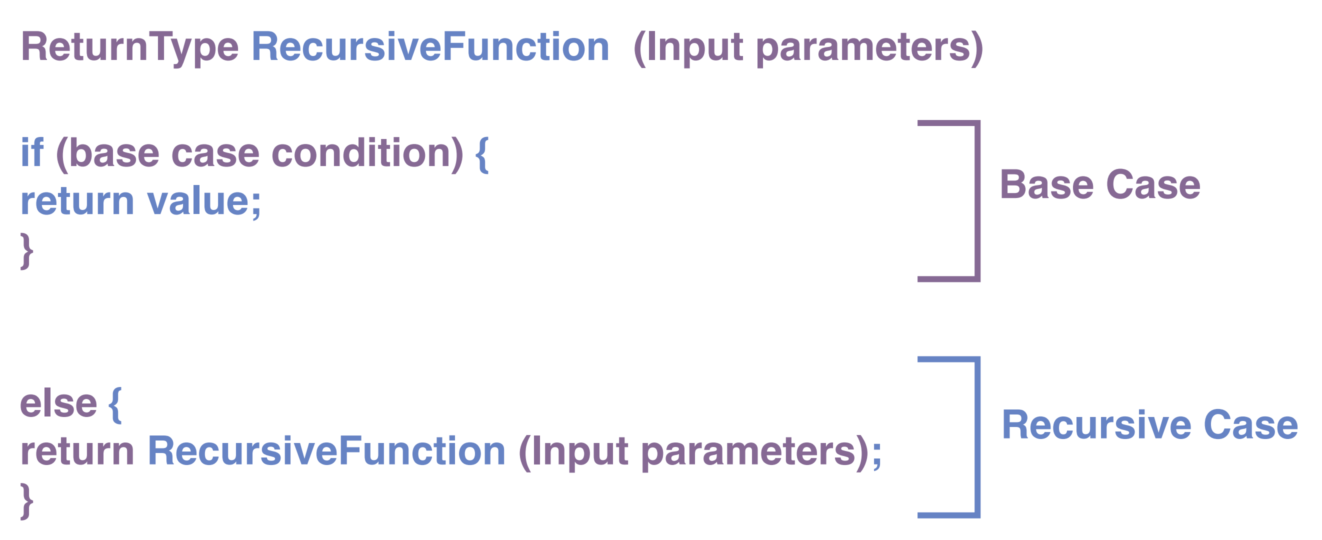
Structure of a Recursive Program

Learn about the basic structure of the recursive function.

We'll cover the following

Basic syntax

Let's go over the basic syntax of the recursive function.



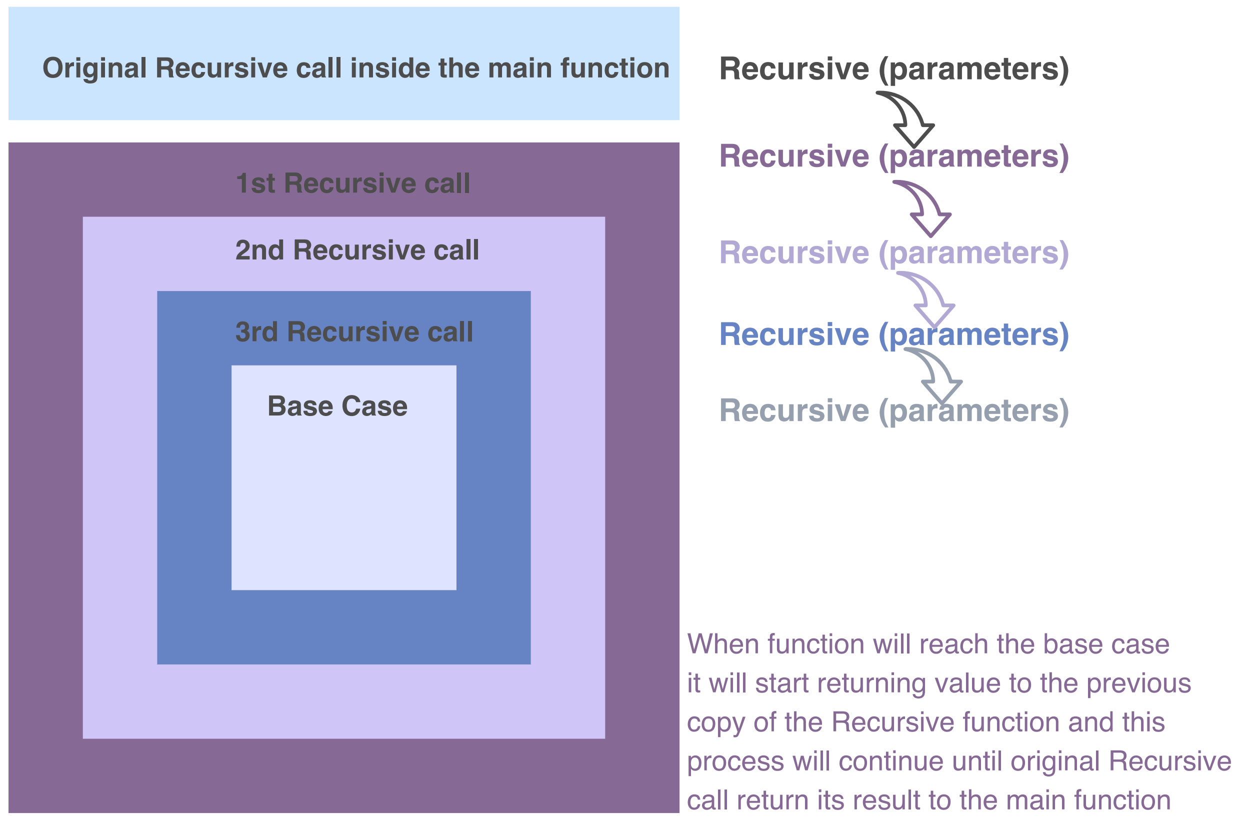
```
ReturnType RecursiveFunction  (Input parameters)

if (base case condition) {
return value;
}

else {
return RecursiveFunction (Input parameters);
}
```

The recursive function consists of the following two parts:

- **if:** represents the base case when the function should stop calling itself. It simply returns the result to the calling point.
- **else:** represents the recursive case when the function calls itself repetitively until it reaches a base case.

Illustration#

The recursive function keeps calling itself in a nested manner until it encounters a base condition. When the base condition is reached, it continues returning the value to the calling function until the original calling function is reached.


Original Recursive call inside the main function
1st Recursive call
2nd Recursive call
3rd Recursive call
Base Case
Recursive (parameters)
Recursive (parameters)
Recursive (parameters)
Recursive (parameters)
Recursive (parameters)
When function will reach the base case it will start returning value to the previous copy of the Recursive function and this process will continue until original Recursive call return its result to the main function

6 of 11

📝 The purpose of the main function is to serve as a starting point for a program. Therefore, the `main` function is not called recursively in a program.

Recursive case#

Whenever we write a recursive function, we break a complex problem into a smaller subproblem and a simpler version of itself.

Consider the example given in this lesson (https://www.educative.io/collection/page/10370001/6619096843026432/5233806832304128). Let’s translate this example into a C++ program.

📝 The program given below generates an error.

```
#include <iostream>

using namespace std;

int ticket_price(int person) {
  int price;
  person--;
  return ticket_price(person);
}

int main() {
  int price;
  price = ticket_price(10);
  cout << "Ticket price = " << price << endl;

16  }
```

Output 1.14s

```
Segmentation fault (core dumped)
```

Explanation#

In the code above, we write a function `ticket_price` that takes a value of type `int` in its input parameters (which is similar to the number of a person in a line who is asking the ticket_price) and returns a value of type `int` (which similar to returning ticket price) in the output.

Lines No. 5 to 9: If `person != 1`, it means we have not reached the first person yet. Therefore, we decrement the `person` value and again call the function `ticket_price` but with a different value of a `person` (which is similar to asking the price from the different person). Here, we are calling `ticket_price` in the body of the function `ticket_price`.
Therefore, `ticket_price` is known as a recursive function.

Why is the code given above generating an error?

The code gives an error because we are continuously calling the function `ticket_price` in its body. We need to stop the recursive calls when we meet our condition. Otherwise, the program runs out of memory and gives us an error.

Base case#

When should we stop calling the function recursively in a program?

Since we have divided the larger subproblem into a series of smaller subproblems of itself, after some time the problem will become so simple that you can solve it directly without dividing it any further. This is the **base case**.

The condition where the function stops calling itself in its body is known as the ***base case.***

The recursive function only knows how to solve the simplest case known as a **base case**. When we call the recursive function with a **base case**, it simply returns us the result. There are no recursive calls in the function.

> 📝Every recursive function must have a base case or an error is generated because of memory overflow.

Example program

Run the code given below and see the output!

```
#include <iostream>

using namespace std;

```

```
5   int ticket_price(int person) {
6     int price;
7     if (person == 1) {
8       price = 100;
9       return price;
10    }
11    else {
12      cout << "Person" << person << " is asking price" << endl;
13      person--;
14      price = ticket_price(person);
15      cout << "Person" << person << " is telling price" << endl;
16      return price;
17    }
18  }
19
20  int main() {
21    int price;
22    price = ticket_price(5);
23    cout << "Ticket price = " << price << endl;
24
25  }
```

Output 0.97s

```
Person5 is asking price
Person4 is asking price
Person3 is asking price
Person2 is asking price
Person1 is telling price
Person2 is telling price
Person3 is telling price
Person4 is telling price
Ticket price = 100
```

Lines No. 7 to 10: If `person == 1`, it means that we have just asked the `price` from the first person in line. Therefore, we return the `price` to the calling function. In the recursive function, if we can solve the problem directly, we simply return the results.

Why use recursion?#

- You might encounter a problem that is too scary. The easiest way to solve such problems is to use the **divide and conquer** rule.
- Recursion is a very powerful tool when we can define the problem in terms of itself.
- Recursion helps to write shorter code.
- We can convert loops into a recursive function.

In the next lesson, we will see how to calculate the factorial of a number using the recursive function.

Back
What is Recursion?

Next
Calculating the Factorial of a Number

Mark as Completed

Report an Issue