Strings in C

Introduction

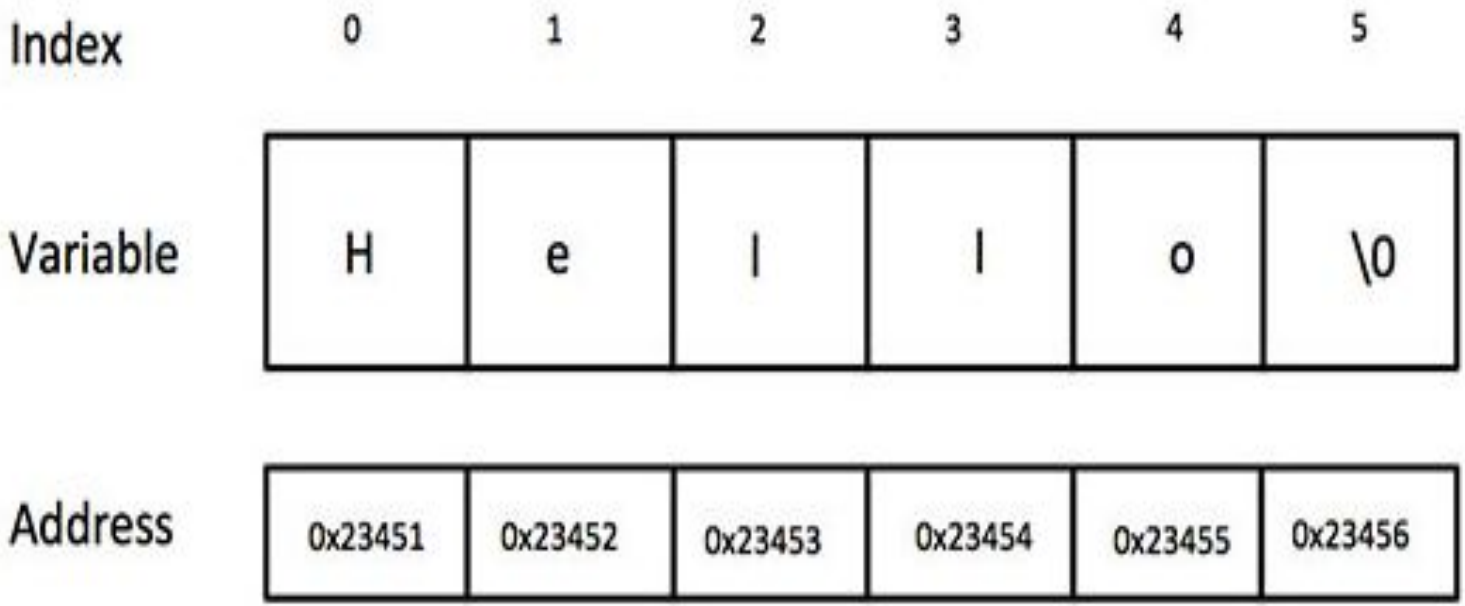
Strings are actually one-dimensional array of characters terminated by a null character '\0'.

Declaration/ Initilazation

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

```
char greeting[] = "Hello";
```

Following is the memory presentation of the above defined string in C/C++ –



Program 1

```c
#include <stdio.h>

int main ()

{

   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

   printf("Greeting message: %s\n", greeting );

}
```

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

```c
#include <stdio.h>
#include <string.h>
int main () {
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );
```

```
len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );
   return 0;
}
```

Test 1

Write a C program to copy String S1 into String S2 without using built-in functions.

Test 2

Write a C program to join Strings S1 and String S2 into String S3. (Kindly note the resultant string will be in S3).

USER DEFINED FUNCTIONS

USER DEFINED FUNCTION

- A function is a block of code that performs a specific task.
- C allows you to define functions according to your need. These functions are known as user-defined functions.

```
#include <stdio.h>
int addNumbers(int a, int b); //function prototype
int main()
 {
int n1,n2,sum;
printf("Enters two numbers: ");
scanf("%d %d",&n1,&n2);
sum = addNumbers(n1, n2); // function call printf("sum = %d",sum);
return 0;
}
```

```
int addNumbers(int a,int b) // function definition
 {
 int result;
result = a+b;
return result; // return statement
}
```

ELEMENTS OF THE USER DEFINED FUNCTIONS

- In order to make a user defined function, there must be 3 elements related to functions:

1. Function definition
2. Function call
3. Function declaration

GENERAL FORMAT OF A FUNCTION

```
return_type function_name(parameter list)
{
        local variable declaration;
        executable statement1;
        executable statement2;
        ……..
        ……..
        return statement;
}
```

Location of function in a program

- Type 1: placing function first followed by the entire main program.
- Type 2: placing function declaration before main program followed by main program and then followed by function body.

TYPE 1

```
#include <stdio.h>
int addNumbers(int a, int b);
int addNumbers(int a,int b) // function definition
 {
 int result;
result = a+b;
return result; // return statement
}
```

TYPE 1

```
int main()
 {
int n1,n2,sum;
printf("Enters two numbers: ");
scanf("%d %d",&n1,&n2);
sum = addNumbers(n1, n2); // function call printf("sum = %d",sum);
return 0;
}
```

TYPE 2

```
#include <stdio.h>
int addNumbers(int a, int b); //function prototype
int main()
 {
int n1,n2,sum;
printf("Enters two numbers: ");
scanf("%d %d",&n1,&n2);
sum = addNumbers(n1, n2); // function call printf("sum = %d",sum);
return 0;
}
```

TYPE 2 contd…

```
int addNumbers(int a,int b) // function definition
 {
 int result;
result = a+b;
return result; // return statement
}
```

Function parameters

- These are the values that are supplied to the function during its call to do specific action.

- Actual Parameters: Parameters that are transferred from the calling program.

- Formal Parameters: Dummy parameters that are transferred into the calling function.

Types of function based on parameters

- TYPE 1: with parameter with return value

```
#include<stdio.h>
int add(int i, int j);
void main()
{
        int sum,a=10,b=20;
        sum=add(a,b);
        printf(“Sum is %d”,sum);
```

```
}

int add(int i,int j)
{
        int sum;
        sum=i+j;
        return sum;
}
```

Type 2: with parameter without returning value

```
#include<stdio.h>
void add(int i, int j);
void main()
{
        int a=10,b=20;
        add(a,b);
}
void add(int i,int j)
{
        int sum;
        sum=i+j;
        printf(“sum is %d”,sum);
}
```

Type 3: without parameter with return value

```
#include<stdio.h>
int add();
void main()
{
        int sum;
        sum=add();
        printf(“Sum is %d”,sum);
}
int add()
{
        int sum, i=10,j=20;
        sum=i+j;
        return sum;
}
```

Type 4: without parameter without returning value

```
#include<stdio.h>
int add();
void main()
{
        add();
}
int add()
{
    int sum, i=10,j=20;
    sum=i+j;
    printf(“sum is %d”,sum);
}
```

Parameter passing mechanism

- Parameter passing by value or call by value
- Parameter passing by reference or call by reference

Call by value

```
#include<stdio.h>
int add(int i, int j);
void main()
{
        int sum,a=10,b=20;
        sum=add(a,b);
        printf(“Sum is %d”,sum);
}
int add(int i, int j)
{
        int sum;
        sum=i+j;
        return sum;
}
```

Call by reference

```
#include<stdio.h>
int add(int *i, int *j);
void main()
{
        int sum,a=10,b=20;
        sum=add(&a,&b);
        printf(“Sum is %d”,sum);
}
int add(int *i, int *j)
{
        int sum;
        sum=*i+*j;
        return sum;
}
```

Passing Arrays in Functions(pass by value)

```
#include<stdio.h>
void display(int p);
void main()
{
        int n,i,a[10];
        printf(“Enter the number of elements”);
        scanf(“%d”,&n);
        printf(“Enter number:”);
        for(i=0;i<n;i++)
                scanf(“%d”,&a[i]);
        printf(“Elements are \n”);
        for(i=0;i<n;i++)
                display(a[i]);
}
```

```
void display(int p)
{
        printf(“%d\t”,p);
}
```

Array passing by reference

```
#include<stdio.h>
void display(int b[20],int n);
void main()
{
        int n,i,a[10];
        printf(“Enter the number of elements”);
        scanf(“%d”,&n);
        printf(“Enter number:”);
        for(i=0;i<n;i++)
                scanf(“%d”,&a[i]);
        printf(“Elements are \n”);
                display(a,n);
}
```

```
void display(int b[],int n)
{
    int i;
    for(i=0;i<n;i++)
        printf("%d\t",b[i]);
}
```

Passing strings in functions(pass by value)

```
#include<stdio.h>
void display(char ch);
void main()
{
        char c[50];
        printf(“Enter string:\n”);
        scanf(“%s”,&c);
        for(i=0;c[i]!=‘\0’;i++)
                display(c[i]);
}
void display(char ch)
{
        putchar(ch);
}
```

Swapping of two numbers

```
#include<stdio.h>
void swap(int *d,int *g);
void main()
{
        int a,b;
        printf(“Enter a and b:”);
        scanf(“%d%d”,&a,&b);
        swap(&a,&b);
        printf(“After swapping a is %d b is %d”,a,b);
}
```

```
void swap(int *d,int *g)
{
        int temp;
        temp=*d;
        *d=*g;
        *g=temp;
}
```

Program to find the sum of n numbers

```
#include<stdio.h>
int add(int p);
void main()
{
        int n,sum;
        printf(“Enter n:”);
        scanf(“%d”,&n);
        sum=add(n);
        printf(“Sum is %d”,sum);
}
```

RECURSIVE FUNCTION

- RF are those that call themselves during their execution until certain conditions are not satisfied.
- Ex: Factorial

  fact(5)= 5* fact(4)

  fact(4)=4*fact(3)

  fact(3)=3*fact(2)

  fact(2)=2*fact(1)

  fact(1)=1*fact(0)

  fact(0)=1

Factorial Program

```
#include<stdio.h>
int factorial(int n);
void main()
{
        int n,fact;
        printf(“Enter a number:”);
        scanf(“%d”,&n);
        fact=factorial(n);
        printf(“Factorial is %d”,fact);
}
```

```
int factorial(int n)
{
        if( n == 0)
                return 1;
        else
                return (n*factorial(n-1));
}
```


Thank
you !!!
...