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

Lecture Note: 15

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.