Function

- Is a self-contained subprogram to do some specific, well-defined task.
- Divides the program into different modules.
- Avoids repetition of code.
- Easy to understand, modify, and debug.
- Stored in a library and can be reused.
- Function can be:
  - Pre defined
  - User defined
    - Declaration
    - Definition
    - Call

P1. Function to draw a line

```
void drawline(void);                    /*Function Declaration*/
main( )
{
    drawline( );                        /*Function Call*/
}
void drawline(void)                     /*Function Definition*/
{
    int i;
    for(i=1;i<=80;i++)
        printf("-");
}
```

P2. Find the sum of two numbers

```
int sum(int x,int y);          /*Function declaration*/
main( )
{
    int a,b,s;
    printf("Enter values for a and b : ");
    scanf("%d %d",&a,&b);
    s=sum(a, b);                        /*Function call*/
    printf("Sum of %d and %d is %d\n",a,b,s);
}
int sum(int x,int y)          /*Function definition*/
{
    int s;
    s=x+y;
    return s;
}
```

Function Definition

- Tells what the function will do and what are its inputs and outputs
- Consists of two parts:
  - Header and Body
- Syntax of a function definition:

```
return_type   func_name( type1 arg1,type2 arg2,…… )
{
    local variables declarations;
    statement;
    ……….
    return(expression);
}
```

Return Type:

- Return type is optional and if omitted, it is assumed to be **int** by default
- A function can return either one value or no value
- If it does not return any value then **void** should be written in place of return type

Function Name:

- specifies the name of the function and can be any valid C identifier

Function Arguments:

- The type and name of the arguments must be given in parentheses
- Known as formal arguments and used to accept values
- A function can take no argument or any number of arguments
- If there are no arguments then either the parentheses can be left empty or void can be written inside the parentheses

Body of the Function:

- Consists of declarations of variables and C statements followed by an optional **return statement**
- The variables declared inside the function are known as local variables
- They have existence only in the function in which they are declared, they can not be used anywhere else in the program
- The **return statement** is optional, it may be absent if the function does not return any value
- Function definition cannot be placed inside another function definition

Function Call

- When a function is called, the control passes to the called function, which is executed and after this , the control is transferred to the statement following the function call in the calling function
- Example:

```
s = sum(a, b);
b = max(x, y)*10;
if ( isprime(x) = =1 )
        printf("Number is prime");
printf("%d\n", sum(a+b) );
```

If the function is declared as void then it cannot be used in this way, for Example: S = drawline( );

- A function call can be used as a statement, if a semicolon is placed after it
- In this case if the function returns any value, it is just discarded
- For example-

```
draw(x, y);
display( a, b, c);
printprimes( );
```

- A function call cannot occur on the left hand of an assignment statement

```
func(x, y) = a; /* Invalid */
```

- The code 'of a function is executed only when it is called by some other function.
- If the function is defined and not called even once then it's code will never be executed.
- A function can be called more than once, so the code is executed each time it is called.
- The execution of a function finishes either when the closing braces of the function body are reached or if return statement is encountered

Function Declaration

- If definition of the called function is placed before the calling function, then declaration is not needed
- Function declaration tells the compiler about three things:
  - Name of the function
  - Number and type of arguments
  - Type of value returned by the function
- The names of the arguments in function declaration are optional

P3. Find the sum of two numbers

```
int sum(int x,int y)
{
    int s;
    s=x+y;
    return s;
}

main()
{
    int a,b,s;
    printf("Enter values for a and b : ");
    scanf("%d %d",&a,&b);
    s=sum(a,b);                    /*Function call*/
    printf("Sum of %d and %d is %d\n",a,b,s);
}
```

P4. Find whether a number is even or odd

```
void find( int n);
main( )
{
    int num;
    printf("Enter a number : ");
    scanf("%d",&num);
    find(num);
}
void find( int n)
{
    if(n%2==0)
        printf("%d is even\n",n);
    else
        printf("%d is odd\n",n);
}
```

P5. Find the maximum of two numbers

```
int  max(intx,int  y);
main( )
{
    int  a,b;
    printf("Enter  two  numbers  :  ");
    scanf("%d%d",&a,&b);
    printf("Maximum  of  %d  and  %d  is  :  %d\n",a,b,max(a,b));
}
max(int  x,int  y)
{
    if(x>y)
        return  x;
    else
        return  y;
}
```

P6. Understand the use of return statement

```
void funct(int age,float ht);
main( )
{
    int age;
    float ht;
    printf("Enter age and height: " );
    scanf("%d %f", &age, &ht);
    funct(age,ht);
}
```

```
void funct(int age,float ht)
{
    if(age>25)
    {
        printf("Age should be less than 25\n");
        return;
    }
    if(ht<5)
    {
        printf("Height should be more than 5\n");
        return;
    }
    printf("Selected\n");
}
```

- The value returned by the return statement may be any constant, variable, expression or even any other function call (which returns a value)
- It is optional to enclose the returning value in parentheses
- We can use multiple return statements in a function but as soon as the first return statement is encountered the function terminates and all the other statements following it are not executed
- Examples of valid return statements:

```
return 1;
return x++;
return ( x+y*z );
return ( 3 * sum(a, b) );
```

```
int cmpdate( int d1, int m1, int y1, int d2, int m2, int y2)
{
    if(y1<y2)
        return 1;
    if(y1>y2)
        return -1;
    if(m1<m2)
        return 1;
    if(m1>m2)
        return -1;
    if(d1<d2)
        return 1;
    if(d1>d2)
        return -1;
    return 0;
}
```

P7. Find the factorial of a number

```
long int factorial(int n);
main( )
{
    int num;
    printf("Enter a number : ");
    scanf("%d",&num);
    if(num<0)
        printf("No factorial of negative number\n");
    else
        printf("Factorial of %d is %ld\n",num,factorial(num));
}
```

```
long  int  factorial(int  n)
   {
        int  i;
        long  int  fact=1;
        if(n==0)
            return  1;
        for(i=n;i>1;i- -)
            fact*=i;
        return  fact;
   }
```

- A function can return only one value.
- If we want to return more than one value then we have to use another technique discussed in further chapters
- It is incorrect to try to return more than one value using comma, for example:

  return 1, 2, 3;
- Here this expression is using comma operator and hence only the rightmost value will be returned
- If no value is to be returned from the function then it should be declared as void
- All functions, which are not of void type, return a value.
- If the function is not of void type and no value is returned through return statement, then a garbage value is returned automatically.
- If the value returned is not of the type specified in the function definition then it is converted to that type if the conversion is legal

Actual arguments:

- The arguments which are mentioned in the function call are known as actual arguments, since these are the values which are *actually* sent to the called function.
- Actual arguments can be written in the form of variables, constants or expressions or any function call that returns a value, For example-

```
fun(x)
func(a*b, c*d+k )
func( 22, 43 )
func( 1, 2, sum(a, b) )
```

Formal arguments:

- The name of the arguments, which are mentioned in the function definition are called formal arguments
- Formal arguments are like other local variables of the function which are created when the function call starts and are destroyed when the function ends.
- However there are two differences:
- Formal arguments are declared inside the parentheses while other local variables are declared at the beginning of the function block.
- Formal arguments are automatically initialized with the values of the actual arguments passed, while other local variables are assigned values through the statements written inside the function body
- The order, number and type of actual arguments in the function call should match with the order, number and type of formal arguments in the function definition

P8. Program to understand formal and actual arguments

```
main( )
{
    int  m=6,n=3;
    printf("%d\t",multiply(m,n));
    printf("%d\t",multiply(15,4));
    printf("%d\t",multiply(m+n,m-n));
    printf("%d\n",multiply(6,sum(m,n)));
}
multiply(int  x,int  y)
{
    int  p;
    p=x*y;
    return  p;
}
sum(int  x,  int  y)
{
    return  x+y;
}
```

Output:

18 60 27 54

P9. Program to understand formal and actual arguments

```
main( )
{
    int  a=6,b=3;
    func(a,b);
    func(15,4);
    func(a+b,a-b);
}
func(int  a,int  b)
{
    printf("a  =  %d      b  =  %d\n",a,b);
}
```

Output:

```
a = 6     b = 3
a = 15    b = 4
a = 9     b = 3
```

- The names of formal and actual arguments maybe same or different, because they are written in separate functions
- In the last program, we can see that the values of a and b inside the main( ) are 6 and 3, while a and b inside the func( ) are initialized with values of actual arguments sent on, each call.
- Any changes made to the formal arguments inside the function do not affect the actual arguments

Types Of Functions

The functions can be classified into four categories on the basis of the arguments and return value

1. Functions with no arguments and no return value
2. Functions with no arguments and a return value
3. Functions with arguments and no return value
4. Functions with arguments and a return value

P10. Functions with no arguments and no return value

```
void dispmenu(void);
main( )
{
    int choice;
    dispmenu( );
    printf("Enter your choice :");
    scanf("%d",&choice);
}
void dispmenu(void )
{
    printf("1.Create database\n");
    printf("2.Insert new record\n");
    printf("3.Modify a record\n");
    printf("4.Delete a record\n");
    printf("5.Display all records\n");
    printf("6.Exit\n");
}
```

P11. Functions with no arguments but a return value.
Find the sum of squares of all odd numbers from 1 to 25

```
main( )
{
    printf("%d\n",func( ));
}
int  func(void)
{
    int  num,s=0;
    for(num=1;num<=25;num++)
    {
        if(num%2!=0)
            s+=num*num;
    }
    return  s;
}
```

P12. Functions with arguments but no return value. Find the type and area of a triangle

```
#include<math.h>
void type(float a,float b,float c);
void area(float a,float b,float c);
main()
{
    float a,b,c;
    printf("Enter the sides of triangle : ");
    scanf("%f%f%f",&a,&b,&c);
    if(a<b+c && b<c+a && c<a+b)
    {
        type(a,b,c);
        area(a,b,c);
    }
    else
        printf("No triangle possible with these sides\n");
}
```

```
void type(float a,float b,float c)
{
    if((a*a)+(b*b)==(c*c)||(b*b)+(c*c)==(a*a)||(c*c)+(a*a)==(b*b))
        printf("The triangle is right angled triangle\n");
    if(a==b && b==c)
        printf("The triangle is equilateral\n");
    else if(a==b || b==c || c==a)
        printf("The triangle is isosceles\n");
    else
        printf("The triangle is scalene\n");
}

void area(float a,float b,float c)
{
    float s, area;
    s=(a+b+c)/2;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("The area of triangle = %f\n",area);
}
```

P14. Function with arguments and returns value. Find the sum of digits of any number

```
int  sum(int  n);
main()
{
    int  num;
    printf("Enter  the  number  :  ");
    scanf("%d",&num);
    printf("Sum  of  digits  of  %d  is  %d\n",num,sum(num));
}
int  sum(int  n)
{
    int  i,sum=0,rem;
    while(n>0)
    {
        rem=n%10;          /*rem  takes  the  value  of  last  digit*/
        sum+=rem;
        n/=10;             /*skips  the  last  digit  of  number*/
    }
    return  (sum);
}
```