Recursive Function in C++

The process in which a function calls itself is known as recursion and the corresponding function is called the **recursive function.**

Syntax

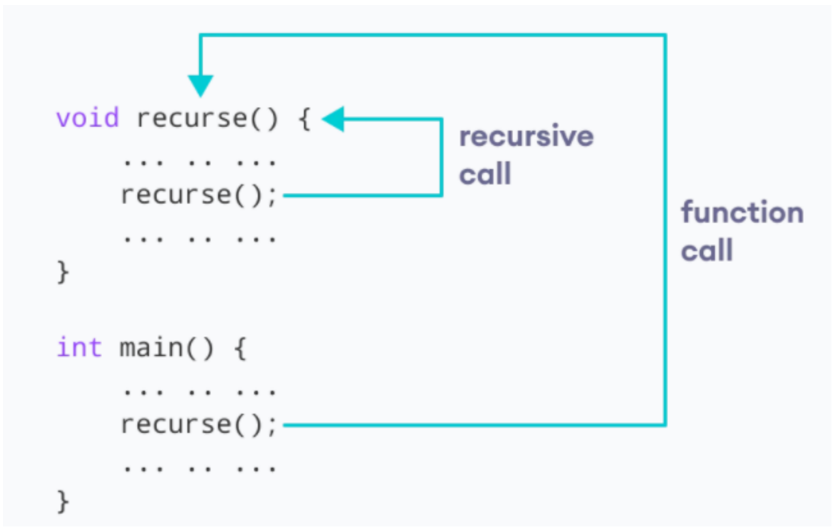
```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

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

Example

```
void recurse()
{
  recurse(); //Function calls itself
}

int main()
{
  recurse(); //Sets off the recursion
}
```



```
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.

Example

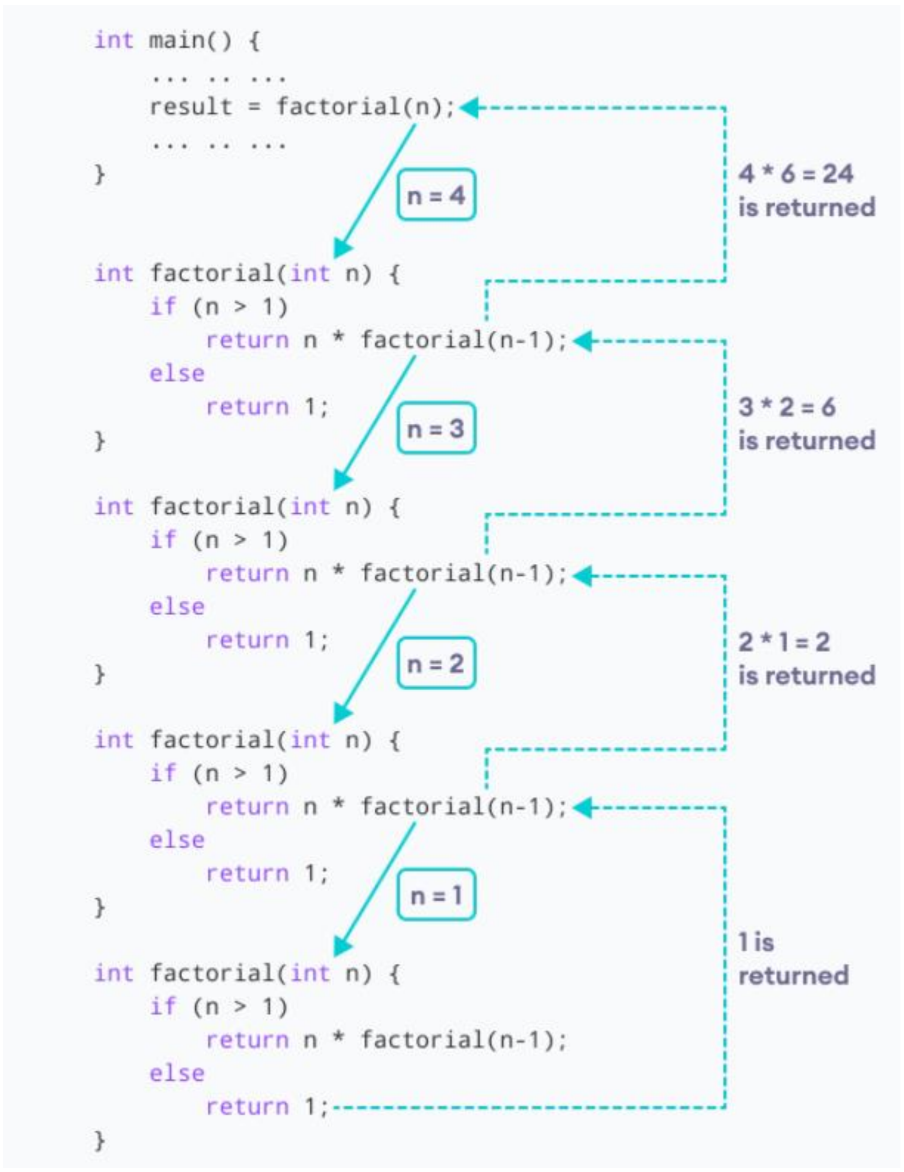
```
#include <iostream>
using namespace std;

//Factorial function
int factorial (int n){
    /* This is called the base condition, it is
    * very important to specify the base condition
    * in recursion, otherwise your program will throw
    * stack overflow error.
    */

   if (n <= 1)
        return 1;
   else
       return n*factorial (n-1);
}
int main(){
   int num;
   cout<<"Enter a number: ";
   cin>>num;
   cout<<"Factorial of entered number: "<< factorial(num);
   return 0;
}
```

Output:

```
Enter a number: 5
Factorial of entered number: 120
```



```
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;
}
```

As we can see, the factorial() function is calling itself. However, during each call, we have decreased the value of n by 1. When n is less than 1, the factorial() function ultimately returns the output.

Base condition

In the above program, you can see that I have provided a base condition in the recursive function. The condition is:

```
if (n <= 1)
        return 1;
```

The purpose of recursion is to divide the problem into smaller problems till the base condition is reached.

For example the above factorial program solves the factorial function `factorial(n)` by calling a smaller factorial function f(n-1), this happens repeatedly until the n value reaches base condition(`factorial(1)=1`). If you do not define the base condition in the recursive function then you will get. **stack overflow error**

Direct recursion vs indirect recursion

Direct recursion: When function calls itself, it is called direct recursion, the example we have seen above is a direct recursion example.

Indirect recursion: When function calls another function and that function calls the calling function, then this is called indirect recursion. For example: function A calls function B and Function B calls function A.

Indirect Recursion Example in C++

```
#include <iostream>
using namespace std;
int fa(int);
int fb(int);
int fa(int n){
   if(n<=1)
      return 1;
   else
      return n*fb(n-1);
}
int fb(int n){
   if(n<=1)
      return 1;
   else
      return n*fa(n-1);
}
int main(){
   int num=5;
   cout<<fa(num);
   return 0;
}
```

Output:

```
120
```

Structures in C++

Data structures

A *data structure* is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths.

```
syntax:
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

where `type_name` is a name for the structure type, `object_name` can be a set of valid identifiers for objects that have the type of this structure.

Right at the end of the `struct` definition, and before the ending semicolon (`;`), the optional field `object_names` can be used to directly declare objects of the structure type.

Example:

```
struct product {
  int weight;
  double price;
} ;

product apple;
product banana, melon;
```

This declares a structure type, called `product`, and defines it having two members: `weight` and `price`, each of a different fundamental type. This declaration creates a new type (`product`), which is then used to declare three objects (variables) of this type: `apple`, `banana`, and `melon`. Note how once `product` is declared, it is used just like any other type.

The structure objects `apple`, `banana`, and `melon` can be declared at the moment the data structure type is defined:

```
struct product {
  int weight;
  double price;
} apple, banana, melon;
```

In this case, where `object_names` are specified, the type name (`product`) becomes optional: `struct` requires either a `type_name` or at least one name in `object_names`, but not necessarily both.

It is important to clearly differentiate between what is the structure type name (`product`), and what is an object of this type (`apple`, `banana`, and `melon`). Many objects (such as `apple`, `banana`, and `melon`) can be declared from a single structure type (`product`).

Example:

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

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```

How to access structure elements?

Structure members are accessed using dot (.) operator.

Example

```
apple.weight
apple.price
banana.weight
banana.price
melon.weight
melon.price
```

Each one of these has the data type corresponding to the member they refer to: apple.weight, banana.weight,and melon.weight are of type int, while `apple.price`, `banana.price`, and `melon.price` are of type double.

Example:

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

struct Person
{
    char name[50];
    int age;
    float salary;
};

int main()
{
    Person p1;

    cout << "Enter Full name: ";
    cin.get(p1.name, 50);
    cout << "Enter age: ";
    cin >> p1.age;
    cout << "Enter salary: ";
    cin >> p1.salary;

    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p1.name << endl;
    cout <<"Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;

    return 0;
}
```

Here a structure `Person` is declared which has three members name, `age` and `salary`. Inside `main()` function, a structure variable `p1` is defined. Then, the user is asked to enter information and data entered by user is displayed.

C++ Structure and Function

Passing structure to function in C++

A structure variable can be passed to a function in similar way as normal argument. Consider this example:

Example

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

struct Person {
    char name[50];
    int age;
    float salary;
};

void displayData(Person);   // Function declaration

int main() {
    Person p;

    cout << "Enter Full name: ";
    cin.get(p.name, 50);
    cout << "Enter age: ";
    cin >> p.age;
    cout << "Enter salary: ";
    cin >> p.salary;

    // Function call with structure variable as an argument
    displayData(p);

    return 0;
}


void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

Output

```
Enter Full name: Bill Jobs
Enter age: 55
Enter salary: 34233.4

Displaying Information.
Name: Bill Jobs
Age: 55
Salary: 34233.4
```

In this program, user is asked to enter the name, age and salary of a Person inside `main()` function.
Then, the structure variable `p` is to passed to a function using.

```
displayData(p);
```

The return type of `displayData()` is void and a single argument of type structure Person is passed.

Then the members of structure `p` is displayed from this function.

Example:

```
// example about structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
  string title;
  int year;
} mine, yours;

void printmovie (movies_t movie);

int main ()
{
  string mystr;

  mine.title = "2001 A Space Odyssey";
  mine.year = 1968;

  cout << "Enter title: ";
  getline (cin,yours.title);
  cout << "Enter year: ";
  getline (cin,mystr);
  stringstream(mystr) >> yours.year;

  cout << "My favorite movie is:\n ";
  printmovie (mine);
  cout << "And yours is:\n ";
  printmovie (yours);
  return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```

The example shows how the members of an object act just as regular variables. For example, the member `yours.year` is a valid variable of type `int`, and `mine.title` is a valid variable of type `string`.

But the objects `mine` and `yours` are also variables with a type (of type `movies_t`). For example, both have been passed to Function `printmovie` just as if they were simple variables. Therefore, one of the features of data structures is the ability to refer to both their members individually or to the entire structure as a whole. In both cases using the same identifier: the name of the structure.

Because structures are types, they can also be used as the **type of arrays** to construct tables or databases of them:

```
// array of structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
  string title;
  int year;
} films [3];

void printmovie (movies_t movie);

int main()
{
    string mystr;
    int n;
    for (n = 0; n < 3; n++)
    {
        cout << "Enter title: ";
        cin >> films[n].title;

       cout << "Enter year: ";
        cin >> films[n].year;
    }

    cout << "\nYou have entered these movies:\n";
    for (n = 0; n < 3; n++)
        printmovie(films[n]);
    return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << "  (" << movie.year << ")\n";
}
```

Output:

```
Enter title: Blade Runner
Enter year: 1982
Enter title: The Matrix
Enter year: 1999
Enter title: Taxi Driver
Enter year: 1976

You have entered these movies:
Blade Runner (1982)
The Matrix (1999)
Taxi Driver (1976)
```

Returning structure from function in C++

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

struct Person {
    char name[50];
    int age;
    float salary;
};

Person getData(Person);
void displayData(Person);
```

```
int main() {

    Person p;

    p = getData(p);
    displayData(p);

    return 0;
}

Person getData(Person p) {

    cout << "Enter Full name: ";
    cin.get(p.name, 50);

    cout << "Enter age: ";
    cin >> p.age;

    cout << "Enter salary: ";
    cin >> p.salary;

    return p;
}

void displayData(Person p) {
    cout << "\nDisplaying Information." << endl;
    cout << "Name: " << p.name << endl;
    cout <<"Age: " << p.age << endl;
    cout << "Salary: " << p.salary;
}
```

In this program, the structure variable `p` of type structure `Person` is defined under `main()` function.

The structure variable `p` is passed to `getData()` function which takes input from user which is then returned to main function.

```
p = getData(p);
```

Then the structure variable p is passed to displayData() function, which displays the information.

Note: The value of all members of a structure variable can be assigned to another structure using assignment operator = if both structure variables are of same type. You don't need to manually assign each members.