Tutorial Link https://codequotient.com/tutorials/Strings/5c3b3e46e9022607facc8da1

TUTORIAL

Strings

Chapter

1. Strings

Topics

Strings are one-dimensional array of characters terminated by NULL character (represented by '\0') . C does not have a string data type, we have to declare a character array to do so. For example,

```
char str[10];
```

We can initialize the string at the time of declaration just like other arrays or later we can take inputs from user to assign values to string variables. To initialize a string we can use:

```
char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};
```

or

```
char str[6]=”hello”;
```

both these formats will create and initialize a string str of size 6 with the word “hello”. We can use them like other array variables in our programs, for example: -

```C
#include <stdio.h>
int main ()
{
   char str[14] = {'C', 'o', 'd', 'e', ' ', 'Q', 'u', 'o', 't', 'i', 'e', 'n', 't', '\0'};
   printf("string value is: %s\n", str);               // use %s to input/output a string
}

```

Strings and functions

As strings are just a character array, we can pass strings just like other arrays to the functions and can return string from a function also. Following is an example in which a string is passed to a function and the function print the string and also return a pointer to a new string to calling function: -

```C
#include<stdio.h>

char* print_String(char str[])
{
  char str1[]="Code Quotient";
```

```
  char *b=str1;
  printf("Input String is : %s\n",str);
  return b;
}

int main()
{
  char str[] = "This is String";
  char *result;
  result=print_String(str);
  printf("String Received is : %s\n",result);
  return 0;
}
```

Strings and pointers

We can use a pointer to handle the string in same manner as we do in case of integer array. We can store the base address of string in a pointer to character variable. We then can traverse the string character by character in the same manner. For example,

```c
#include<stdio.h>

void print_String(char *str)
{
  int i=0;
  while(str[i] != '\0')  {              // str[i] is equivalent to *(str+i)
    printf("%c", str[i]);
    i++;
  }
}

int main()
{
  char str[] = "CodeQuotient - Get better at coding";
  print_String(str);
  return 0;
}
```

we can also use the pointer directly on the string, instead of using it with the array indices.

```c
void print_String(char *str)
{
  while(*str != '\0')
  {
    printf("%c", *str);
    str++;
  }
}


```

The difference between above two approaches is that in first, str still points to the beginning of the string, while in second due to increment, after while loop str now points to the end of the string. So use the concept carefully.

String functions in C

C language have a header file for strings called “string.h”, which provides lots of useful functions to operate on strings. Some of them are listed below. There are many important string functions defined in "string.h" library. To use these functions we have to include <string.h> in our programs. Some of these functions are described in next section with their use.

- **strlen()**

```
size_t strlen(const char *str);
```

The strlen( ) function returns the length of the null-terminated string pointed to by str. The null terminator is not counted. for example, the following line will print 10.

```
printf("%d", strlen("Code Hello");
```

- **strcpy()**

```
char *strcpy(char *str1, const char *str2);
```

The strcpy( ) function copies the contents of str2 into str1. str2 must be a pointer to a null-terminated string. The strcpy() function returns a pointer to str1.

- **strcat()**

```
char *strcat(char *str1, const char *str2);
```

The strcat( ) function concatenates a copy of *str2* to *str1* and terminates str1 with a null. The first character of str2 will overwrite the null terminator of str1. The string str2 is untouched by the operation. If the arrays overlap, the behaviour of strcat( ) is undefined. The strcat( ) function returns str1. Remember, no bounds checking takes place, so it is the programmer's responsibility to ensure that str1 is large enough to hold both its original contents and those of str2.

- **strcmp()**

```
int strcmp(const char *str1, const char *str2);
```

The strcmp( ) function lexicographically compares two strings and returns an integer based on the outcome as shown here:

Less than zero - str1 is less than str2

Zero - str1 is equal to str2

Greater than zero - str1 is greater than str2

- **strchr()**

```
char *strchr(const char *str, int ch);
```

The strchr( ) function returns a pointer to the first occurrence of the low-order byte of ch in the string pointed to by str. If no match is found, a null pointer is returned. For example the following line will print “friend first”: -

```
printf ("%s", strchr("Hello friend first", 'f'));
```

- **strstr()**

```
char *strstr(const char *str1, const char *str2);
```

The strstr( ) function returns a pointer to the first occurrence in the string pointed to by str1 of the string pointed to by str2. It returns a null pointer if no match is found. For example the following line will print “friend first”: -

```
printf ("%s", strstr("Hello friend first", "friend"));
```

whereas the following line prints nothing: -

```
printf ("%s", strstr("Hello friend first", "code"));
```

These are some of the most commonly used functions from string.h. There are lots of other functions also, you can explore string.h file for others. below programs shows some of these functions used. You need to refer the header file for further details.

```c
#include<stdio.h>
#include<string.h>
int main()
{
  char ss1[50]={"CodeQuotient - Get better at coding"};
  char ss2[50];
  strcpy(ss2,ss1);
  printf("Value of second string is: %s\n\n", ss2);

  strcat(ss1,ss2);
  printf("Value of string s1 is: %s\n\n", ss1);

  char s1[20]="Coding";
  char s2[20]="Coding";
  char s3[20]="programming";
  char s4[20]="Programming";
  char s5[20]="String";
  char s6[20]="Value";
  printf("Coding == Coding: %d\n", strcmp(s1,s2));  // Equal so produce 0.
  printf("Coding == programming: %d\n", strcmp(s2,s3));  // C comes first in ASCII than p (The difference between the differentiating character ASCII(C)=67, ASCII(p)=112 so 67-112=-45) so -45 will be printed.
  printf("programming == Programming: %d\n", strcmp(s3,s4));  // p comes after P in ASCII (The difference between the differentiating character ASCII(p)=112, ASCII(P)=80 so 112-80=32) so -45 will be printed.
  printf("Programming == String: %d\n", strcmp(s4,s5));  // P comes before S so -3 will be printed as ASCII(P) - ASCII(S)
  printf("String == Value: %d\n", strcmp(s5,s6));       // S comes before V so  -3 will be printed as ASCII(S) - ASCII(V)
  printf("Value == ValuE: %d\n", strcmp("Value","ValuE"));  // In starting characters are same but at later e comes after E.
  // (If string constants are compared it will return -1, 0, 1 not differences as above)

  return 0;
}
```

Video Explanation

<iframe width="560" height="315" src="https://www.youtube.com/embed/cY3q4k5Ucnl" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>