Functions

By

C.Mahesh

Definition:

➔ Functions is basically a set of statements that takes inputs, perform some computation and produces output.

Syntax: Return_type function_name(set_of_inputs)

Note:Return type provides the type of output.

Why functions:

- **Reusability:** Once function is defined, it can be reused over and over again.
- **Abstraction:** If you are just using function in your program then you don't have to worry about how it works inside.
  - Ex: scanf function

Function Declaration in C:

When we declare a variable, we declare its properties to the compiler.
Ex: int students;
Properties:
➔ 1.Name of the variable: students
➔ 2.Type of variable: int
Similarly, function declaration (also called function prototype)means declaring the properties of a function to the compiler.
Ex: **int fun(int, char);**
Properties:

- 1.Name of the function:
- 2.Return type of function:
- 3. Number of parameters:
- 4.Type of parameter 1:
- 5.Type of parameter 2:

Function definition in C:

Function definition consists of block of code which is capable of performing some specific task.

```
int add(int a, int b)
{
    int sum;
    sum = a + b;
    return sum;
}
```

```
#include<stdio.h>

int add(int, int);          //Dont forget to mention the prototype of a  function
                            //No need to mention names of the parameters.

int main()
{
        int m=30, n=40, sum;
        sum = add(m,n);                     //calling function
        printf("sum is %d",sum);
}

int add(int a, int b)       //This is the way to mention both data type and name of parameters.
{
        return (a+b);
}
```

Function Return type:

- Every C function must specify the type of data being generated by the function.
- If the function does not generate any data, its return type can be “void”.
- The type of expression returned must match the type of the function, or be capable of being converted, otherwise we can see undefined behaviour.

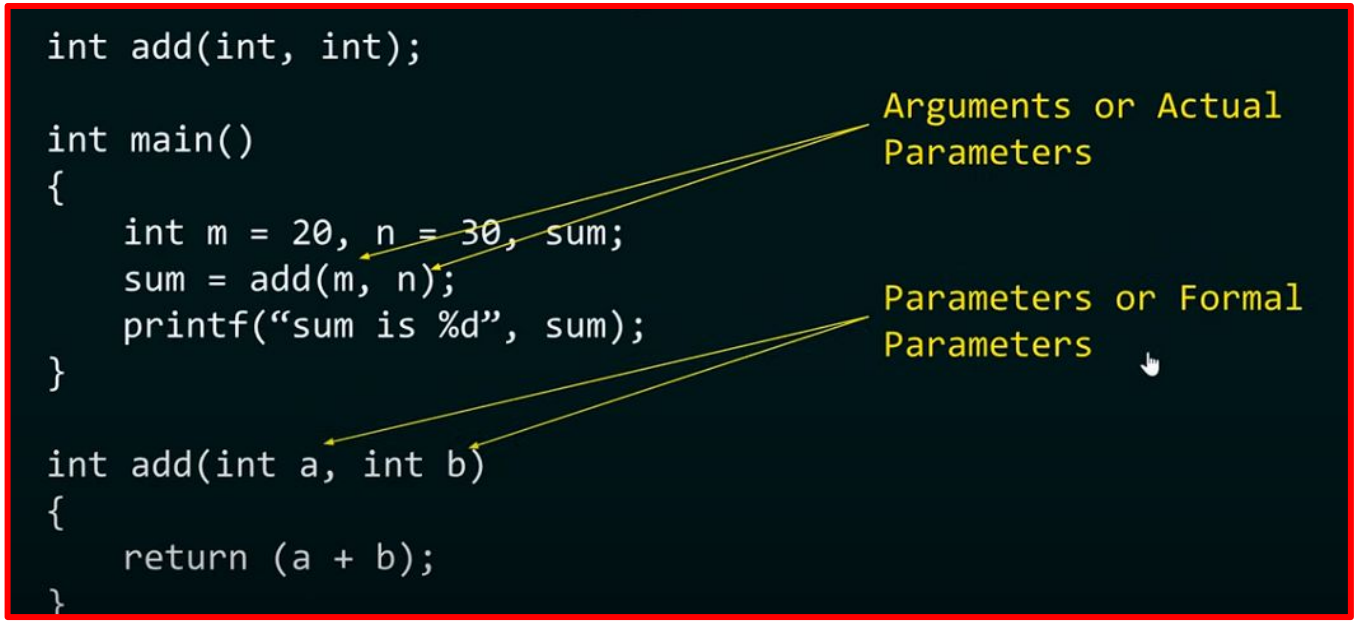
Example:

```
void print_happy_birthday( int age )
   {
     printf("Congratulations on your %d th Birthday\n", age);
     return;
    }
```

Difference between ARGUMENT AND A PARAMETER:

Parameter: is a variable in the declaration and definition in the function.
Argument: is the actual value of parameter that gets passed to the function.

Note: **Parameter** is also called as **Formal Parameter** and **Argument** is also called as **Actual Parameter.**



CALL BY VALUE AND CALL BY REFERENCE:

Actual Parameters: The parameters passed to a function
Formal Parameters: The parameters received by a function.

CALL BY VALUE: Here the actual parameters will be copied to formal parameters and these two different parameters stores the values in different locations.

```
int x = 10, y = 20;
fun(x, y);
printf("x = %d, y = %d", x, y);
```

```
int fun(int x, int y)
{
    x = 20;
    y = 10;
}
```

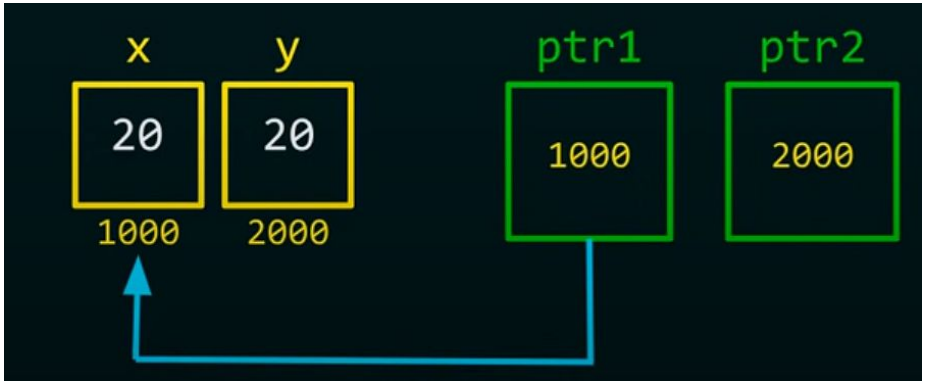
CALL BY REFERENCE: Here both actual and formal parameters refers to same memory location.
So any changes made to the formal parameters will get reflected to actual parameters.

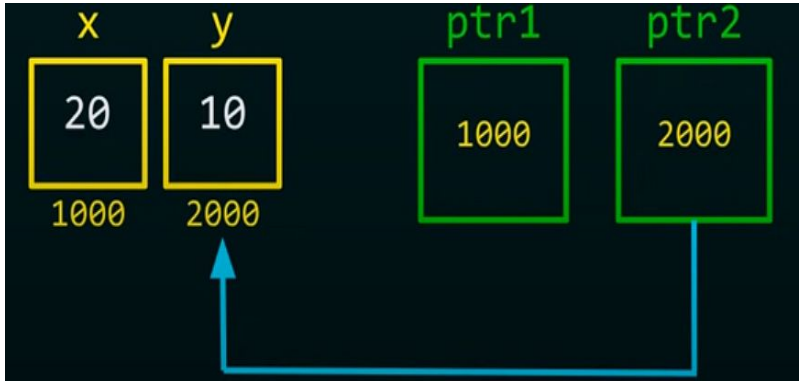
Note: Here instead of passing values, we pass addresses.

```
int x = 10, y = 20;
fun(&x, &y);
printf("x = %d, y = %d", x, y);
```

```
int fun(int *ptr1, int *ptr2)
{
    *ptr1 = 20;
    *ptr2 = 10;
}
```

ptr -> dereference operator -> access the value from that particular address.





Point	Call by Value	Call by Reference
Copy	Duplicate Copy of Original Parameter is Passed	Actual Copy of Original Parameter is Passed
Modification	No effect on Original Parameter after modifying parameter in function	Original Parameter gets affected if value of parameter changed inside function

Recursion

Recursion:

Recursion is a process in which a function calls itself directly or indirectly.

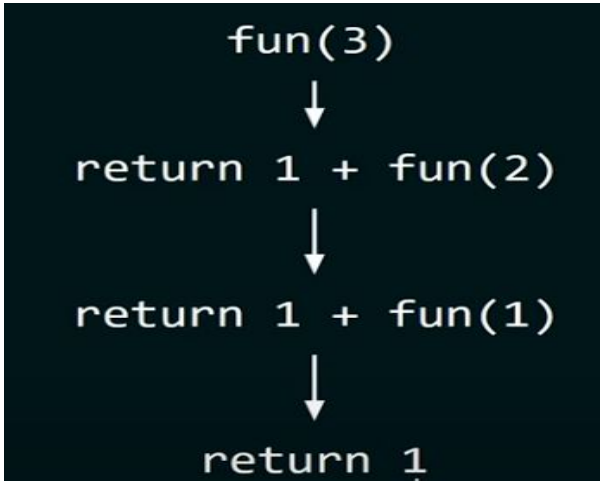
For example

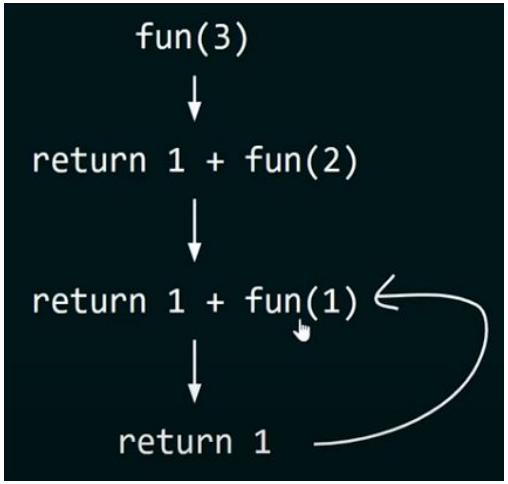
```
int fun()
{
    ...
    fun();
}
```

->Program to demonstrate Recursion:

```
int fun(int n)
{
    if( n==1 )
        return 1;
    else
        return 1 + fun( n-1 );
}

int main() {
    int n = 3;
    printf("%d", fun(n));
    return 0;
}
```





return 1 -> return back to caller.

What is the output of below C program.

```c
#include<stdio.h>
int fun(int n)
{
        if(n==0)
        {
                return 1;
        }
        else
        {
                return 7 + fun(n-2);
        }
}
        int main()
        {
                printf("%d" , fun(4));
                return 0;
        }
```

```
void fun1(int n)
{
    if(n>0)
    {
        printf("%d", n);
        fun1(n-1);
    }
}
```

```
void main()
{
    int x=3;
    fun1(x);
}
```

```
void fun(int n)
{
    if(n>0)
    {
        1. Calling
        2. fun(n-1)
        3. Returning
    }
}
```

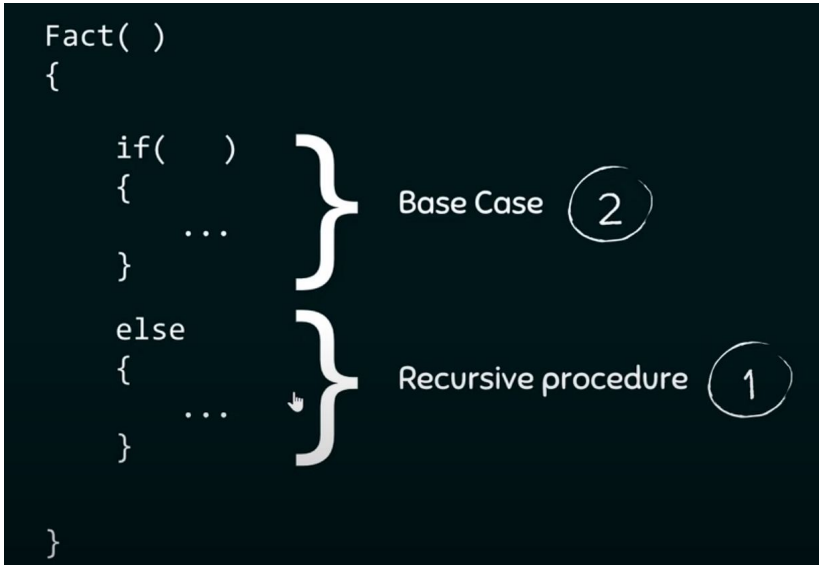
```
void fun2(int n)
{
    if(n>0)
    {
    1. fun2(n-1);
    2. printf("%d", n)
    }
}
```

```
void main()
{
    int x=3;
    fun2(x);
}
```

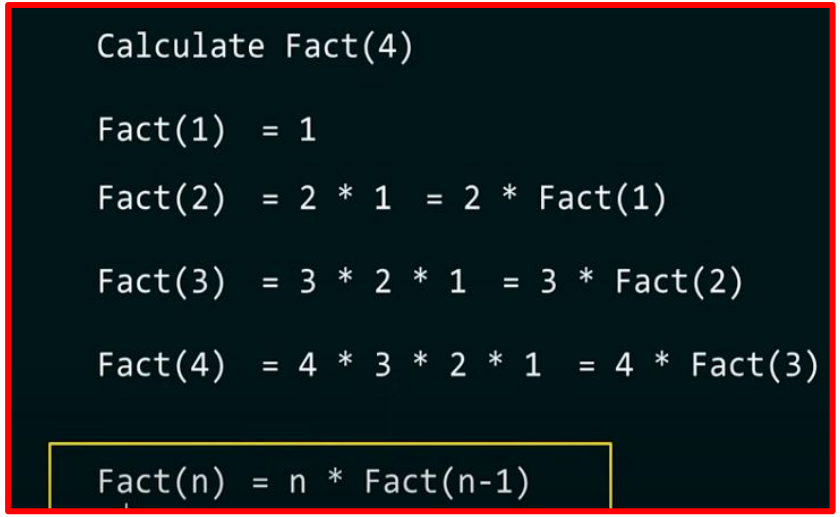
How to write Recursive Functions:

1.Divide the problem into smaller sub-problems
2.Specify the base condition to stop the recursion.

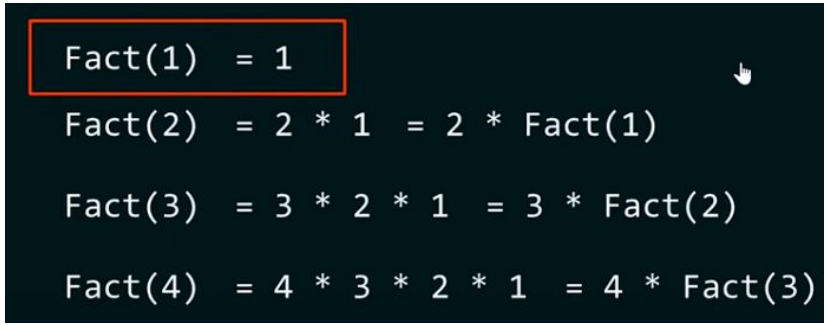
Ex: calculate factorial of n.
Fact of 5 -> 5*4*3*2*1 = 120.
Basic structure for Recursion.



1.Divide the problem into sub-problems.



2.Specify the base condition for stop the recursion.: base condition one which doesn’t require call to the same function again & it helps to the stopping the recursion.



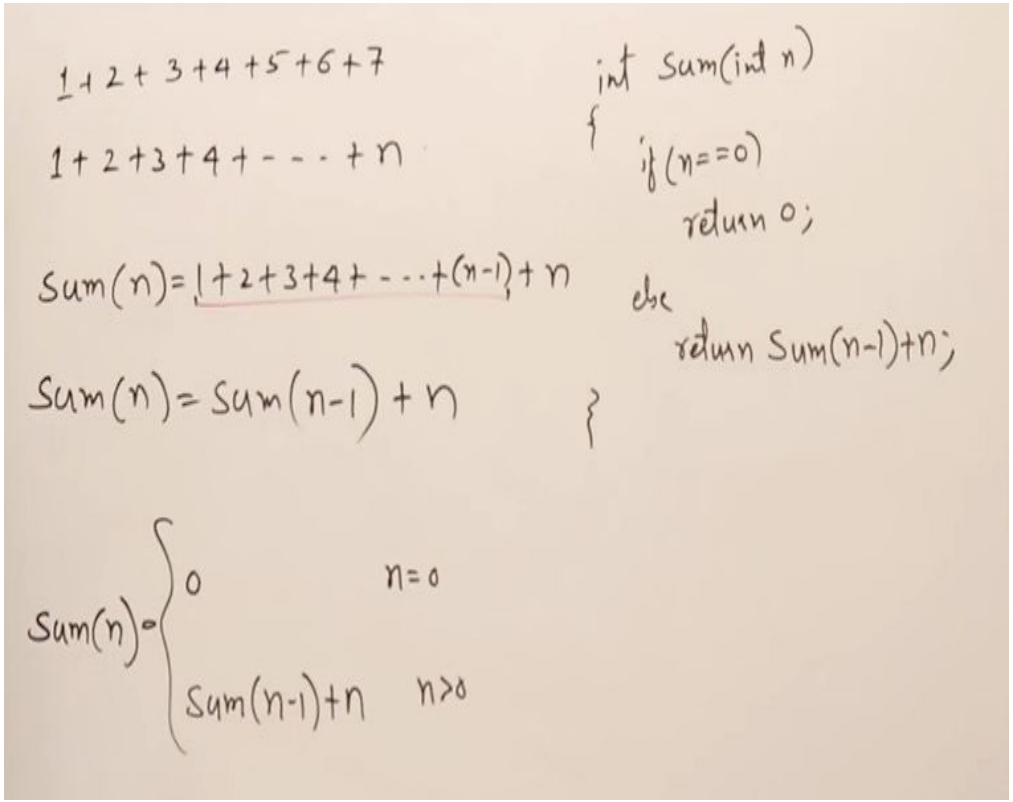
```
Fact(int n)
{

    if( n == 1 )
    {
        return 1;
    }

    else
    {
        return n * Fact(n-1);
    }


}
```

Sum of First ‘n’ natural numbers.




```
int sum(int n)
{
    return n*(n+1)/2;
}
```

```c
#include<stdio.h>
void add();                                          //Function Declaration
int main()
{
  add();                                             //Function Calling
  return 0;
}
void add()                                           //Function Definition
{
   int a,b,c;
   printf("\nEnter The Two values:");
   scanf("%d%d",&a,&b);
   c=a+b;
   printf("Addition:%d",c);
}
```

This program is about print the an example for **no return value and with argument using functions.**

```
#include<stdio.h>
void add(int,int);
int main()
{
   int a,b;
   printf("\nEnter The Two Values:");
   scanf("%d%d",&a,&b);
   add(a,b);
   return 0;
}
void add(int a,int b)
{
   int c;
   c=a+b;
   printf("\nAddition:%d",c);
}
```

This program is about example for **return type without arguments using functions.**

```c
#include<stdio.h>
int add();
int main()
{
   printf("\nAddition:%d",add());
   return 0;
}
int add()
{
   int a,b,c;
   printf("\nEnter The Two Values:");
   scanf("%d%d",&a,&b);
   c=a+b;
   return c;
}
```

This program is about example **for return type with arguments using functions.**

```
#include<stdio.h>
int add(int,int);
int main()
{
   int a,b;
   printf("\nEnter The Two Values:");
   scanf("%d%d",&a,&b);
   printf("\nAddition:%d",add(a,b));
   return 0;
}
int add(int a,int b)
{
   int c;
   c=a+b;
   return c;
}
```

Factorial using functions.

```c
int main()
{

    printf("Enter a Number to Find Factorial: ");
    printf("\nFactorial of a Given Number is: %d ",fact());
    return 0;
}
int fact()
{
    int i,fact=1,n;
    scanf("%d",&n);
    for(i=1; i<=n; i++)
    {
        fact=fact*i;
    }
    return fact;
}
```