A string constant is a set of character that enclosed within the double quotes and is also called a literal. Whenever a string constant is written anywhere in a program it is stored somewhere in a memory as an array of characters terminated by a NULL character ('\o').

Example – "m"

"Tajmahal"

"My age is %d and height is %f\n"

The string constant itself becomes a pointer to the first character in array.

Example-char crr[20]="Taj mahal";

1000	1001	1002	1003	1004	1005	1006	1007	100	1009
T	a	j		M	A	H	a	l	\o

It is called base address.

Lecture Note: 13

String library function

There are several string library functions used to manipulate string and the prototypes for these functions are in header file "string.h". Several string functions are

strlen()

This function return the length of the string. i.e. the number of characters in the string excluding the terminating NULL character.

It accepts a single argument which is pointer to the first character of the string.

For example-

```
        strlen("suresh");
        It return the value 6.
```

In array version to calculate legnth:-

```
        int str(char str[])
{
        int i=0;
        while(str[i]!='\o')
        {
                i++;
        }
        return i;
}
```

```
        Example:-
        #include<stdio.h>
         #include<string.h>
        void main()
{
        char str[50];
        print("Enter a string:");
        gets(str);
        printf("Length of the string is %d\n",strlen(str));
}
```

Output:

```
Enter a string: C in Depth
Length of the string is 8
```

strcmp()

This function is used to compare two strings. If the two string match, strcmp() return a value 0 otherwise it return a non-zero value. It compare the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two string are not same.

```
        strcmp(s1,s2)
        return a value:
                <0 when s1<s2
                =0 when s1=s2
                >0 when s1>s2
```

The exact value returned in case of dissimilar strings is not defined. We only know that if s1<s2 then a negative value will be returned and if s1>s2 then a positive value will be returned.

For example:

```
/*String comparison……………………*/
#include<stdio.h>
#include<string.h>
void main()
{
        char str1[10],str2[10];
        printf("Enter two strings:");
        gets(str1);
        gets(str2);
        if(strcmp(str1,str2)==0)
{
        printf("String are same\n");
}
else
{
        printf("String are not same\n");
}
}
```

strcpy()

This function is used to copying one string to another string. The function strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string.

The old content of the destination string str1 are lost. The function returns a pointer to destination string str1.

Example:-

```
        #include<stdio.h>

        #include<string.h>

        void main()

{

        char str1[10],str2[10];

        printf(“Enter a string:”);

        scanf(“%s”,str2);

        strcpy(str1,str2);

        printf(“First string:%s\t\tSecond string:%s\n”,str1,str2);

        strcpy(str,”Delhi”);

        strcpy(str2,”Bangalore”);

printf(“First string :%s\t\tSecond string:%s”,str1,str2);
```

strcat()

This function is used to append a copy of a string at the end of the other string. If the first string is “”Purva” and second string is “Belmont” then after using this function the string becomes “PusvaBelmont”. The NULL character from str1 is moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected. A pointer to the first string str1 is returned by the function.

Example:-

```
        #include<stdio.h>

        #include<string.h>

        void main()

{

        char str1[20],str[20];

        printf(“Enter two strings:”);

        gets(str1);

        gets(str2);

        strcat(str1,str2);

        printf(“First string:%s\t second string:%s\n”,str1,str2);

        strcat(str1,”-one”);

        printf(“Now first string is %s\n”,str1);

}
```

Output

```
        Enter two strings: data

        Base
```

```
First string: database second string: database
`         Now first string is: database-one
```

Lecture Note: 14

FUNCTION

A function is a self contained block of codes or sub programs with a set of statements that perform some specific task or coherent task when it is called.

It is something like to hiring a person to do some specific task like, every six months servicing a bike and hand over to it.

Any 'C' program contain at least one function i.e main().

There are basically two types of function those are

1. Library function

2. User defined function

The user defined functions defined by the user according to its requirement

System defined function can't be modified, it can only read and can be used. These function are supplied with every C compiler

Source of these library function are pre complied and only object code get used by the user by linking to the code by linker

Here in system defined function description:

Function definition : predefined, precompiled, stored in the library

Function declaration : In header file with or function prototype.

Function call : By the programmer

User defined function

Syntax:-

```
Return type        name of function (type 1 arg 1, type2 arg2, type3 arg3)
  Return type          function name        argument list of the above syntax
```

So when user gets his own function three thing he has to know, these are.

Function declaration

Function definition

Function call

These three things are represented like

```
        int function(int, int, int);      /*function declaration*/
        main()      /* calling function*/
          {
        function(arg1,arg2,arg3);
         }
    int function(type 1 arg 1,type2 arg2,type3, arg3)   /*function definition/*
    {
    Local variable declaration;
  Statement;
  Return value;
   }
```

Function declaration:-

Function declaration is also known as function prototype. It inform the compiler about three thing, those are name of the function, number and type of argument received by the function and the type of value returned by the function.

While declaring the name of the argument is optional and the function prototype always terminated by the semicolon.

Function definition:-

Function definition consists of the whole description and code of the function.

It tells about what function is doing what are its inputs and what are its out put

It consists of two parts function header and function body

Syntax:-

```
return type function(type 1 arg1, type2 arg2, type3  arg3)  /*function header*/
 {
   Local variable declaration;
   Statement 1;
   Statement 2;
   Return value
 }
```

The return type denotes the type of the value that function will return and it is optional and if it is omitted, it is assumed to be int by default. The body of the function is the compound statements or block which consists of local variable declaration statement and optional return statement.

The local variable declared inside a function is local to that function only. It can’t be used anywhere in the program and its existence is only within this function.

The arguments of the function **definition** are known as **formal arguments**.

Function Call

When the function get called by the calling function then that is called, function call. The compiler execute these functions when the semicolon is followed by the function name.

Example:-

```
  function(arg1,arg2,arg3);
```

The argument that are used inside the function call are called **actual argument**

Ex:-

```
   int S=sum(a, b);                    //actual arguments
```

Actual argument

The arguments which are mentioned or used inside the function call is knows as actual argument and these are the original values and copy of these are actually sent to the called function

It can be written as constant, expression or any function call like

```
Function (x);
 Function (20, 30);
Function (a*b, c*d);
Function(2,3,sum(a, b));
```

Formal Arguments

The arguments which are mentioned in function definition are called formal arguments or dummy arguments.

These arguments are used to just hold the copied of the values that are sent by the calling function through the function call.

These arguments are like other local variables which are created when the function call starts and destroyed when the function ends.

The basic difference between the formal argument and the actual argument are

1) The formal argument are declared inside the parenthesis where as the local variable declared at the beginning of the function block.

2). The **formal argument** are automatically initialized when the copy of actual arguments are passed while other local variable are assigned values through the statements.

Order number and type of actual arguments in the function call should be match with the order number and type of the formal arguments.

Return type

It is used to return value to the calling function. It can be used in two way as

```
          return
     Or   return(expression);
      Ex:-  return (a);
          return (a*b);
          return (a*b+c);
```

Here the 1st return statement used to terminate the function without returning any value

Ex:- /*summation of two values*/

```
    int sum (int a1, int a2);
   main()
```

```
{
  int a,b;
   printf(“enter two no”);
   scanf(“%d%d”,&a,&b);
    int S=sum(a,b);
    printf(“summation is = %d”,s);
     }
    int sum(intx1,int y1)
     {
   int z=x1+y1;
    Return z;
    }
```

Advantage of function

By using function large and difficult program can be divided in to sub programs and solved. When we want to perform some task repeatedly or some code is to be used more than once at different place in the program, then function avoids this repeatition or rewritten over and over.

Due to reducing size, modular function it is easy to modify and test

Notes:-

C program is a collection of one or more function.

A function is get called when function is followed by the semicolon.

A function is defined when a function name followed by a pair of curly braces

Any function can be called by another function even main() can be called by other function.

```
main()
{

function1()

}

function1()

{

Statement;

function2;

}

function 2()

{


}
```

So every function in a program must be called directly or indirectly by the main() function. A function can be called any number of times.

A function can call itself again and again and this process is called **recursion**.

A function can be called from other function **but** a function can't be defined in another function

<u>Lecture Note: 15</u>

Category of Function based on argument and return type

i) Function with no argument & no return value

Function that have no argument and no return value is written as:-

```
  void  function(void);
 main()
  {
void  function()
 {
Statement;
 }
```

Example:-

```
  void  me();
    main()
   {
     me();
    printf("in  main");
   }
  void   me()
  {
    printf("come  on");
  }
```

Output: come on

inn main

ii) Function with no argument but return value

Syntax:-

```
 int   fun(void);
 main()
 {
     int  r;
     r=fun();
  }
     int   fun()
   {
   reurn(exp);
   }
```

Example:-

```
  int   sum();
 main()
 {
 int   b=sum();
 printf("entered   %d\n, b");
  }
int   sum()
  {
 int  a,b,s;
  s=a+b;
 return  s;
    }
```

Here called function is independent and are initialized. The values aren't passed by the calling function .Here the calling function and called function are communicated partly with each other.

Lecture Note: 16

iii) function with argument but no return value

Here the function have argument so the calling function send data to the called function but called function dose n't return value.

```
Syntax:-
            void   fun  (int,int);
            main()
           {
          int (a,b);
         }
   void   fun(int x, int y);
    {
    Statement;
    }
```

Here the result obtained by the called function.

iv) function with argument and return value

Here the calling function has the argument to pass to the called function and the called function returned value to the calling function.

Syntax:-

```
              fun(int,int);
               main()
             {
                int r=fun(a,b);
              }
                  int  fun(intx,inty)
               {
                        return(exp);
               }
```

Example:

```
          main()
         {
          int  fun(int);
          int  a,num;
          printf(“enter  value:\n”);
          scanf(“%d”,&a)
```

```
int num=fun(a);
  }
 int  fun(int x)
 {
  ++x;
           return x;
  }
```

Call by value and call by reference

There are two way through which we can pass the arguments to the function such as **call by** value and **call by reference**.

1. Call by value

In the call by value copy of the actual argument is passed to the formal argument and the operation is done on formal argument.

When the function is called by 'call by value' method, it doesn't affect content of the actual argument.

Changes made to formal argument are local to block of called function so when the control back to calling function the changes made is vanish.

Example:-

```
  main()
 {
 int  x,y;
change(int,int);
```

```
          printf(“enter two values:\n”);
           scanf(“%d%d”,&x,&y);
            change(x ,y);
       printf(“value of x=%d and y=%d\n”,x ,y);
          }
       change(int a,int b);
      {
       int k;
       k=a;
       a=b;
       b=k;
      }
```

Output: enter two values: 12

 23

Value of x=12 and y=23

2. Call by reference

Instead of passing the value of variable, address or reference is passed and the function operate on address of the variable rather than value.

Here formal argument is alter to the actual argument, it means formal arguments calls the actual arguments.

Example:-

```
void main()
```

```
        {
                int a,b;
                change(int *,int*);
                printf("enter two values:\n");
                scanf("%d%d",&a,&b);
                change(&a,&b);
                printf("after  changing two value of a=%d and b=%d\n:"a,b);
        }
                change(int *a, int *b)
                {
                int  k;
                  k=*a;
                *a=*b;
              *b= k;
printf("value in this function  a=%d and b=%d\n",*a,*b);
}
```

```
Output: enter two values: 12
32
Value in this function a=32  and b=12
After changing two value of  a=32 and b=12
```

So here instead of passing value of the variable, directly passing address of the variables. Formal argument directly access the value and swapping is possible even after calling a function.