

Chapter 9: Pointer and Strings

Er.Nipun Thapa

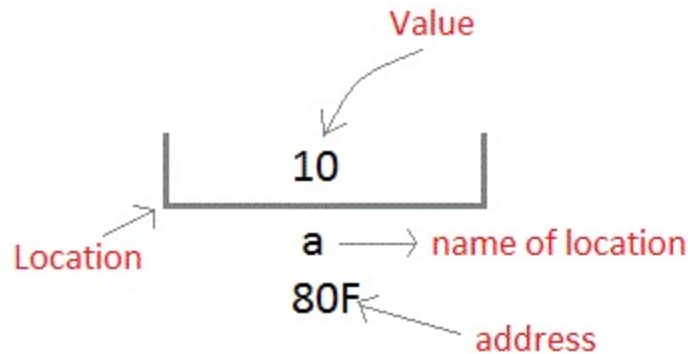
What is Pointer in C?

- The **Pointer** in C, is a variable that stores address of another variable.
- A pointer can also be used to refer to another pointer function.
- A pointer can be incremented/decremented, i.e., to point to the next/ previous memory location.
- The purpose of pointer is to save memory space and achieve faster execution time.
- A pointer enable us to access a variable that is defined outside the function.
- Pointers are used in dynamic memory allocation
- They are used to pass array to functions.
- They produce compact, efficient and powerful code with high execution speed.
- The pointers are more efficient in handling the data table.
- They use array of pointers in character strings result in saving of data stroge space in memory. Sorting string using pointer is very efficient.
- With the help of pointer, variable can be swapped without physically moving them.
- Pointer are closely associated with arrays and therefore provide an alternate way to access individual array elements.

Concept of Pointers

- Whenever a **variable** is declared in a program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value.
- This location has its own address number, which we just saw above.
- Let us assume that system has allocated memory location 80F for a variable a.

```
int a = 10;
```



We can access the value **10** either by using the variable name **a** or by using its address **80F**. The variables which are used to hold memory addresses are called **Pointer variables**.

Declaration and Initialization of Pointer Variables

Declaration of C Pointer variable

- Data type of a pointer must be same as the data type of the variable to which the pointer variable is pointing. void type pointer works with all data types, but is not often used.

Syntax : ***datatype *pointer_name;***

Here ,

- The asterisk(*) tells that the variable *pointer_name* is a pointer variable
- *pointer_name* needs a memory location
- *pointer_name* points to a variable of type *data_type*

Here are a few examples:

```
int *ip // pointer to integer variable
float *fp; // pointer to float variable
double *dp; // pointer to double variable
char *cp; // pointer to char variable
```

Declaration and Initialization of Pointer Variables

Initialization of C pointer variable

- Pointer Initialization is the process of assigning address of a variable to a pointer variable.
- Pointer variable can only contain address of a variable of the same data type.
- In C language address operator “&” is used to determine the address of a variable. The “&” (immediately preceding a variable name) returns the address of the variable associated with it.

```
int x;  
int *ptr;  
ptr=&x;
```

Here, x is an integer variable and pointer ptr is initiating with the address of x.

Declaration and Initialization of Pointer Variables

Valid Example	Invalid Example
<code>int *p</code>	<code>int *p</code>
<code>int num</code>	<code>float num;</code>
<code>p= &num</code>	<code>p= &num</code>

Declaration and Initialization of Pointer Variables

```
#include<stdio.h>
void main()
{
    int a = 10;
    int *ptr;    //pointer declaration
    ptr = &a;    //pointer initialization
}
```

Pointer variable always point to variables of same datatype. Let's have an example to showcase this:

```
#include<stdio.h>
void main()
{
    float a;
    int *ptr;
    ptr = &a;    // ERROR, type mismatch
}
```

Declaration and Initialization of Pointer Variables

If you are not sure about which variable's address to assign to a pointer variable while declaration, it is recommended to assign a NULL value to your pointer variable. A pointer which is assigned a NULL value is called a NULL pointer.

```
#include <stdio.h>

int main()
{
    int *ptr = NULL;
    return 0;
}
```


Using the pointer or Dereferencing of Pointer

Once a pointer has been assigned the address of a variable, to access the value of the variable, pointer is dereferenced, using the indirection operator or dereferencing operator *.

//Lab 7 QN1 : WAP to illustrate use of pointer

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

```
int main()
```

```
{
```

```
    int a, *p; // declaring the variable and pointer
```

```
    a = 10;
```

```
    p = &a; // initializing the pointer
```

```
    printf("value of a = %d", *p);
```

```
    printf("\nValue of a = %d", *&a);
```

```
    printf("\nAddress of a = %d", &a);
```

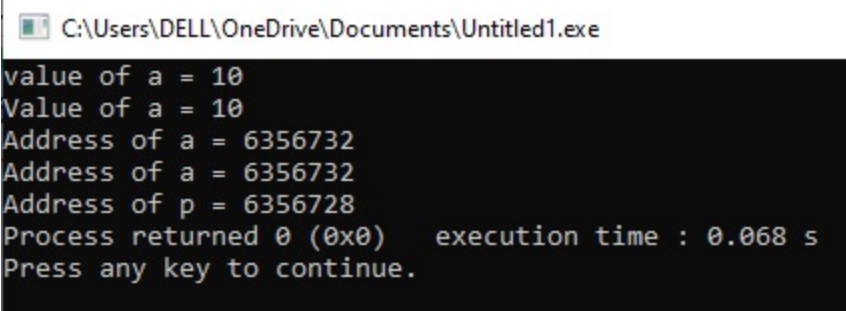
```
    printf("\nAddress of a = %d", p);
```

```
    printf("\nAddress of p = %d", &p);
```

```
    return 0;
```

```
}
```

O/P

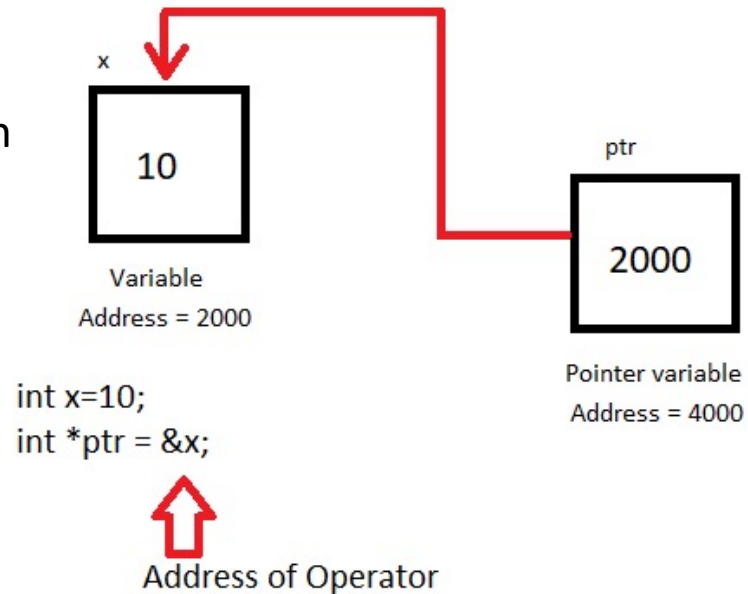


```
C:\Users\DELL\OneDrive\Documents\Untitled1.exe
value of a = 10
Value of a = 10
Address of a = 6356732
Address of a = 6356732
Address of p = 6356728
Process returned 0 (0x0)   execution time : 0.068 s
Press any key to continue.
```

Initialization of C Pointer variable

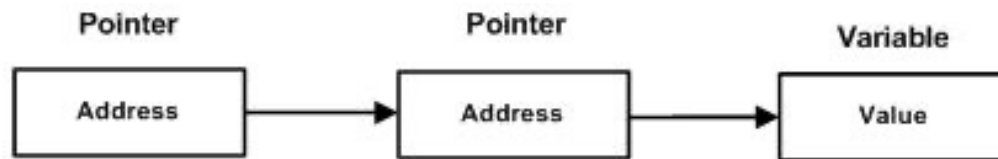
- **Pointer Initialization** is the process of assigning address of a variable to a **pointer** variable.
- It contains the address of a variable of the same data type. In C language **address operator** & is used to determine the address of a variable.
- The & (immediately preceding a variable name) returns the address of the variable associated with it.

```
int x = 10;  
int *ptr;    //pointer declaration  
ptr = &x;    //pointer initialization
```



Pointer to Pointer(Double Pointer)

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers.
- Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int –

```
int **var;
```

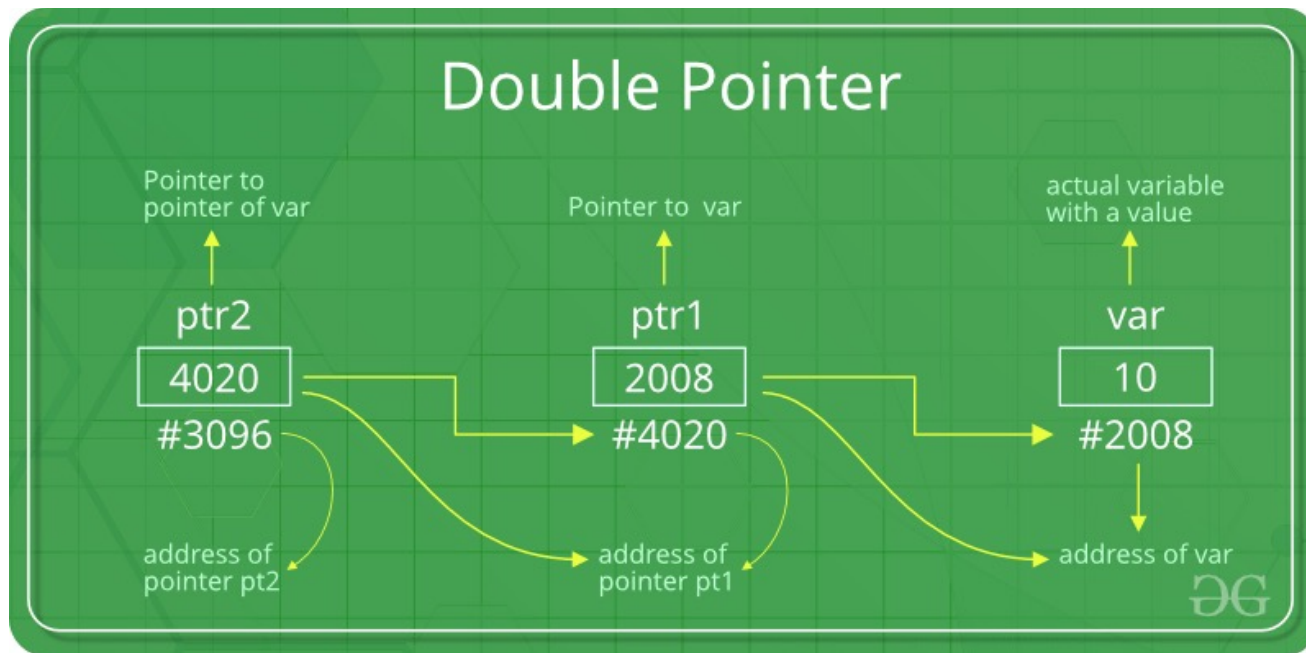
Pointer to Pointer(Double Pointer)

- When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example –

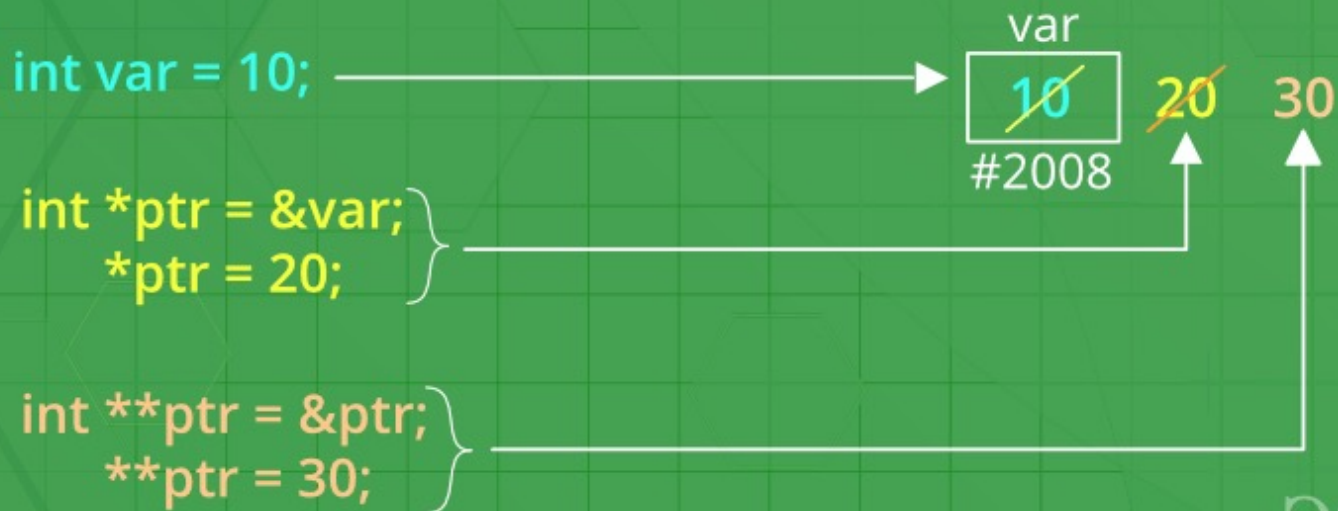
How to declare a pointer to pointer in C?

- Declaring Pointer to Pointer is similar to declaring pointer in C. The difference is we have to place an additional '*' before the name of pointer.
- **Syntax:**

```
int **ptr; // declaring double pointers
```



How pointer works in C



//Lab 7 QN2 : WAP to illustrate use of double pointer

```
#include <stdio.h>
// C program to demonstrate pointer to pointer
int main()
{
    int var = 789;
    int *ptr2;
    int **ptr1;
    // storing address of var in ptr2
    ptr2 = &var;
    // Storing address of ptr2 in ptr1
    ptr1 = &ptr2;
    // Displaying value of var using
    // both single and double pointers
    printf("Value of var = %d\n", var );
    printf("Value of var using single pointer = %d\n", *ptr2 );
    printf("Value of var using double pointer = %d\n", **ptr1);

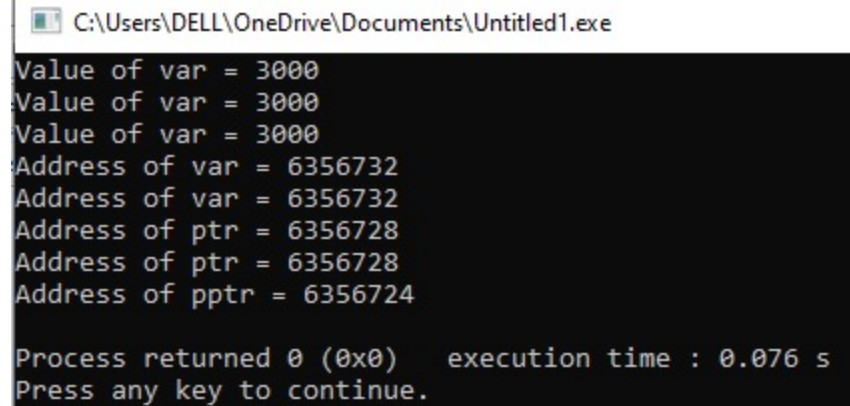
    return 0;
}
```

```
Output:
Value of var = 789
Value of var using single pointer = 789
Value of var using double pointer = 789
```

//Lab 7 QN3 : WAP to illustrate use of double pointer

```
#include <stdio.h>
int main ()
{
    int var;
    int *ptr;
    int **pptr;
    var = 3000;
    ptr = &var;
    pptr = &ptr;
    printf("Value of var = %d\n", var );
    printf("Value of var = %d\n", *ptr );
    printf("Value of var = %d\n", **pptr);
    printf("Address of var = %d\n",&var);
    printf("Address of var = %d\n",ptr);
    printf("Address of ptr = %d\n",&ptr);
    printf("Address of ptr = %d\n",pptr);
    printf("Address of pptr = %d\n",&pptr);
    return 0;
}
```

O/P



```
C:\Users\DELL\OneDrive\Documents\Untitled1.exe
Value of var = 3000
Value of var = 3000
Value of var = 3000
Address of var = 6356732
Address of var = 6356732
Address of ptr = 6356728
Address of ptr = 6356728
Address of pptr = 6356724

Process returned 0 (0x0)   execution time : 0.076 s
Press any key to continue.
```


Array of Pointer

- A pointer variable always contains an address of a variable. So, an array of pointers is actually an array of memory addresses of different variables.

Syntax: **data_type *pointer_name[size]**

Example :

```
int *p[5];
```

This declaration an array of 5 pointers, each of which contains the address of an integer. The first pointer is p[0] or *p and the fifth pointer is p[4] or *p+4

Lab7 QN4: Illustrating the use of array of pointer

```
#include<stdio.h>
Int main()
{
    int a=1,b=2,c=3,d=4,e=5;
    int *p[5];
    int i;
    p[0]=&a;
    p[1]=&b;
    p[2]=&c;
    p[3]=&d;
    p[4]=&e;
    for(i=0;i<5;i++)
    {
        printf("\nP[%d]=%d",i,*p[i]);
    }
    return 0;
}
```

```
o/p
P[0]=1
P[1]=2
P[2]=3
P[3]=4
P[4]=5
```

Lab7 QN5: Illustrating the use of array of pointer

```
#include<stdio.h>
Int main()
{
    int a[5]={1,2,3,4,5};
    int *p[5];
    int i;
        for(i=0;i<5;i++)
        {
            p[i]=&a[i];
            printf("\nP[%d]=%d",i,*p[i]);
        }
    return 0;
}
```

```
o/p
P[0]=1
P[1]=2
P[2]=3
P[3]=4
P[4]=5
```

Lab7 QN6: Illustrating the use of array of pointer

```
#include <stdio.h>
const int MAX = 3;
int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr[MAX];
    for ( i = 0; i < MAX; i++)
    {
        ptr[i] = &var[i]; /* assign the address of integer. */
    }
    for ( i = 0; i < MAX; i++)
    {
        printf("Value of var[%d] = %d\n", i, *ptr[i] );
    }
    return 0;
}
```

Lab7 QN6: Illustrating the use of array of pointer

```
#include <stdio.h>
const int MAX = 4;
int main ()
{
    char *names[] = { "Zara Ali", "Hina Ali", "Nuha Ali", "Sara Ali" };
    int i = 0;
    for ( i = 0; i < MAX; i++)
    {
        printf("Value of names[%d] = %s\n", i, names[i] );
    }
    return 0;
}
```

o/p

Value of names[0] = Zara Ali

Value of names[1] = Hina Ali

Value of names[2] = Nuha Ali

Value of names[3] = Sara Ali

Passing Pointer to function

- C programming allows passing a pointer to a function.
- To do so, simply declare the function parameter as a pointer type.

Lab 7 QN7: Program that lower case letter to upper case and upper case to lower case by passing of pointer to function.

```
#include<stdio.h>
Void conversion(char *);
Int main()
{
    char ch;
    printf("enter character:");
    scanf("%c",&ch);
    conversion(&ch);
    printf("After conversion:%c",ch)
    return 0;
}
Void conversion(char *c)
{
    if(*c>=97 && *c<=122)
        *c=*c -32;
    else if(*c>=65 && *c<=90)
        *c = *c+32;
}
```

Pointer Arithmetic

- We can perform arithmetic operations on the pointers like addition, subtraction, etc.
- However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer.
- In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:
 - Increment
 - Decrement
 - Addition
 - Subtraction
 - Comparison

Incrementing Pointer in C

- If we increment a pointer by 1, the pointer will start pointing to the immediate next location.
- This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.

`new_address = current_address + i * size_of(data type)`

32-bit

For 32-bit int variable, it will be incremented by 4 bytes.

64-bit

For 64-bit int variable, it will be incremented by 8 bytes.

Lab 7 QN8: Illustrate the example of increment pointer

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

```
int main()
```

```
{
```

```
    int number=50;
```

```
    int *p;//pointer to int
```

```
    p=&number;//stores the address of number variable
```

```
    printf("Address of p variable is %u \n",p);
```

```
    p=p+1;
```

```
    printf("After increment: Address of p variable is %u \n",p);
```

```
    // in our case, p will get incremented by 4 bytes.
```

```
    return 0;
```

```
}
```

o/p

Address of p variable is 3214864300

After increment: Address of p variable is 3214864304

Decrementing Pointer in C

- Like increment, we can decrement a pointer variable. If we decrement a pointer, it will start pointing to the previous location.
- The formula of decrementing the pointer is given below:

$$\text{new_address} = \text{current_address} - i * \text{size_of}(\text{data type})$$

32-bit

For 32-bit int variable, it will be decremented by 4 bytes.

64-bit

For 64-bit int variable, it will be decremented by 8 bytes.

Lab 7 QN8: Illustrate the example of increment pointer

```
#include<stdio.h>
int main()
{
    int number=50;
    int *p;//pointer to int
    p=&number;//stores the address of number variable
    printf("Address of p variable is %u \n",p);
    p=p-1;
    printf("After increment: Address of p variable is %u \n",p);
    // in our case, p will get incremented by 4 bytes.
    return 0;
}
```

```
o/p
Address of p variable is 3214864300
After increment: Address of p variable is 3214864296
```

String and Pointer

- We know that a string is a sequence of characters which we save in an array.
- And in C programming language the `\0` null character marks the end of a string.

Creating a string

- In the following example we are creating a string `str` using `char` character array of size 6.

```
char str[6] = "Hello";
```

The above string can be represented in memory as follows.

```
char str[6] = "Hello";
```

index	0	1	2	3	4	5
value	H	e	l	l	o	\0
address	1000	1001	1002	1003	1004	1005

Each character in the string `str` takes 1 byte of memory space.

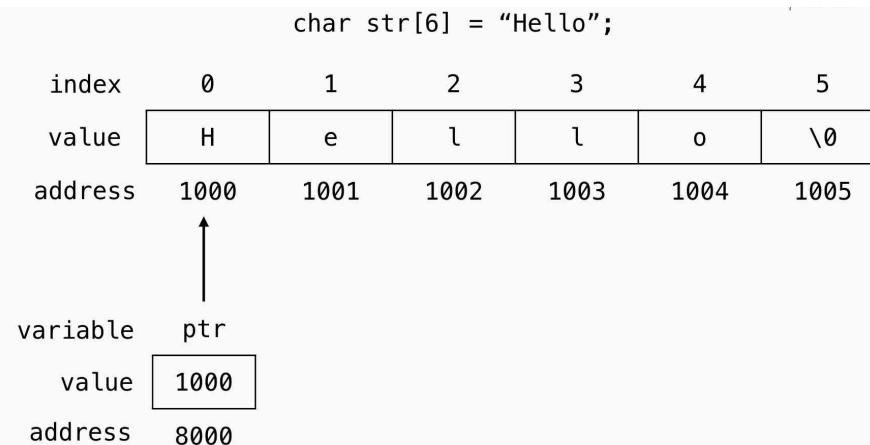
String and Pointer

Creating a pointer for the string

- The variable name of the string `str` holds the address of the first element of the array i.e., it points at the starting memory address.
- So, we can create a character pointer `ptr` and store the address of the string `str` variable in it. This way, `ptr` will point at the string `str`.
- In the following code we are assigning the address of the string `str` to the pointer `ptr`.

```
char *ptr = str;
```

We can represent the character pointer variable `ptr` as follows.



Lab 7 QN9; In the following example we are using while loop to print the characters of the string variable str.

```
#include <stdio.h>
int main(void)
{
    // string variable
    char str[6] = "Hello";
    // pointer variable
    char *ptr = str;
    // print the string
    while(*ptr != '\0')
    {
        printf("%c", *ptr); // move the ptr pointer to the next memory location
        ptr++;
    }
    return 0;
}
```

String functions in c language with examples

- There are various string functions which we can use in C Language.
- We need to include header file **string.h** in our program to use these functions in our program.

Various string functions in c language are:

- **strcpy()**
- **strncpy()**
- **strcat()**
- **strlen()**
- **strrev()**
- **strcmp()**
- **strcmpi()**
- **strncmp()**
- **strlwr()**
- **strupr()**

1. strcpy()

- strcpy() stands for string copy. This function is used to copy value of one string variable or string constant in another string variable.
- The header file required for this function is “string.h”.

The syntax for strcpy() is

strcpy(Str_Target,Str_Source);

- **Str_Target** is the string variable in which want to store the value.
- **Str_Source** is the string value which we want to store in string variable **Str_Target**. This value can be a string constant or a string variable. Size of **Str_Target** should be larger than or equal to the size of value to be stored in the variable.

We can't assign value to a string variable directly as

```
char name[20];  
name="Amit";
```

In place of assignment operator =, we need to use strcpy() function.

```
char name[20];  
strcpy(name,"Amit");
```

Lab 7 Qn10:Program to demonstrate the use of strcpy() function

```
#include<stdio.h>
#include<string.h>
int main()
{
    char studentname[20];
    strcpy(studentname,"Amit");
    printf("\nStudentname=%s",studentname);
    return 0;
}
```

Output

Studentname=Amit

2. strcat()

- strcat() stands for string concat.
- This function is used to combine values of two string variables together.
- The header file required for this function is “string.h”.
- The syntax for strcat() function is

strcat(Str_Target,Str_Source);

- **Str_Target** is the string variable whose value we want to combine with some other string value.
- **Str_Source** is the string value which we want to combine with the value of string variable

Str_Target. This value can be a string constant or a string variable. Size of Str_Target should be larger than or equal to the combined size of **Str_Target** and **Str_Source**.

Lab 7 QN11: Program to demonstrate the use of strcat() function.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char studentname[20]="Amit";
    strcat(studentname," Singh");
    printf("\nStudentname=%s",studentname);
    return(0);
}
```

Output

Studentname=Amit Singh

In above program, string variable **studentname** contains value "**Amit**". After applying **strcat()** function, its value will be concatenated with "**Singh**" and the final value of string variable **studentname** would become **Amit Singh**

3. strlen()

- strlen() stands for string length.
- This function is used to find number of characters stored in a string variable or string constant.
- This function doesn't count null character.
- The header file required for this function is "string.h"

Lab7: Qn12: Program to demonstrate the use of strlen() function.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char studentname[20]="Amit";
    int n;
    n=strlen(studentname);
    printf("\nNumber of characters=%d",n);
    return(0);
}
```

Output

Number of characters=4

In above program, string variable studentname contains value "Amit". strlen() function finds number of characters in the value "Amit" in string variable studentname that contains 4 characters. So strlen() would return 4.

4. strcmp()

- strcmp() stands for string compare.
- This function is used to compare value of one string value with another string value.
- The header file required for this function is “string.h”.
- This Function returns 0 if the values of both the strings being compared are same.

Lab7: QN13: Program to demonstrate the use of strcmp() function.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char studentname[25];
    strcpy(studentname,"Amita");
    if(strcmp(studentname,"Amit")==0)
        printf("\nWelcome");
    else
        printf("\nBye");
    return(0);
}
```

Output

Bye

In this program, string variable studentname has been assigned value "Amita". Further in the program, value of string variable studentname has been compared with the string value Amit. If the value of studentname matches Amit it would show Welcome. But in the program, the value of studentname doesn't match Amit so the output would be Bye.

Unit 9

Finished