UNIT-3/STRING

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.

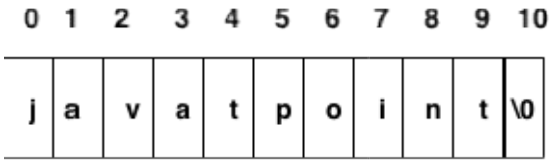
By char array

By string literal

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

```
char ch[10]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', '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[]={'h', 'o', 'n', 'e', 'y',  '\0'};
```

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

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

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

An array of characters (or) collection of characters is called a string.

Declaration

Refer to the declaration given below –

char stringname [size];

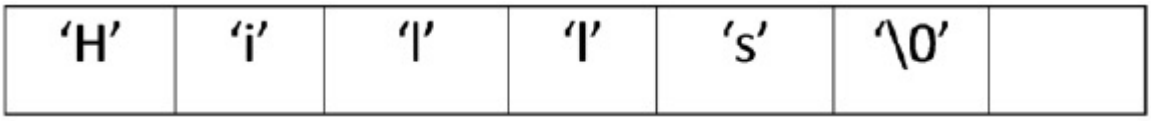
For example - char a[50]; a string of length 50 characters.

Initialization

The initialization is as follows –

Using **single character** constant –

char string[20] = { 'H', 'i', 'l', 'l', 's' ,'\0'}



Using string constants –

char string[20] = "Hello":;

'\0' is called a null character. It marks the end of the string.

'\0' is automatically placed by the compiler, if a string is given as input. The user has to take care of placing '\0' at the end if a single character is given.

Accessing – There is a control string "%s" used for accessing the string, till it encounters '\0'.

Example

Following is the C program for a string –

```
#include<stdio.h>
main ( ){
   char a[10] = "Hello";
   clrscr ( );
   printf ( " given string is %s",a)
   getch ( );
}
```

Output:

When the above program is executed, it produces the following result –

Given string is Hello

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[11]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
   char ch2[11]="javatpoint";
    printf("Char Array Value is: %s\n", ch);
   printf("String Literal Value is: %s\n", ch2);
 return 0;
}
```

Output

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

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

Output:

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:

C String Functions

There are many important string functions defined in "string.h" library.

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

C String Length: strlen() function

The strlen() function returns the length of the given string. It doesn't count null character '\0'.

```
#include<stdio.h>
#include <string.h>
int main(){
char ch[20]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
  printf("Length of string is: %d",strlen(ch));
 return 0;
}
```

Output:

```
Length of string is: 10
```

C Copy String: strcpy()

The strcpy(destination, source) function copies the source string in destination.

```
#include<stdio.h>
#include <string.h>
int main(){
 char ch[20]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
   char ch2[20];
   strcpy(ch2,ch);
   printf("Value of second string is: %s",ch2);
 return 0;
}
```

Output:

C String Concatenation: strcat()

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

```
#include<stdio.h>

#include <string.h>

int main(){

  char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};

   char ch2[10]={'c', '\0'};

   strcat(ch,ch2);

   printf("Value of first string is: %s",ch);

 return 0;

}
```

Output:

```
Value of first string is: helloc
```

C Compare String: strcmp()

The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Here, we are using *gets()* function which reads string from the console.

```
#include<stdio.h>

#include <string.h>

int main(){

  char str1[20],str2[20];
```

```
 printf("Enter 1st string: ");
 gets(str1);//reads string from console
 printf("Enter 2nd string: ");
 gets(str2);
 if(strcmp(str1,str2)==0)
    printf("Strings are equal");
 else
    printf("Strings are not equal");
return 0;
}
```

Output:

```
Enter 1st string: hello
Enter 2nd string: hello
Strings are equal
```

Implementation without using standard library functions:

Program 1: Print the reverse of a string without using the library function

Let's consider an example to print the reverse of a string using user defined function.

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

```
  // function definition of the revstr()
void revstr(char *str1)
{  // declare variable
    int i, len, temp;
    len = strlen(str1); // use strlen() to get the length of str string
      // use for loop to iterate the string
    for (i = 0; i < len/2; i++)
    {
        // temp variable use to temporary hold the string
        temp = str1[i];
        str1[i] = str1[len - i - 1];
        str1[len - i - 1] = temp;
    }
}
     int main()
    {
        char str[50]; // size of char string
        printf (" Enter the string: ");
        gets(str); // use gets() function to take string
          printf (" \n Before reversing the string: %s \n", str);
           // call revstr() function
        revstr(str);
```

```
        printf (" After reversing the string: %s", str);    }
```

Output:

Program 2 : Check Whether Entered Character is Uppercase Letter or Not Without using Library Function.

Way 1 :

```
#include<stdio.h>

int main() {

char ch;

  printf("\nEnter The
Character : ");

 scanf("%c", &ch);

if (ch >= 'A' && ch <= 'Z')

  printf("Character is
Upper Case Letters");

 else

  printf("Character is
Not Upper Case Letters");

return (0);

}
```

Way 2 :

```
#include<stdio.h>
 int main() {
   char ch;
printf("\nEnter The Character : ");
scanf("%c", &ch);
 if (ch >= 65 && ch <= 90)
     printf("Character is Upper Case Letters");
   else
     printf("Character is Not Upper Case Letters");
 return (0);
}
```

Output :

1 Enter The Character : A

2 Character is Uppercase Letters

Program 3. Write a program to reverse a string:

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

```
 int main() {
   char str[100], temp;
   int i, j = 0;
 printf("\nEnter the
string:");
   gets(str);
  i = 0;
   j = strlen(str) - 1;
 while (i < j) {
      temp = str[i];
      str[i] = str[j];
      str[j] = temp;
      i++;
      j--;
   }
  printf("\nReverse
string is :%s", str);
   return (0);
}
```

Output :

1 Enter the string : Pritesh

2 Reverse string is : hsetirP

Program 4. C Program to Compare Two Strings Without Using Library Function.

```
#include<stdio.h>

 int main() {

   char str1[30], str2[30];

   int i;

 printf("\nEnter two strings :");

gets(str1);

   gets(str2);

  i = 0;

   while (str1[i] == str2[i] && str1[i] != '\0')

      i++;

   if (str1[i] > str2[i])

      printf("str1 > str2");

   else if (str1[i] < str2[i])

      printf("str1 < str2");

   else

      printf("str1 = str2");

 return (0);

}
```