Functions :

* A function is a set of instructions to carry out a particular task.
* Generally the functions are classified into standard function & user defined functions.
* sqrt(), pow(), sin() etc are standard functions, such functions are selective.
* Programmer can define other functions also and are named as user defined functions.

Defination of Function

```
type name_of_function (parameters list)
{
    Variable declaration ;
    Statement 1
    :
    :
    return (value_computed)
}
```

where

type → data type of value returned by function

name-of-function → name of user defined function

parameter list → list of variables that receives value from calling function

return → a keyword used to send output of function, back to the calling function.

value_computed → The value to be returned to the calling function after processing

Function Basics

1) Function Declaration: All functions in C language need to be declared before they are used.

2) Function definition: It is actual code for the function to perform specific task.

③ Function Call: In order to use function in program, function call is used.
The program or function that calls the function is reffered as calling function

ex

```
# include <stdio.h>
void print_message();  // function declaration
void main()
  {
    print_message(); // function call
  }

print_message()  // function definition
  {
  printf("This is a module called print_message\n");
  }
```

Program to add two numbers & display result using function addnum().

Sol^n

```
#include <stdio.h>
Void main()
    {
     int n1, n2, result;
     printf ("Enter two numbers \n");
      scanf ("%d %d", &n1, &n2);
     result = addnum (n1, n2);
     printf ("The sum of two numbers is %d", result);
    }
  int addnum (int val1, int val2)
    {
      int sum;
      sum = val1 + val2;
       return (sum);
    }
```

Arguments & Parameters

Arguments and parameters are variables used in a function.

→ Variables used in a function reference are called arguments. These are written in paranthesis followed by the name of function in function call.

→ Variables used in the function definition are called parameters. They are also referred to as formal parameters, because they are not accepted values
These are written in paranthesis followed by name of function in function definition.

Function with Arguments but No Return Value

→ Here the called function receives the data from the calling function.

* The arguments and parameters should match in number, data type & order, but the called function does not return any value back to the calling function.

* It is one way data communication between the calling

Scope Rule:

Scope Rule explains, if an entity (ie variable, parameter and function) is visible or accessible of certain places, Thus, places where an entity can be accessed or visible is reffered to the scope of that entity.

The simplest rule is as follows

① The scope of an entity is the program or function in which it is declared.

② A global entity is visible to all contained functions. Including the function in which that entity is declared

③ An entity declared in the scope of another entity is always a different entity if their names are identical.

7

```
#include <stdio.h>
#include <conio.h>
void main()
  {
    int x, y;
    printf("Enter the value of x & y \n");
    scanf("%d %d", &x, &y);
    maximum(x, y);
  }
maximum(p, q)
maximum(int p, int q)
  {
     if (p>q)
       printf("Maximum = %d", p);
     else
       printf("maximum = %d", q);
  }
```

Function with No Arguments & Return Value

→ Here the called function does not receive any data from calling function. It manages with its local data to carry out specified task.

* After the processing is complete the called function returns the computed data to calling function

* It is also a one-way data communication between calling function & called function

eg

```
# include < stdio.h >
void main ( )
    { float sum;
      float total ( );
       sum = total ( );
       printf ("sum = %f \n", sum);
    }

float total ( )
      {
        float x, y;
         x = 20.0;
         y = 10.0;
      return (x+y);
```

Parameter Passing Mechanism

Parameter passing means the way in which the arguments are given to the called function.

There are two techniques of passing parameters

(i) Call by Value

(ii) Call by reference

Call by Value

→ When the values of arguments are passed from the calling function to the called function, the values are copied into called function.

→ If any changes are made to the value in the called function, there is no change in the original value within the calling function.

Program

```
#include <stdio.h>
main()
{
  int n1, n2, x;
  int call_by_val();
  n1 = 6;
  n2 = 9;
  printf("n1 = %d & n2 = %d \n", n1, n2);
  x = call_by_val(n1, n2);
  printf("n1 = %d & n2 = %d \n", n1, n2);
  printf("x = %d", x);
}

call_by_val(int p1, int p2)
{
  int sum;
  sum = p1 + p2;
  p1 += 2;
  p2 += p1;
  printf("p1 = %d & p2 = %d \n", p1, p2);
  return(sum);
}
```

Output

n1 = 6 & n2 = 9
p1 = 8 & p2 = 17
n1 = 6 & n2 = 9
x = 15

Call by Reference :

In this method, the actual values are not passed, instead their address are passed. There is no copying of values since their memory locations are referenced.

* If any modification is made to the values in the called function, the original values get changed within calling function.

```
# include<stdio.h>
void main()
{
    int a = 10, b = 20;
    swapr(&a, &b);
    printf("\n a = %d b = %d", a, b);
}

swapr(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
```

output :
a = 20 b = 10;

Recursion :

* It is possible for a function to call itself. A function is called 'recursive' if a statement within the body of a function calls the same function
* Recursion is thus the process of defining something in terms of itself.

Program

```
# include <stdio.h>
int facto(int);
void main()
    {
     int n, answer;
    printf("Enter a number");
     scanf("%d", &n);
      answer = facto(n);
     printf("factorial is %d", answer.);
    }
```

```
jacto (int n)
  { int factorial;
    if (n==1)
    {
    return (1);
    }
    else
        { factorial = n * jacto(n-1);
          return (factorial);
        }
  }
```

Output:

Enter a number 5
Factorial is 120