## C Functions:

- A function is a **block of statements** that performs a specific task.
- Let's say you are writing a C program and you need to perform a same task in that program more than once. In such case you have two options:
    a) Use the same set of statements every time you want to perform the task.
    b) Create a function to perform that task, and just call it every time you need to perform that task.
- Using option (b) is a good practice and a **good programmer always uses functions while writing code in C**.
- The function contains the set of programming statements enclosed by {}.
- A function can be called multiple times to provide **reusability** and **modularity** to the C program.

**Syntax of a function**

```
return_type function_name (argument list)
{
    Set of statements – Block of code
}
```

**Types of functions**

### 1) Predefined standard library functions:

**Standard library functions** are also known as **built-in functions**. Functions such as **puts(), gets(), printf(), scanf()** etc are standard library functions. These functions are already defined in header files (files with **.h extensions are called header files** such as stdio.h), so we just call them whenever there is a need to use them.

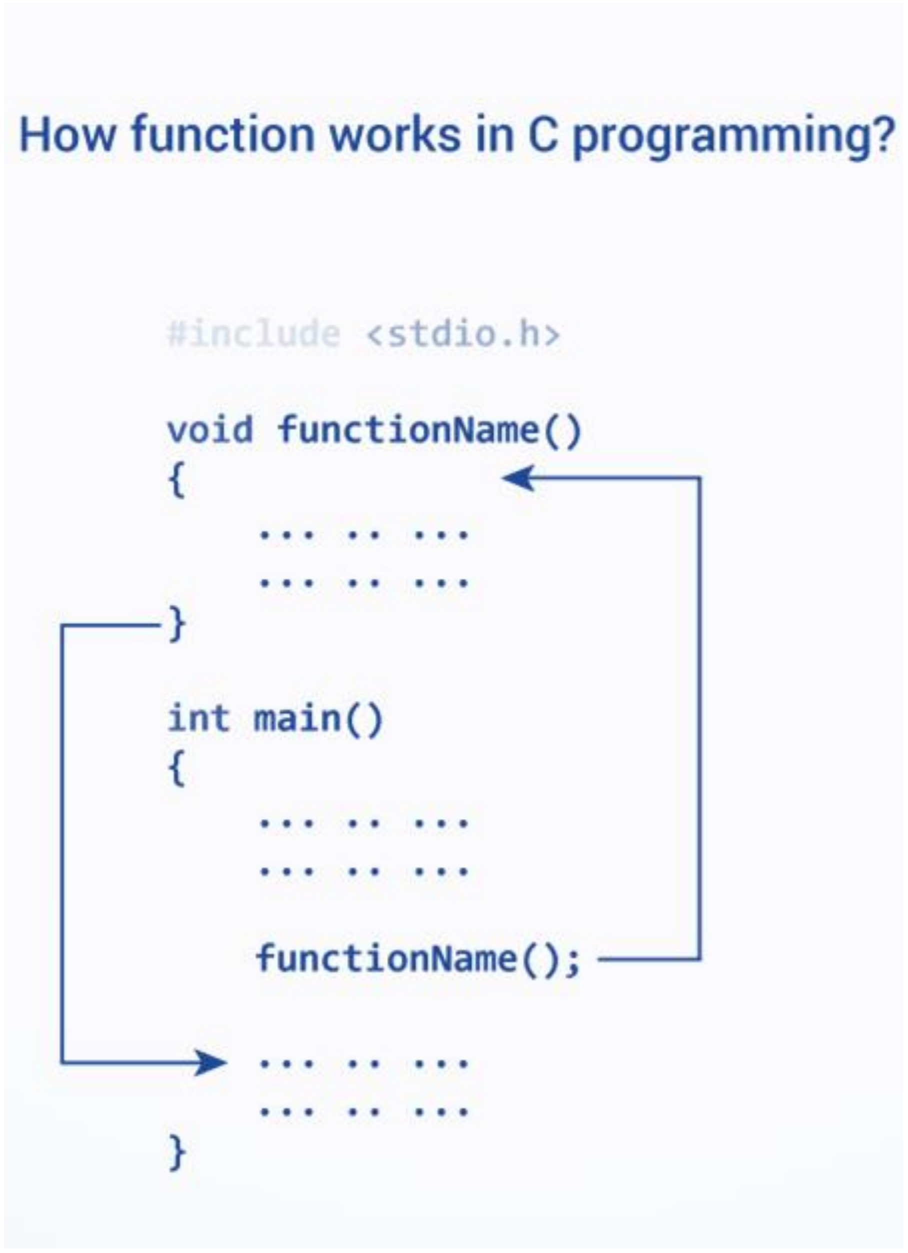
**For example**,

printf() function is defined in <stdio.h> header file so in order to use the printf() function, we need to include the <stdio.h> header file in our program using #include <stdio.h>.

### 2) User Defined functions:

You can also create functions as per your need. **Such functions created by the user** are known as user-defined functions.

```
#include <stdio.h>
void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...
    functionName();
    ... .. ...
    ... .. ...
}
```



```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

**Example:**

```
#include<stdio.h>
void printHello();
void main ()
{
    printf("Hello ");
    printName();
}
void printHello()
{
    printf("World");
}
```

**OUTPUT:**

```
Hello World
```

**Example1: Creating a user defined function addition()**

```
#include <stdio.h>
int addition(int num1, int num2)
{
     int sum;
     sum = num1+num2;
     return sum;
}

int main()
{
     int var1, var2;
     printf("Enter number 1: ");
     scanf("%d",&var1);
     printf("Enter number 2: ");
     scanf("%d",&var2);
     int res = addition(var1, var2);
     printf ("Output: %d", res);

     return 0;
}
```

**Output**

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

**Function Arguments:**

If a function is **to use arguments**, it **must declare variables that accept the values** of the arguments. These variables are called the **formal parameters** of the function.

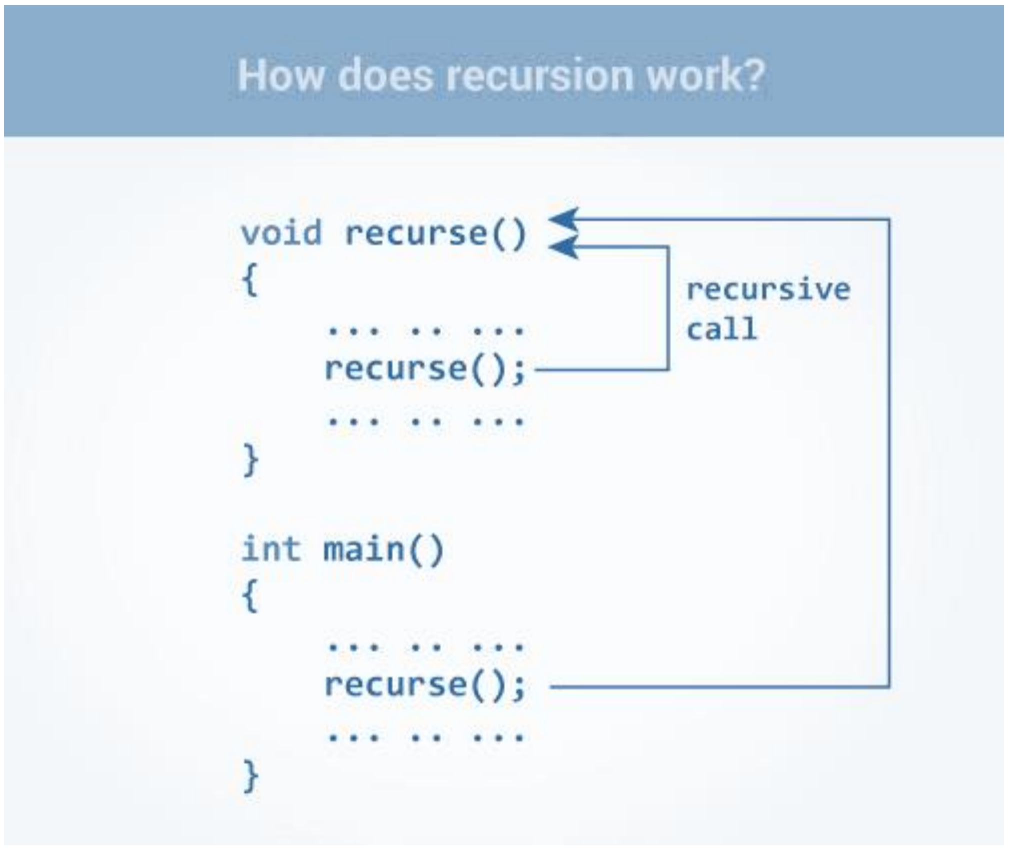
While calling a function, there are two ways in which arguments can be passed to a function −

| *Sr.No.* | *Call Type & Description* |
|---|---|
| *1* | **Call by value**<br>This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument. |
| *2* | **Call by reference**<br>This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument. |

**Note: By default**, C uses **call by value** to pass arguments.

## C Recursion

- The process in which a **function calls itself directly or indirectly is called recursion** and the corresponding function is called as recursive function.
- Using recursive algorithm, certain problems can be solved quite easily.
- Recursion involves **several numbers of recursive calls.**
- The **recursion continues until some condition is met** to prevent it.
- **To prevent infinite recursion, if-else statement can be used** where one branch makes the recursive call, and other doesn't.



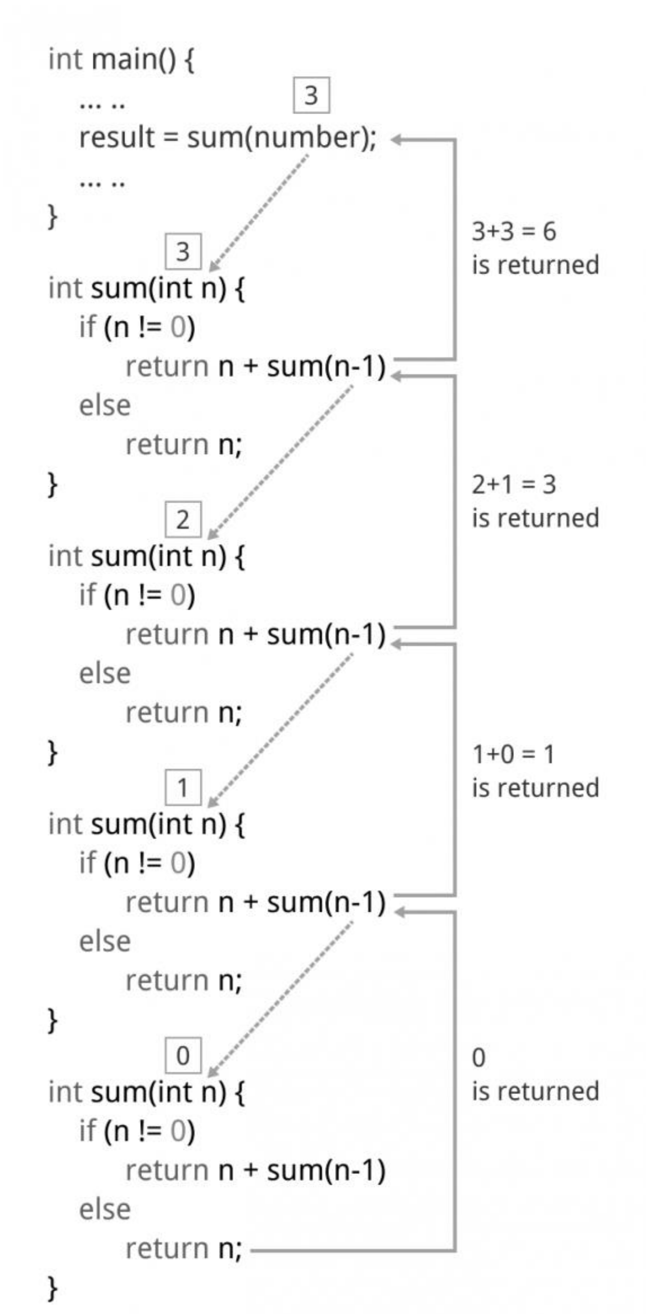
```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

```
int fact(int n)
{
    if (n < = 1) // base case
        return 1;
    else
        return n*fact(n-1);
}
```

In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.

## Sum of Natural Numbers Using Recursion

```c
#include <stdio.h>
int sum(int n);

int main() {
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}

int sum(int n) {
    if (n != 0)
// sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}
```



```c
int main() {
  ... ..
  result = sum(number);
  ... ..
}
int sum(int n) {
  if (n != 0)
     return n + sum(n-1)
  else
     return n;
}
```

## Advantages and Disadvantages of Recursion

Recursion makes program elegant. However, if performance is vital, use loops instead as recursion is usually much slower.

## C Storage Class

A storage class defines the scope means visibility and life-time of variables and/or functions within a C Program.

We have four different storage classes in a C program –

- auto
- register
- static
- extern

### The auto Storage Class

The **auto** storage class is the default storage class for all local variables.

```
{
   int mount;
   auto int month;
}
```

### The register Storage Class

The **register** storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
   register int  miles;
}
```

### Local Variable

The variables declared inside a block are automatic or local variables. The local variables exist only inside the block in which it is declared.

Example:

```
#include <stdio.h>

int main(void) {

  for (int i = 0; i < 5; ++i) {
     printf("C programming");
  }

 // Error: i is not declared at this point
  printf("%d", i);
  return 0;
}
```

## Global Variable / Extern

Variables that are declared outside of all functions are known as external or global variables. They are accessible from any function inside the program.

**Example**

```c
#include <stdio.h>
void display();

int n = 5;  // global variable

int main()
{
    ++n;
    display();
    return 0;
}

void display()
{
    ++n;
    printf("n = %d", n);
}
```

Output

```
n = 7
```

## Static Variable:

A static variable is declared by using the static keyword. For example;

static int i;

The static variables are used within function/ file as local static variables. They can also be used as a global variable

- Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.
- Static global variables are global variables visible only to the file in which it is declared.

**Example**

```c
#include <stdio.h>
void display();

int main()
{
    display();
    display();
}
void display()
{
    static int c = 1;
    c += 5;
    printf("%d  ",c);
}
```

**Output**

```
6 11
```