C Strings

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

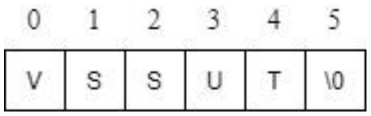
There are two ways to declare a string in c language.

1. By char array
2. By string literal

Let's see the example of declaring **string by char array** in C language.

```
char ch[6]={'V', 'S', 'S', 'U', 'T', '\0'};
```

As we know, array index starts from 0, so it will be represented as in the figure given below.



While declaring string, size is not mandatory. So we can write the above code as given below:

```
char ch[ ]={'V', 'S', 'S', 'U', 'T', '\0'};
```

We can also define the **string by the string literal** in C language. For example:

```
char ch[]="VSSUT";
```

In such case, '\0' will be appended at the end of the string by the compiler.

Difference between char array and string literal

There are two main differences between char array and literal.

- We need to add the null character '\0' at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the character array.
- The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

String Example in C

Let's see a simple example where a string is declared and being printed. The '%s' is used as a format specifier for the string in c language.

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

```
#include <string.h>
int main(){
  char ch[6]={'V', 'S', 'S', 'U', 'T', '\0'};
   char ch2[6]="VSSUT";

   printf("Char Array Value is: %s\n", ch);
   printf("String Literal Value is: %s\n", ch2);
 return 0;
}
```

OUTPUT

```
Char Array Value is: VSSUT
String Literal Value is: VSSUT
```

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.

Let's discuss each one of them.

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[6] = "vssut";
    int i = 0;
    int count = 0;
    while(i<6)
    {
        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);
}
```

OUTPUT

```
The number of vowels 1
```

Using the null character

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

```
#include<stdio.h>
void main ()
{
    char s[6] = "vssut";
    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);
}
```

OUTPUT

```
The number of vowels 1
```

Accepting string as the input

Till now, we have used scanf to accept the input from the user. However, it 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);
}
```

OUTPUT

```
Enter the string?VSSUT is the best
You entered VSSUT
```

It is clear from the output that, the 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);
}
```

OUTPUT

```
Enter the string?VSSUT is the best
You entered VSSUT is the best
```

Here we must also notice that 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.

Some important points

However, there are the following points which must be noticed while entering the strings by using scanf.

- The compiler doesn't perform bounds checking on the character array. Hence, there can be a case where the length of the string can exceed the dimension of the character array which may always overwrite some important data.
- Instead of using scanf, we may use gets() which is an inbuilt function defined in a header file string.h. The gets() is capable of receiving only one string at a time.

Pointers with strings

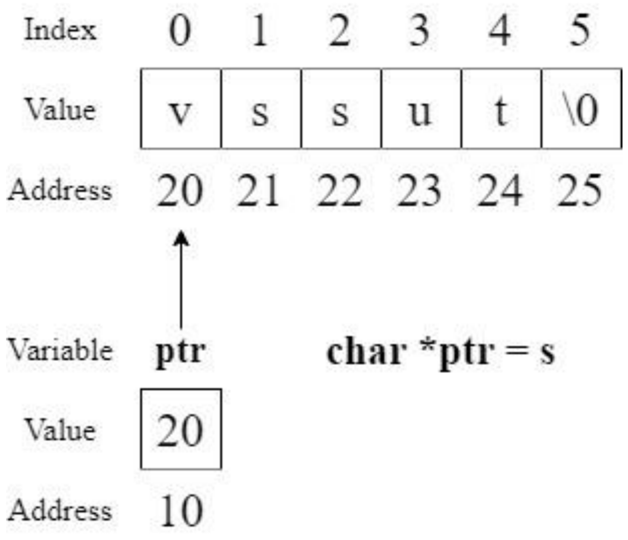
We have used pointers with the array, functions, and primitive data types so far. However, pointers can be used to point to the strings. There are various advantages of using pointers to point strings. Let us consider the following example to access the string via the pointer.

```
#include<stdio.h>
void main ()
{
   char s[6] = "vssut";
   char *p = s; // pointer p is pointing to string s.
   printf("%s",p); // the string vssut is printed if we print p.
```

}

OUTPUT

vssut



As we know that string is an array of characters, the pointers can be used in the same way they were used with arrays. In the above example, p is declared as a pointer to the array of characters s. P affects similar to s since s is the base address of the string and treated as a pointer internally. However, we can not change the content of s or copy the content of s into another string directly. For this purpose, we need to use the pointers to store the strings. In the following example, we have shown the use of pointers to copy the content of a string into another.

```
#include<stdio.h>
void main ()
{
   char *p = "hello vssut";
   printf("String p: %s\n",p);
   char *q;
   printf("copying the content of p into q...\n");
   q = p;
   printf("String q: %s\n",q);
}
```

OUTPUT

```
String p: hello vssut
copying the content of p into q...
String q: hello vssut
```

Once a string is defined, it cannot be reassigned to another set of characters. However, using pointers, we can assign the set of characters to the string. Consider the following example.

```
#include<stdio.h>
void main ()
{
   char *p = "hello vssut";
   printf("Before assigning: %s\n",p);
   p = "hello";
   printf("After assigning: %s\n",p);
}
```

OUTPUT

```
Before assigning: hello vssut
After assigning: hello
```

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.

Declaration

```
char[] gets(char[]);
```

Reading string using gets()

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

OUTPUT

```
Enter the string?
vssut is the best
You entered vssut is the best
```

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, 10, stdin);
  printf("%s", str);
}
```

OUTPUT

```
Enter the string? vssut is the best
vssut is
```

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.

Declaration

```
int puts(char[])
```

Let's see an example to read a string using gets() and print it on the console using puts().

```
#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;

}

OUTPUT

```
Enter your name: D C RAO
Your name is: D C RAO
```