hellostudents!
```

4. strcmp( ) : This is a function which compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first. If the two strings are identical, **strcmp( )** returns a value zero. If they're not, it returns the numeric difference between the ASCII values of the first non-matching pairs of characters. Here is a program which puts **strcmp( )** in action.

```
void main( )
 {
   char string1[ ] = "Jerry" ;
   char string2[ ] = "Ferry" ;
   int i, j, k ;
   i = strcmp ( string1, "Jerry" ) ;
   j = strcmp ( string1, string2 ) ;
   k = strcmp ( string1, "Jerry boy" ) ;
   printf ( "\n%d %d %d", i, j, k ) ;
}
```

Output:

0 4 -32

Note:

- In the first call to **strcmp( )**, the two strings are identical—"Jerry" and "Jerry"—and the value returned by **strcmp( )** is zero.
- In the second call, the first character of "Jerry" doesn't match with the first character of "Ferry" and the result is 4, which is the numeric difference between ASCII value of 'J' and ASCII value of 'F'.
- In the third call to **strcmp( )** "Jerry" doesn't match with "Jerry boy", because the null character at the end of "Jerry" doesn't match the blank in "Jerry boy". The value returned is -32, which is the value of null character minus the ASCII value of space, i.e., '\0' minus ' ', which is equal to -32.
- The exact value of mismatch will rarely concern us. All we usually want to know is whether or not the first string is alphabetically before the second string. If it is, a negative value is returned; if it isn't, a positive value is returned. Any non-zero value means there is a mismatch.

Problem set-1

1. Write a program that converts all lowercase characters in a given string to its equivalent uppercase character.
2. Write a program to count the number of vowels, consonants, digits and special symbols of a string.
3. Write a program that converts a string like "124" to an integer 124.
4. Write a program that replaces two or more consecutive blanks in a string by a single blank.