Strings in C

In this tutorial, you will master everything about strings in the C programming language. Also, you will learn how to declare and initialize a string, various ways to input and output strings, etc with the help of user-friendly examples. Also will cover some of the common library functions used to manipulate strings in C.

String Constants

Unlike other programming languages, in C string is not an elementary data type, rather a collection of adjacent characters. We can generate it by character array to use words and sentences in programs. As it is an array of characters, there includes a null character at the end of every word to indicate the termination of space. In C programming, strings are written inside double quotation (" ") like:

- "Computer "
- "Programming in C"
- " "

A visual representation of string is as follows:

This is how the compiler takes a sequence of characters bounded in double-quotes and by default at the end will append a null character '\0.

How to declare a string variable

We can use any proper variable name as the name of the string variable and declare it as an array.

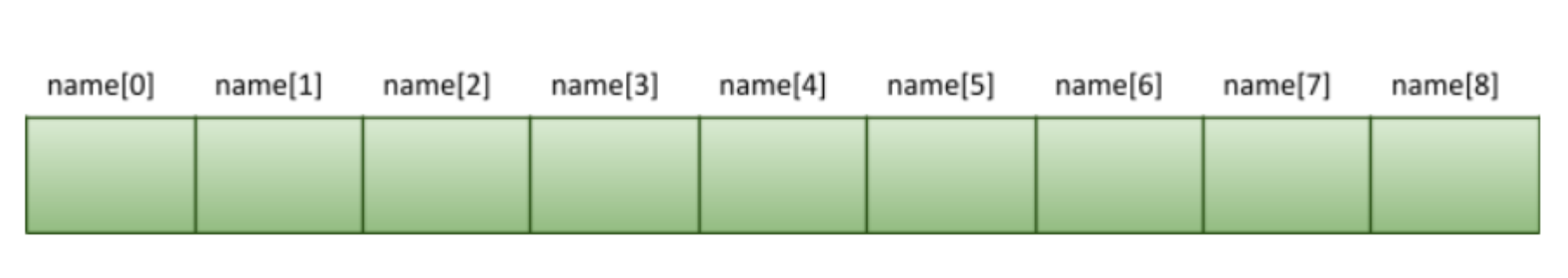
The prototype of the string declaration is:

```
char string_name [SIZE];
```

We can calculate the number of maximum characters inside the string by 'SIZE'. If we store any variable in the string, the compiler adds a null ('\0') character at the end of any string. So 'SIZE' will display the number, one more than the actual number of characters. If we declare,

```
char name[9];
```

the compiler will allow us to place 8 characters in the declared strings 'name' following a null character. The string will be allocated as

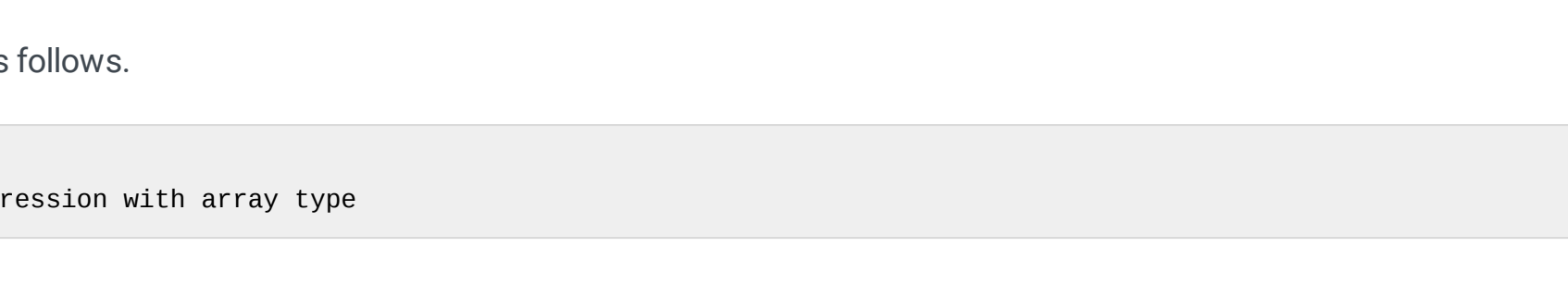


How to initialize a string variable in C

we can determine the value of the string variable during its declaration. There are different ways to initialize the string variables.

```
char city[] = “Madrid”;
char city[7] = “Madrid”;
char city[7] = {‘M’, ‘a’, ‘d’, ‘r’, ‘i’,  ‘d’};
```

Here "MADRID" will be stored as



ASSIGNING VALUES TO STRING VARIABLES

Since string belongs to an array type, it is not possible to assign values to a string variable separately after declaration as shown below program.

```
#include<stdio.h>
main()
{
    char c[10];
    c = "Learn C";
    printf("%s",c);
}
```

This example raises an error as follows.

```
error: assignment to expression with array type
```

In such cases, the string function `strcpy()` can be used.

How to traverse over a string

In most programming languages, string traversal is an important aspect. By traversing it means the compiler allows us to determine the value of each array element inside a string. While traversing a string, there is no need to know the size of the string before as the null character indicates the end of the sequence. If the null character is not present at the end of the string, it will be treated as a mere collection of characters. It is because the compiler can not understand where the string ends.

From the following example. we can find-

```
#include <stdio.h>
main()
{
       char name [ ] = "Learn eTutorials";
        int i =0,count=0;
        while (name [i] != '\0')
         {
            if(name[i]=='a' || name[i] == 'e' || name[i] == 'i' || name[i] == 'u' || name[i] == 'o')
          {
            count ++;
          }
               printf ("%c ", name [i]);
                i++;
        }
        printf("\nNumber of Vowels in string %s is :%d",name,count);
}
```

Output:

```
L e a r n   e T u t o r i a l s
Number of Vowels in string Learn eTutorials is :7
```

If we run the program, it will print "**Learn eTutorials**". Evidently, the while loop continues printing each character of the array element, till it encounters a null character. Meanwhile, it counts the number of vowels in the given string.

Here `%c` is the format specifier used to print the string characters one by one while `%s` is the format specifier used to print the string literal.

How to manipulate strings using pointers

From our previous tutorial, we are well educated about pointers in C. Now we can manipulate strings using pointers too. As strings are an array of characters, pointers on string works in the same manner it does on arrays. It works by storing the address of the array (first element's) in the pointer.

Here is an example program.

```
#include <stdio.h>

int main() {
  char name[] = "Steve Jobs";

  printf("%c\n", *name);
  printf("%c\n", *(name+2));
  printf("%c\n", *(name+6));

  char *Ptr;
  Ptr = name;
  printf("\n%s", Ptr);
}
```

Output:

```
S
e
J

Steve Jobs
```

String I/O Functions

There is a set of I/O functions in C to access the input from the keyboard and display it on the screen as per requirement. Mostly used output functions are: printf(), puts(), putchar()and input functions are scanf(), gets(), getchar(), getch(), getche().

Read and write string using scanf and printf

Throughout the series of our tutorials, you have seen the use of `scanf` to read the input from the keyboard and printf to write the output. No surprise that `scanf` and `printf` can be used to access strings also. But the only difference is that while using `scanf` you can only read the string till it encounters whitespace like newline, tab, or space.

Here is the example that demonstrates the functioning of `scanf` and `printf` on string:

```
#include<stdio.h>
int main()
{

   char name[10];

   printf("Enter your full name:");
   scanf("%s",name);

   printf("Full Name :%s",name);

}
```

Output:

```
Enter your full name:Chris Jake
Full Name :Chris
```

From the output, it is evident that even though **Chris Jake** was given as the input, `scanf` only reads the first string Chris as it encounters whitespace thereafter. Here `%s` is used as format specifiers to use `scanf` and `printf`.

Read and write string using gets() and puts()

In C, we can read and write strings using `gets()` and `puts()` function. The following example is a simple demonstration of the use of `gets()` and `puts()` function.

```
#include <stdio.h>
int main()
{
       char nm[40];
       puts ("Type anything you wish:");
       gets(nm);
       puts(nm);

}
```

Output:

```
Type anything you wish:
Welcome to Learn eTutorials... Let's learn C
Welcome to Learn eTutorials... Let's learn C
```

Here it is clearly visible that `gets()` reads and stores whatever we enter through the keyboard irrespective of its size. This may lead to a buffer overflow. So as to prevent this C has developed another function called `fgets()` in which we need to specify the size limit.

```
#include <stdio.h>
int main()
{
       char nm[40];
       puts ("Type anything you wish:");
       fgets(nm,sizeof(nm),stdin);
       puts(nm);

}
```

How to pass a string as an argument to a function

Like arrays, we can also pass strings to a function. We can pass a string to a function as an argument with pointers or without pointers. Below shown examples illustrate these two scenarios.

Passing string without pointers

```
#include <stdio.h>
void printString(char str[]);

int main()
{
    char str[50];
    printf("Enter desired string: ");
    fgets(str, sizeof(str), stdin);
    printString(str);
    return 0;
}
void printString(char str[])
{
    printf("String printed as: ");
    puts(str);
}
```

Output:

```
Enter desired string: Welcome to Learn eTutorials
String printed as: Welcome to Learn eTutorials
```

In this example, since str is a character array the **printString** function expects a string array as its arguments hence we passed string as str[] and also used `fgets` and `puts` function to read and write the string.

Passing string using Pointers

This could be a simple and error-free way of passing a string as an argument to a function. The above code snippet will be modified as follows :

```
#include <stdio.h>
void printString(char *strptr);

int main()
{
    char *strptr[50];
    printf("Enter desired string: ");
    fgets(strptr, sizeof(strptr), stdin);
    printString(strptr);
    return 0;
}
void printString(char *strptr)
{
    printf("String printed as: ");
    puts(strptr);
}
```

Output:

```
Enter desired string: Welcome to Learn eTutorials
String printed as: Welcome to Learn eTutorials
```

Here we have declared a character pointer so the printString() function expects the pointer variable as its argument. That means we are passing the address of the string (**strptr**) to the function, consequently any changes made to the string in function will reflect back.

String Library Functions

String functions are an inevitable part of a programming language as it makes your string manipulation easier just by simply calling the specific function and implementing them wherever you need it in your piece of code. String manipulation in the sense of copying a string or concatenating string or any other functions can be easily invoked with these predefined functions.

Some of the most often used string functions are given in the table below and its detail as follows:

Function	Work of Function
strlen()	Determines the length of string
strcpy()	copies a string to another
strcat()	concatenates(joins) two strings
strcmp()	compares two strings
strlwr()	converts string to lowercase
strupr()	converts string to uppercase

1. Strlen()

As its name indicates, `strlen()` computes the length of the string without including the null character or end character(\0).

```
#include<stdio.h>
#include<string.h>
void main()
{
    char str[20]="Learn eTutorials";
    printf("Enter the length of string:%d\n",strlen(str));
    printf("Enter the size of string: %d\n",sizeof(str));
}
```

Output:

```
Enter the length of string:16
Enter the size of string: 20
```

From the example, you can evidently observe the difference between `strlen` and `sizeof` operator. `Strlen` determines the length of the string while the `sizeof` operator returns the size of the string(total allocated space).

2. Strcpy()

This function is used to copy one string to another. So `strcpy()` takes two arguments string 1 and string 2 and copies string 2 to string 1 as shown in the example below.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[30]="Learn eTutorials";
    char s2[30]="C programming Language ";
    printf("String copied to S1 is :%s\n",strcpy(s1,s2));

}
```

Output:

```
String copied to S1 is :C programming Language
```

3. Strcat()

Like `strcpy()`, this function also takes two arguments and returns the concatenated strings. The following example shows the concatenation of two strings.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[30]="C Programming";
    char s2[30]=" Language ";
    printf("Concatenated String is :%s\n",strcat(s1,s2));

}
```

Output:

```
Concatenated String is :C Programming Language
```

4. Strcmp()

`strcmp()` function compares two strings and returns an integer value.

- If s1==s2 , it returns zero.
- If s1<s2, it returns a negative integer
- If s1>s2, it returns a positive integer.

```
#include<stdio.h>
#include<string.h>
void main()
{
    char s1[30]="Language";
    char s2[30]="Language";
    if (strcmp(s1,s2)==0)
    {
        printf("Strings are equal");
    }
```