# Functions in C++

A function is a block of code that performs a specific task.

Suppose we need to create a program to create a circle and color it. We can create two functions to solve this problem:

- a function to draw the circle
- a function to color the circle

Dividing a complex problem into smaller chunks makes our program easy to understand and reusable.

There are two types of function:

1. **Standard Library Functions:** Predefined in C++
2. **User-defined Function:** Created by users

## ○ User-defined Function:

C++ allows the programmer to define their own function.

A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).

When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.

### ➢ Function Declaration:

The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

Here's an example of a function declaration.

```
// function declaration
void greet() {
    cout << "Hello World";
        }
```

Here,

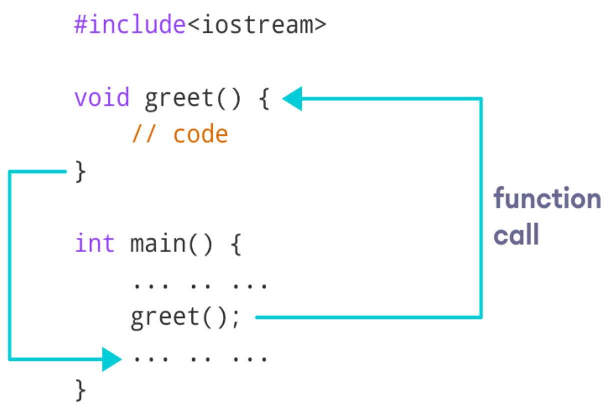
- the name of the function is greet()
- the return type of the function is void

- the empty parentheses mean it doesn't have any parameters
- the function body is written inside {}

➢ Calling a Function:

- In the above program, we have declared a function named greet().
- To use the greet() function, we need to call it.
- Here's how we can call the above greet() function.

```
int main() {
// calling a function
 greet();
   }
```



```
#include<iostream>

void greet() {
    // code
}

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

➢ Function Parameters:

A function can be declared with parameters (arguments). A parameter is a value that is passed when declaring a function.

For example, let us consider the function below:

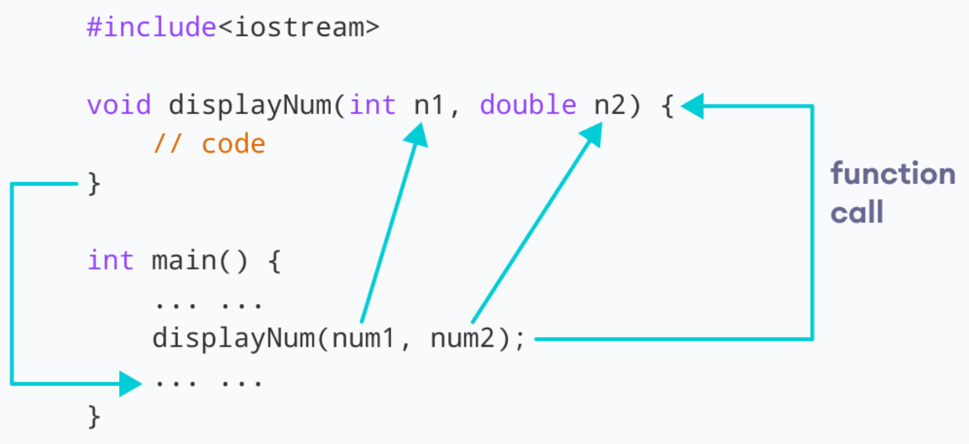
```
void printNum(int num) {
    cout << num;
}
```

Here, the int variable num is the function parameter.

We pass a value to the function parameter while calling the function.

```
    int main() {
    int n = 7;
    // calling the function
    // n is passed to the function as argument
    printNum(n);
  return 0;
}
```

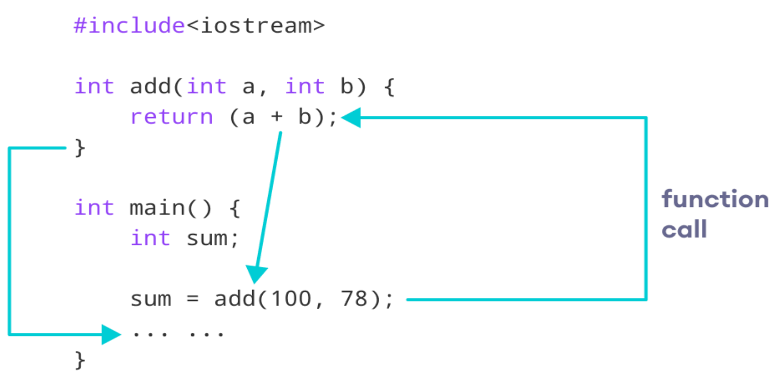
- Function with Parameters:



```
#include<iostream>

void displayNum(int n1, double n2) {
    // code
}

int main() {
    ... ...
    displayNum(num1, num2);
    ... ...
}
```

The type of the arguments passed while calling the function must match with the corresponding parameters defined in the function declaration.

- Return Statement:



```
#include<iostream>

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

int main() {
    int sum;

    sum = add(100, 78);
    ... ...
}
```

➢ Function Prototype:

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype.

For example,

```
// function prototype
void add(int, int);
int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}
// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

In the above code, the function prototype is:

```
void add(int, int);
```

This provides the compiler with information about the function name and its parameters. That's why we can use the code to call a function before the function has been defined.

The syntax of a function prototype is:

```
returnType functionName(dataType1, dataType2, ...);
```

○ Advantages of User Defined Functions:

- Functions make the code reusable. We can declare them once and use them multiple times.
- Functions make the program easier as each small task is divided into a function.
- Functions increase readability.

## Library Functions

- Library functions are the built-in functions in C++ programming.
- Programmers can use library functions by invoking the functions directly; they don't need to write the functions themselves.
- Some common library functions in C++ are sqrt(), abs(), isdigit(), etc.

- In order to use library functions, we usually need to include the header file in which these library functions are defined.
- For instance, in order to use mathematical functions such as `sqrt()` and `abs()`, we need to include the header file `cmath`.

## Types of User-defined functions:

user-defined functions can be categorized as:

- Function with no argument and no return value
- Function with no argument but return value
- Function with argument but no return value
- Function with argument and return value

1. Function with no argument and no return value:

```
# include <iostream>
using namespace std;

void prime();

int main()
{
    // No argument is passed to prime()
    prime();
    return 0;
}


// Return type of function is void because value is not returned.
void prime()
{

    int num, i, flag = 0;

    cout << "Enter a positive integer enter to check: ";
    cin >> num;

    for(i = 2; i <= num/2; ++i)
    {
        if(num % i == 0)
        {
            flag = 1;
            break;
        }
    }

    if (flag == 1)
    {
        cout << num << " is not a prime number.";
    }
    else
    {
        cout << num << " is a prime number.";
    }
}
```

2. Function with no argument but return value:

```
#include <iostream>
using namespace std;

int prime();

int main()
{
    int num, i, flag = 0;

    // No argument is passed to prime()
    num = prime();
    for (i = 2; i <= num/2; ++i)
    {
        if (num%i == 0)
        {
            flag = 1;
            break;
        }
    }

    if (flag == 1)
    {
        cout<<num<<" is not a prime number.";
    }
    else
    {
        cout<<num<<" is a prime number.";
    }
    return 0;
}

// Return type of function is int
int prime()
{
    int n;

    printf("Enter a positive integer to check: ");
    cin >> n;

    return n;
}
```

3. Function with argument but no return value:

```
#include <iostream>
using namespace std;

void prime(int n);

int main()
{
    int num;
    cout << "Enter a positive integer to check: ";
    cin >> num;

    // Argument num is passed to the function prime()
    prime(num);
    return 0;
}

// There is no return value to calling function. Hence, return type of function is void. */
void prime(int n)
{
    int i, flag = 0;
    for (i = 2; i <= n/2; ++i)
    {
        if (n%i == 0)
        {
            flag = 1;
            break;
        }
    }

    if (flag == 1)
    {
        cout << n << " is not a prime number.";
    }
    else {
        cout << n << " is a prime number.";
    }
}
```

4. Function with argument and return value:

```cpp
#include <iostream>
using namespace std;

int prime(int n);

int main()
{
    int num, flag = 0;
    cout << "Enter positive integer to check: ";
    cin >> num;

    // Argument num is passed to check() function
    flag = prime(num);

    if(flag == 1)
        cout << num << " is not a prime number.";
    else
        cout<< num << " is a prime number.";
    return 0;
}

/* This function returns integer value.  */
int prime(int n)
{
    int i;
    for(i = 2; i <= n/2; ++i)
    {
        if(n % i == 0)
            return 1;
    }

    return 0;
}
```

## Function Overloading

In C++, two functions can have the same name if the number and/or type of arguments passed is different.

These functions having the same name, but different arguments are known as overloaded functions.

For example:

```cpp
// same number different arguments

int test() { }
int test(int a) { }
float test(double a) { }
int test(int a, double b) { }
```

Here, all 4 functions are overloaded functions.

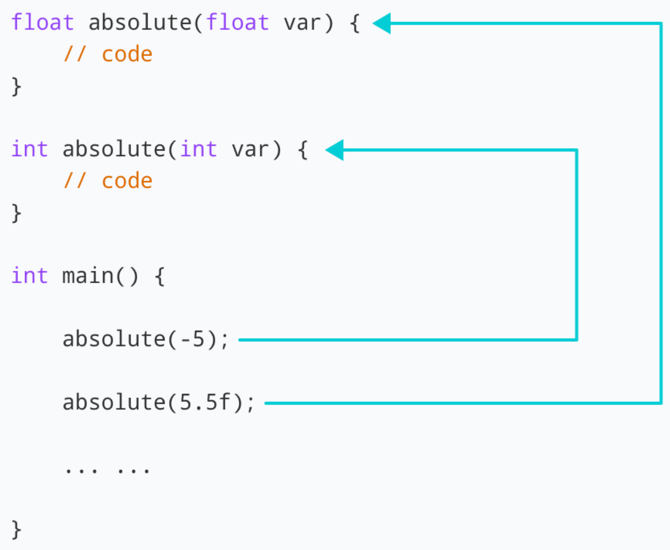
Notice that the return types of all these 4 functions are not the same. Overloaded functions may or may not have different return types, but they must have different arguments.

For example,

```
// Error code
int test(int a) { }
double test(int b){ }
```

Here, both functions have the same name, the same type, and the same number of arguments. Hence, the compiler will throw an error.

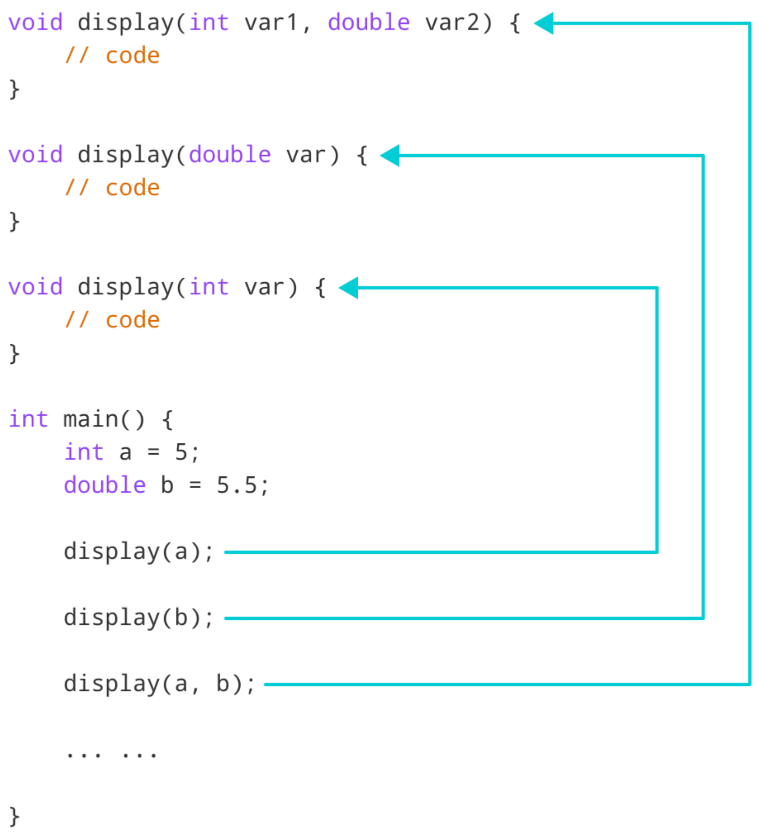
1. <u>Function Overloading using Different Types of Parameters:</u>



```
float absolute(float var) {
    // code
}

int absolute(int var) {
    // code
}

int main() {

    absolute(-5);

    absolute(5.5f);

    ... ...

}
```

In this program, we overload the `absolute()` function. Based on the type of parameter passed during the function call, the corresponding function is called.

2. Function Overloading using Different Number of Parameters:



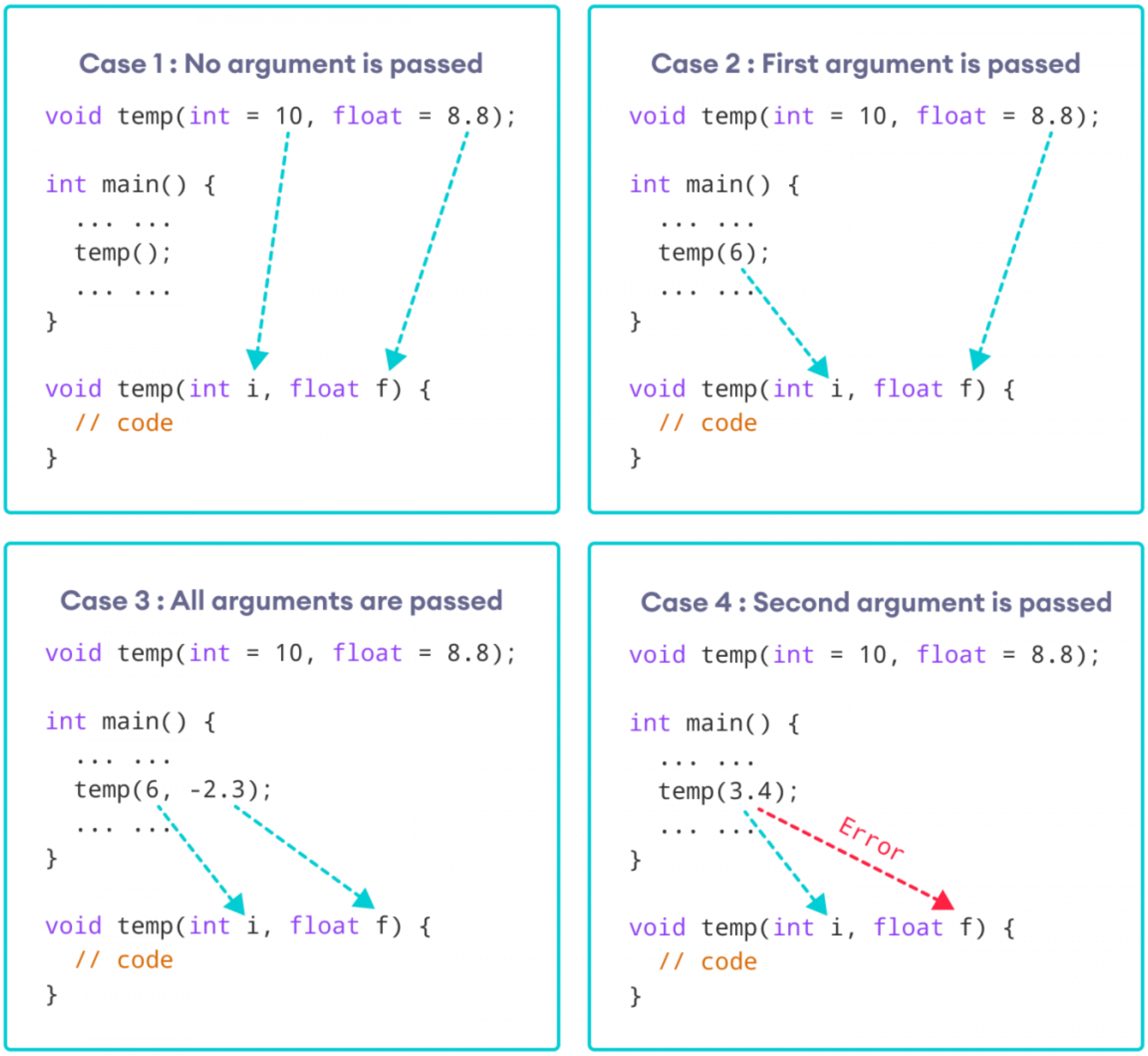
```
void display(int var1, double var2) {
    // code
}

void display(double var) {
    // code
}

void display(int var) {
    // code
}

int main() {
    int a = 5;
    double b = 5.5;

    display(a);

    display(b);

    display(a, b);

    ... ...

}
```

- The return type of all these functions is the same but that need not be the case for function overloading.
- In C++, many standard library functions are overloaded. For example, the sqrt() function can take double, float, int, etc. as parameters. This is possible because the sqrt() function is overloaded in C++.

## Default Argument

- In C++ programming, we can provide default values for “function parameters”.
- If a function with default arguments is called without passing arguments, then the default parameters are used.
- However, if arguments are passed while calling the function, the default arguments are ignored.

For Example, Default Arguments in C++



Case 1 : No argument is passed

```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp();
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Case 2 : First argument is passed

```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp(6);
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Case 3 : All arguments are passed

```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp(6, -2.3);
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Case 4 : Second argument is passed

```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp(3.4);
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Points to Remember for Default Arguments:

1. Once we provide a default value for a parameter, all subsequent parameters must also have default values.
2. If we are defining the default arguments in the function definition instead of the function prototype, then the function must be defined before the function call.

## Storage Classes

Every variable in C++ has two features: type and storage class.

Type specifies the type of data that can be stored in a variable. For example: `int`, `float`, `char` etc.

And, storage class controls two different properties of a variable: lifetime (determines how long a variable can exist) and scope (determines which part of the program can access it).

Depending upon the storage class of a variable, it can be divided into 4 major types:

- Local variable
- Global variable
- Static local variable
- Register Variable
- Thread Local Storage

1. **Local Variable:**

A variable defined inside a function (defined inside function body between braces) is called a local variable or automatic variable.

Its scope is only limited to the function where it is defined. In simple terms, local variable exists and can be accessed only inside a function.

The life of a local variable ends (It is destroyed) when the function exits.

Example for Local Variable:

```
#include <iostream>
using namespace std;
void test();
int main() {
    // local variable to main()
    int var = 5;
    test();
    // illegal: var1 not declared inside main()
    var1 = 9;
}
void test(){
    // local variable to test()
    int var1;
    var1 = 6;
   // illegal: var not declared inside test()
    cout << var;
}
```

The variable `var` cannot be used inside `test()` and `var1` cannot be used inside `main()` function.

Keyword `auto` was also used for defining local variables before as: `auto int var;`

But, after C++11 `auto` has a different meaning and should not be used for defining local variables.

Reference go through the below link:

https://en.cppreference.com/w/cpp/language/auto

2. Global Variable:

- If a variable is defined outside all functions, then it is called a global variable.
- The scope of a global variable is the whole program. This means, It can be used and changed at any part of the program after its declaration.
- Likewise, its life ends only when the program ends.

Example program for Global Variable:

```
#include <iostream>
using namespace std;
// Global variable declaration
int c = 12;
void test();
int main(){
    ++c;
    // Outputs 13
    cout << c <<endl;
    test();
    return 0;
}
void test(){
    ++c;
    // Outputs 14
    cout << c;
}
```

In the above program, `c` is a global variable.

This variable is visible to both functions `main()` and `test()` in the above program.

3. Static Local Variable:

Keyword `static` is used for specifying a static variable.

For example:

```
... .. ...
int main()
{
  static float a;
  ... .. ...
}
```

A static local variable exists only inside a function where it is declared (similar to a local variable) but its lifetime starts when the function is called and ends only when the program ends.

The main difference between local variable and static variable is that, the value of static variable persists the end of the program.

Example for Static Local Variable:

```
#include <iostream>
using namespace std;
void test()
{
    // var is a static variable
    static int var = 0;
    ++var;
    cout << var << endl;
}
int main()
{
    test();
    test();
   return 0;
}
```

4. Register Variable:

Keyword register is used for specifying register variables.

Register variables are similar to automatic variables and exists inside a particular function only. It is supposed to be faster than the local variables.

If a program encounters a register variable, it stores the variable in processor's register rather than memory if available. This makes it faster than the local variables.

However, this keyword was deprecated in C++11 and should not be used.

5. Thread Local Storage:

Thread-local storage is a mechanism by which variables are allocated such that there is one instance of the variable per extant thread.

Keyword **thread_local** is used for this purpose.

For reference go through the link:

https://www.codeproject.com/Articles/8113/Thread-Local-Storage-The-C-Way

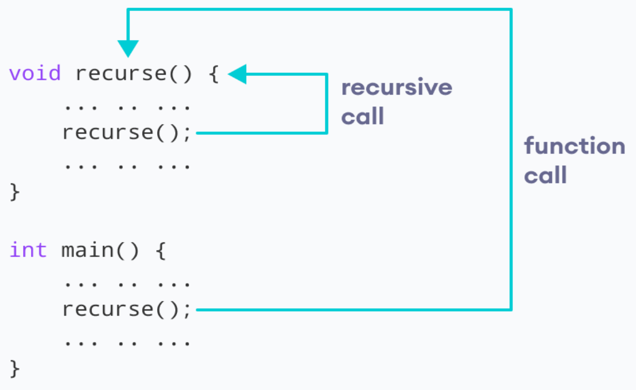
## Recursion in C++

A function that calls itself is known as a recursive function. And, this technique is known as recursion.

- Working of Recursion in C++:

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

The figure below shows how recursion works by calling itself repeatedly.


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

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

- The recursion continues until some condition is met.
- To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and the other doesn't.

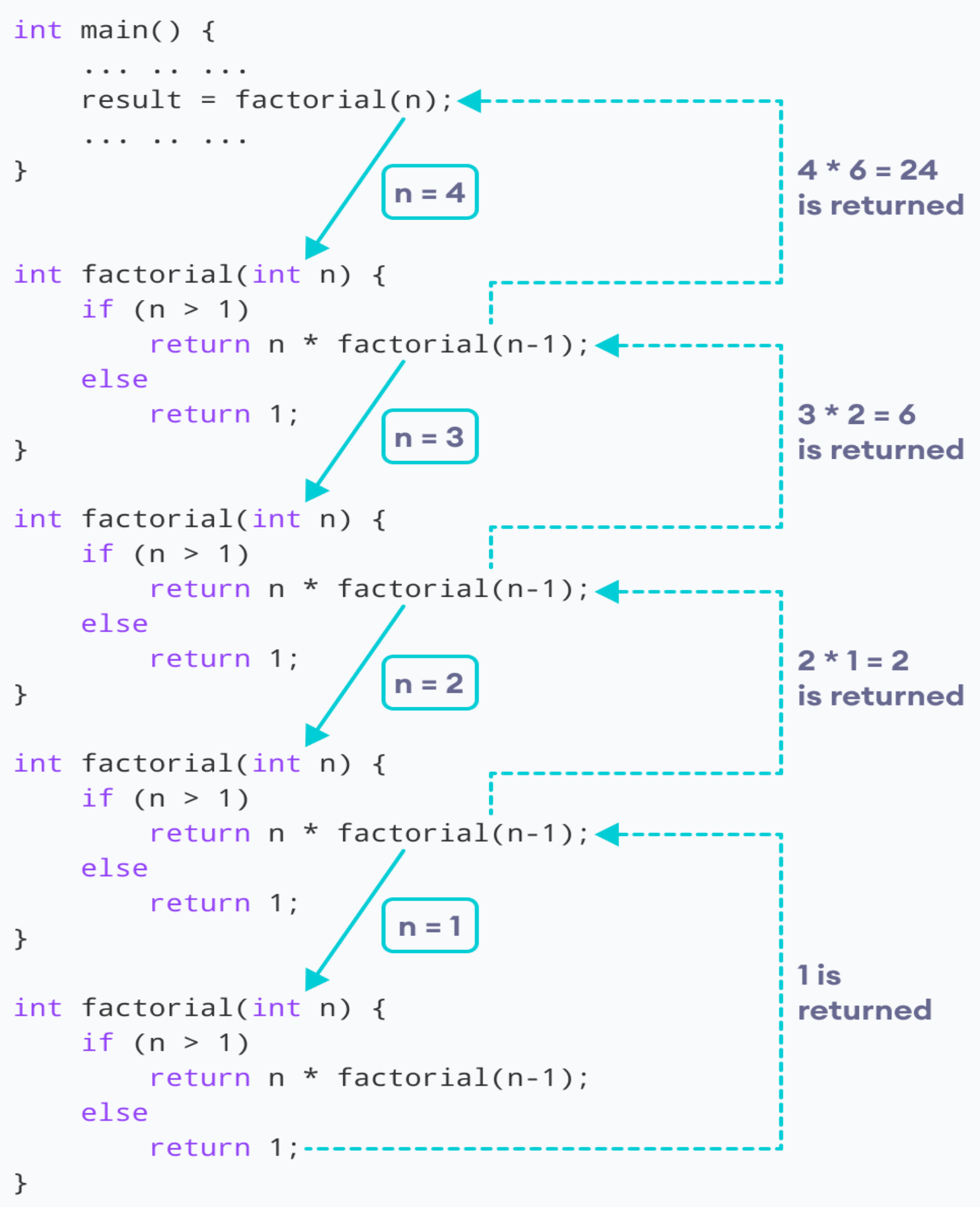
**Advantages of Recursion:**

- It makes our code shorter and cleaner.
- Recursion is required in problems concerning data structures and advanced algorithms, such as Graph and Tree Traversal.

**Disadvantages of Recursion:**

- It takes a lot of stack space compared to an iterative program.
- It uses more processor time.
- It can be more difficult to debug compared to an equivalent iterative program.

Factorial program is an example for recursion.



```
int main() {
    ... .. ...
    result = factorial(n);
    ... .. ...
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

int factorial(int n) {
    if (n > 1)
        return n * factorial(n-1);
    else
        return 1;
}

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

## Return by Reference in C++

In C++ Programming, not only can you pass values by reference to a function, but you can also return a value by reference. For this we must know about “Global Variables”.

Example for Return by Reference:

```
#include <iostream>
using namespace std;
// Global variable
int num;
// Function declaration
int& test();
int main(){
    test() = 5;
    cout << num;
    return 0; }
int& test(){
    return num; }
```

✓ Points to Remember:

- Ordinary function returns value, but this function doesn't. Hence, you cannot return a constant from the function.

  e.g.:

  ```
  int& test() {
      return 2;
  }
  ```

- You cannot return a local variable from this function.

  ```
  int& test()
  {
      int n = 2;
      return n;
  }
  ```