FUNCTIONS:

Definition : A function is a self-contained subprogram that carries out some specific, well-defined task.

→ Function generally classified into 3 types.

1) User defined and system declared function. : main()
2) system defined and system declared function : library func?
3) User defined and user declared function : user defined function.

User-defined function : User can create their own function for performing any specific task of the program. These types of functions are called user defined function.

Advantages of using function :

1) Generally difficult problems is divided into subproblems and then solved. This divide and conquer technique is implemented in C through functions. so the C functions modularizes and divides the work of a program.

2) when some specific code is to be used more than once and at different places in the program, then the use of function avoids repetition of that code.

3) The program becomes easily understandable, modifiable and easy to debug and test. It becomes simple to write the program and understand what work is done by each part of the program.

4) reduction of work and time.

5) Functions can be stored in a library and reusability can be achieved

6) Functions can be considered as "black-box", because arguments will be passed as input and provide result in the form of return value.

Parts of a Function :

The parts of a function are ;

1. Function declaration / function prototype
2. Function definition
3. Function call.

1. Function Declaration :

→ Like variables, all functions in a C program must be declared, before they are invoked.

→ A function declaration (also known as function prototype) consists of 4 parts.

- Function type (return type)
- Function name
- parameter list
- Terminating semicolon.

→ The general syntax for function declaration is ;

```
Function-type function-name (parameter list);
```

→ Example : `int sum (int x, int Y); /* function prototype */`

→ When a function does not take any parameters and does not return any value, its prototype is written as ;

```
void sum (void);
```

A prototype declaration may be placed in two places in a program.

1. Above all the functions (including main)
2. Inside a function definition

- Function prototype is used to inform the compiler of the name, data type and no. & datatypes of the arguments of all user-defined functions employed in the program.

- When we place the declaration above all the functions (in the global declaration section), the prototype is referred to as a global prototype. Such declarations are available for all the functions in the program.
- When we place the declaration in a function definition (in the local declaration section), then the prototype is called a local prototype. Such declarations are primarily used by the functions containing them.

The place of declaration of a function defines a region in a program in which the function may be used by other functions. This region is known as the scope of the function.

→ It is a good programming style to declare prototypes in the global declaration section before main. It adds flexibility, provides an excellent quick reference to the functions used in the program and enhances documentation.

2. Function definition:

→ The function definition consists of the whole description and code of a function.

→ It tells what the function is doing and what are its input and outputs.

→ A function definition consists of two parts - a function header and a function body.

→ The general syntax of a function definition is;

```
return-type function-name (parameter list)      ← called func^n
{
    local variable declaration;
    statement;
    - - - - - -
    return (expression);
}
```

→ The first line in the function definition is known as the function header and after this the body of the function is written enclosed in curly braces.

→ Example :
```
int sum ( int x, int Y)
{
  int S ;
  S = X + Y ;
  return s ;
}
```

→ If the function is not returning any value, then void is written at the place of return-type, and hence since it does not return accept any arguments so void is written inside parentheses.

```
void sum (void)
{
  int s ;
  s = x + Y ;
}
```

3. Function Call :

→ The function definition describes the what a function can do, but to actually use it in the program, the function should be called somewhere.

→ A function is called by simply writing its name followed by the argument list inside the parentheses.

→ General syntax for function call is;

```
function-name ( argument list) ;
```
↳ Calling func^n

→ Example : `sum ( a , b) ;`

→ If there are no arguments, the function call should have the empty parentheses, like `sum ( ) ;`

→ Here, the function-name is known as the called function while the function in which this function call is placed is known as the calling function.

Function Arguments :

→ The calling function sends some values to the called function for Communication ; these values are called arguments or parameters

→ In function, arguments are classified into two types.

1. Actual arguments.
2. Formal arguments.

1. Actual Arguments :

→ The arguments which are mentioned in the function call are known as the 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.

→ Example :
```
fun (x)
func (22.43)
func (a*b, c*d+k) etc.
```

2. Formal Arguments :

→ The name of the arguments, which are mentioned in the function definition are called formal or dummy arguments. Since they are used just to hold the values that are sent by the calling function.

→ The formal arguments are simply like other local variables of the function which are created when the function call starts and destroyed when the function ends.

return statement :

→ The return statement is used in a function to return a value to the calling function.

→ It may also be used for immediate exit from the called function to the calling function without returning a value.

→ The statement can appear anywhere inside the body of the function.

→ There are two ways in which it can be used -

```
return ;
return (expression) ;
```

→ Here return is a keyword. The first form of return statement is used to terminate the function without returning any value. In this case only return keyword is written.

→ The second form of return statement is used to terminate a function and return a value to the calling function. The value returned by the return statement may be any constant, variable, expression or even any other function call which return a value.

Example:
```
return 1 ;
return x++ ;
return (x+y*z); etc.
```

Examples of user-defined functions :

```
/* program to find the sum of two numbers */
#include <stdio.h>
#include <conio.h>
int sum (int x, int y);   /* Function prototype */
void main()
  {
    int a, b, s ;
    clrscr();
```

```
printf(" Enter the values of a & b :");
scanf(" %d %d", &a, &b);
   s = sum(a, b);   /* Function call */ [calling function]
              /* Actual arguments */
printf(" Sum = %d\n", s);
getch();
}
int sum( int x, int y)   /* Function definition */  -> [Called function]
                         (x, y: Formal Arguments)
 {
   int s;
   s = x + y;
   return s;
 }
```

Out-put : Enter the values of a and b : 14 10

sum = 24

TYPES OF FUNCTIONS :

→ The function 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 return value.

1. Functions with no arguments and no return value :

→ When a function has no arguments, it does not receive any data from the calling function.

→ Similarly, when it does not return a value, the calling function does not receive any data from the called function. Infact, there is no data transfer between the calling function and the called function.

→ The functions that have no arguments and no return value are written as ;

```
void func(void);
main()
{
   -----
   func();
   -----
}
void func(void)
   {
   -----
   statement;
   -----
   }
```

In the above example, the function func() is called by main() and the function definition is written after the main() function. As the function func() has no arguments, main() cannot send any data to func() and since it has no return statement, hence function cannot return any value to main(). There is no communication between the calling function and the called function. Since there is no return value, these type of functions cannot be used as operands in expressions.

Examples:

```
/* write a program to draw a line */
#include <stdio.h>
void drawline (void); /* Function prototype */
main()
  {
    drawline();   /* Function call */
  }
void drawline (void)   /* function definition */
    {
       int i;
       for(i=1; i<=80; i++)
            printf("_");
    }
```

```
/* program for displaying the menu */
#include <stdio.h>
void dispmenu(void);
main()
 {
    int choice;
    dispmenu();
    printf("Enter ur 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. Exit\n");
   }
```

2. Function with no arguments but a return value :

→ The function with no arguments but a return value, do not receive any arguments but they can return a value

→ The function with no arguments but a return value can be written as ;

```
int func (void);
main()
{
    int n ;
    - - - - -
    n = func() ;
    - - - - -
}
int func (void)
{
    - - - - -
    - - - - - -
    return (expression) ;
}
```

Example :

```
/* write a program that returns the sum of square of all
   odd numbers from 1 to 25 */
#include <stdio.h>
int func (void);
main()
{
   printf ("%d\n", func());
}
   int func(void)
      {  int num, s=0;
         for (num = 1 ; num <= 25 ; num++)
               {
                  if ( num % 2 != 0)
```

```
        S = S + (num * num);
    }
    return S;
}
```

output : 2925

3. Function with arguments but no return value :

→ These type of functions have arguments, hence the calling function can send data to the called function but the called function does not return any value.

→ These functions can be written as -

```
void func (int, int);
main()
  {  ----
     func (a,b);
  }  ----
void func (int, int)
  {  -----
     Statements;
  }  ------
```

Example :

```
/* program to find the area and type of a triangle */
#include<stdio.h>
#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 a 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 ( " Triangle is ninght angled triangle \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 ( " Area of triangle = %f \n", area) ;
    }
```

4. Function with arguments and return value :

→ These type of functions have arguments, so the calling function can send data to the called function, it can also return any value to the calling function using return statement

→ This function can be written as -

```
int func (int, int);
main()
{
   int r;
   ------
   r = func (a,b);
   ------
   func (c,d);
}
int func (int a, int b);
 {
   -------
   -------
   return (expression);
 }
```

Example :

```
/* write a program to find the sum of digits of any no. */
#include <stdio.h>
#include <conio.h>
int sum (int n);
main()
  { int num;
    clrscr();
   printf ("enter the number :");
    scanf ("%d", &num);
    printf ("sum of digits of no. is %d\n", sum(num));
  }
```

```
int sum ( int n)
  {
    int sum=0, rem ;
     while (n>0)
        {
          rem = n %10;
           sum = sum + rem ,
           n = n/10 ;
        }
   return (sum) ;
}
```

Examples of Functions :

```
/* write a program to find the reverse of a number */
#include <stdio.h>
#include <conio.h>
int reverse ( int n);
void main()
   {
      int num;
      clrscr();
      printf ("Enter a number :");
      scanf ("%d", &num);
   printf ("Reverse of a no. is %d\n", reverse (num));
   getch();
   }
  int reverse ( int n)
     {
       int rem, rev = 0 ;
       while (n>0)
          {
            rem = n% 10 ;
            rev = (rev *10)+ rem ;
          }  n = n/10;
     return rev ;
```

```
/* write a program to check whether a number is palindrome
   or not */
#include <stdio.h>
#include <conio.h>
int reverse (int n);
void main()
{
   int num;
   clrscr();
   printf("Enter a number :");
   scanf("%d", &num);
   printf("Reverse of the number is %d\n", reverse(num));
  if ( num == reverse(num) )
      printf("Number is a palindrome\n");
  else
      printf("The number is not a palindrome\n");
   getch();
}
int reverse (int n)
   {
     int rev=0, rem;
     while (n>0)
        {
          rem = n%10;
          rev = (rev*10) + rem;
          n = n/10;
        }
     return rev;
   }
```

Output

```
Enter a number : 103
Reverse of a number is 301.
The number is not a palindrome
```

```
/* write a program to find whether the number is prime or
   not */
#include <stdio.h>
# include <conio.h>   # include <math.h>
int isprime (int n);
void main()
  {
    int num;
    clrscr();
   printf ("Enter a number :");
   scanf ("%d", &num);
  if (isprime (num))
       printf ("Number is prime\n");
   else
       printf ("Number is not prime\n");
   getch();
}
  int isprime (int n)
   {
     int i, flag=1;
    for (i=2; i<=sqrt (n); i++)
        {  if (n%i==0)
              {
                flag=0;
                 break;
              }
        }
     return (flag);
}
```

```
/* write a program to find GCD (Greatest Common Divisor) of
   two numbers */

# include <stdio.h>
# include <conio.h>
int gcd ( int x , int Y) ;
void main( )
  {
    int a, b, P ;
    clrscr( ) ;
   printf (" Enter two nos : ") ;
    scanf (" %d %d", &a , &b) ;
 if ( a<b)
      printf ( " a = %d  b = %d ", a, b) ;
       P = gcd (a,b);
      printf ( " GCD = %d\n", P) ;
 else
      printf ( " a = %d  b = %d ", a, b);
        P = gcd ( b, a) ;
        printf ( " GCD = %d\n", P) ;
    getch( ) ;
  }
int gcd ( int x , int Y)
   {
     int n ;
     n = Y % x ;
 if ( n = = 0)
      return x ;
else
   {
```

```
    while ( r != 0)
        {
            Y = X ;
            X = r ;
            r = Y % X ;
        }
    return X ;
  }
}
```

/* write a program to find the factorial of a number */

```
# include <stdio.h>
# include <conio.h>
int factorial (int n);
void main()
{
   int num, fact ;
   clrscr(),
   printf (" Enter the number : ");
   scanf (" %d", &num);
  if ( num < 0)
        printf (" No factorial found ");
  else
        fact = factorial (num);
      printf (" factorial is %d\n", fact);
   getch();
}
 int factorial (int n)
   {
      int fact = 1 ;
     while ( n > 1)
         {
            fact = fact * n ;
```

```
    n--;
  }
    return fact;
}
```

/* write a program to find the fibonacci series upto a given range */

```
#include <stdio.h>
#include <conio.h>
void main()
 {
    int a,b;
    clrscr();
  void  fibo(int,int);
    a=0;
    b=1;
  printf("%d%d",a,b);
    fibo(a,b);
   getch();
}
 void  fibo(int a, int b)
    {
      int c,n;
      printf("Enter range:");
      scanf("%d",&n);
     for( c=a+b; c<=n; c=a+b)
         {
            printf("%d",c);
              a=b;
              b=c;
         }
```

output: Enter range : 5

0 1 1 2 3 5

program to find the sum of two numbers using all 4 types of function.

1. No argument and no return value :-

```
#include <stdio.h>
#include <conio.h>
void sum (void);
void main()
{ clrscr();
  sum();
  getch();
}
void sum (void)
{
  int a, b, c;
  printf ("Enter the values of a and b:");
  scanf ("%d %d", &a, &b);
  c = a+b;
  printf ("%d", c);
}
```

2. Argument and no return value :-

```
#include <stdio.h>
#include <conio.h>
void sum (int, int);
void main()
{
  int a, b;
  clrscr();
  printf ("Enter two nos:");
```

```
scanf ("%d %d", &a, &b);
 sum(a, b);
 getch();
}
void sum ( int c, int d)
  {
    int e ;
    e = c + d ;
    printf ("%d", e),
  }
```

3. Argument and return value :-

```
#include <stdio.h>
#include <conio.h>
int sum (int, int);
void main()
{
   int a, b, p ;
   clrscr();
   printf (" Enter two numbers : ");
   scanf ("%d %d", &a, &b);
    p = sum (a, b);
   printf ("%d", p);
   getch();
}
int sum ( int c, int d )
    { int x ;
      x = c + d ;
     return x ;
  }
```

4. No argument and return value :-

```
#include <stdio.h>
#include <conio.h>
int sum();
void main()
{
   int p; clrscr();
   p = sum();
   printf("%d", p);
   getch();
}
int sum()
{
   int a, b, x;
   printf("enter two nos");
   scanf("%d %d", &a, &b);
   x = a + b;
   return x;
}
```

/* Write a program to check whether an integer is strong or not */

If the sum of factorial of all digits of a given number is the number itself, then it is a strong number

145 = 1! + 4! + 5! = 145

```
#include <stdio.h>
#include <conio.h>
int strong(int a);
void main()
{
   int i, x, n, s;
   clrscr();
```

```
printf ("Enter a number:");
scanf ("%d", &x);
t = x;
s = 0;
while (x>0)
   {
      r = x %10;
      s = s + strong (r);
      x = x/10;
   }
if (t == s)
      printf (" %d is a strong number\n", t);
 else
      printf (" %d is not a strong number \n", t);
   getch();
}
int strong (int a)
{
   int i, f;
    f = 1;
    for( i=1; i<=a; i++)
       {
          f = f * i;
       }
    return f;
  }
```


Output : Enter a number : 123

123 is not a strong number

Enter a number : 145

145 is a strong number .

/* Write a program to find the power of a given number using function */

```
#include <stdio.h>
#include <conio.h>
int pow( int a , int b);
void main()
{
   int n, x, power;
   clrscr();
   printf("Enter the values for n and x :");
   scanf("%d%d", &n, &x);
   power = pow(n, x);
   printf(" power of a given number is : %d\n", power);
   getch();
}
int pow ( int a, int b)
{
    int result = 1 ;
  if ( b == 1 )
      return a ;
else {
       while (b>0)
          {
            result = result * a ;
              b--;
           }
       return (result);
      }
  }
```

output : Enter the value for n and x : 2 3

power of a given number is : 8

/* write a program using function to check whether a number is armstrong or not */ [153 = $1^3+5^3+3^3$]

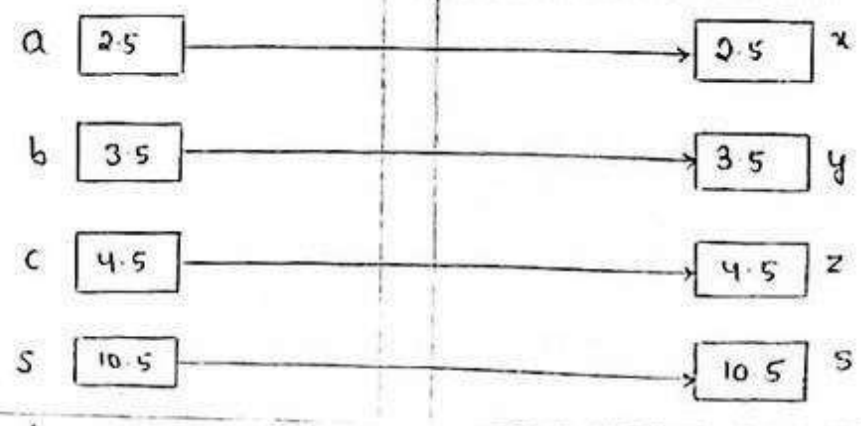
```
#include <stdio.h>
#include <conio.h>
int armstrong (int n);
void main()
{
    int n, a, t;
    clrscr();
    printf("Enter a number :");
    scanf ("%d", &n);
    t = n ;
    a = armstrong (n);
 if ( t == a)
        printf (" The no is an armstrong number :");
  else  printf (" The no. is not an armstrong number");
  getch();
}
int armstrong (int n)
 {   int  s=0, r ;
    while ( n > 0)
          {  r = n % 10 ;
             s = s + (r * r * r) ;
             n = n/10 ;
          }
    return s;
 }
```

Passing arguments to Functions (or parameter passing):

This mechanism is used to convey information to the function is the arguments or parameters. when values are passed to the function via actual arguments, the value of the actual argument is copied in the formal argument. Therefore, the changes made to the formal argument have no effect on the actual argument. This procedure of passing the value of the argument to a function is known as call by value.

```
/* Calling Function */
#include <stdio.h>
#include <conio.h>
void main()
{
   float a,b,c, sum;
   clrscr();
float addition(float, float, float);
printf("Enter 3 numbers:");
scanf("%f%f%f",&a,&b,&c);
 sum = addition(a,b,c);
printf("Result = %f\n", sum);
 getch();
}
```

```
/* called Function */
float addition(float x, float y,
                        float z)
{
  float s;
  s = x+y+z;
  return s;
}
```



(Call by value parameter passing mechanism)

→ Call-by value mechanism does not change the content of the arguments in the calling function even if they are changed in the called function.

→ Formal parameters are stored in the local data area of the called function. So the changes to the formal parameter within the function will effect only the local copy, and will have no effect on the actual argument.

→ Function accesses arguments by using the argument names in the function header. The actual and formal arguments should math in number, type and order.

→ The value of actual arguments are assigned to the formal arguments on one to one basis, starting with the first argument.

Advantages and disadvantages of call by value :

passing an argument by value has advantages and disadvantages.

→ The advantages are that it allows a single-valued argument to be written as an expression, rather than being restricted to a single variable.

→ Furthermore, in cases where the argument is a variable, the value of this variable is protected from alterations which take place within the function.

→ The main disadvantage is that information can not be transferred back to the calling portion of the program via arguments. In other words, passing by value is a strictly one-way method of transferring information.