

Recursions & Recursive Functions in C++ | C++ Tutorials for Beginners #18

In this tutorial, we will discuss recursion and recursive functions in C++

Recursion and Recursive Function

When a function calls itself it is called recursion and the function which is calling itself is called a recursive function. The recursive function consists of a base case and recursive condition. It is very important to add a base case in recursive function otherwise recursive function will never stop executing. An example of the recursive function is shown in Code Snippet 1.

```
int factorial(int n){
    if (n<=1){
        return 1;
    }
    return n * factorial(n-1);
}
```

Code Snippet 1: Factorial Recursive Function

As shown in Code Snippet 1, we created a “factorial” function which takes one argument. In the function body, there is a base case which checks that if the value of variable “n” is smaller or equal to “1” if the condition is “true” return “1”. And there is a recursive condition that divides the bigger value to smaller values and at the end returns a factorial. These are the steps which will be performed by recursive condition:

- 4 * factorial(4-1)
- 4 * 3 * factorial(3-1)
- 4* 3 * 2 * factorial(2-1)
- 4 * 3 * 2 * 1

An example to pass the value to the recursive factorial function is shown in Code Snippet 2.

```
int main(){
    int a;
    cout<<"Enter a number"<<endl;
    cin>>a;
    cout<<"The factorial of "<<a<<" is "<<factorial(a)<<endl;
    return 0;
}
```

Code Snippet 2: Factorial Recursive Function Call

As shown in Code Snippet 2, we created an integer variable “a”, which takes input at the runtime and that value is passed to the factorial function. The output for the following program is shown in figure 1.



```
Enter a number
4
The factorial of 4 is 24
PS D:\Business\code playground\C++ course>
```

Figure 1: Factorial Recursive Function Output

As shown in figure 1, we input the value “4” and it gives us the factorial of it which is “24”. Another example of a recursive function for the Fibonacci series is shown in Code Snippet 3.

```
int fib(int n){
    if(n<2){
        return 1;
    }
    return fib(n-2) + fib(n-1);
}
```

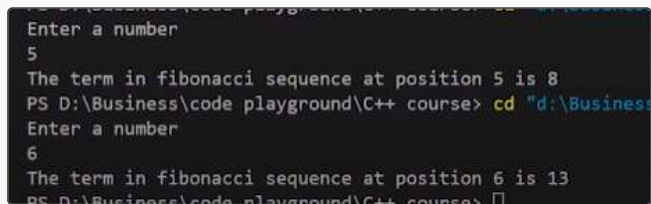
Code Snippet 3: Fibonacci Recursive Function

As shown in Code Snippet 3, we created a “fib” function which takes one argument. In the function body, there is a base case which checks that if the value of variable “n” is smaller than “2”, if the condition is “true” return “1”. And there is a recursive condition that divides the bigger value to smaller values and at the end returns a Fibonacci number. An example to pass the value to the Fibonacci function is shown in Code Snippet 4.

```
int main(){
    int a;
    cout<<"Enter a number"<<endl;
    cin>>a;
    cout<<"The term in fibonacci sequence at position "<<a<<" is "<<fib(a)<<endl;
    return 0;
}
```

Code Snippet 4: Fibonacci Recursive Function Call

As shown in Code Snippet 4, we created an integer variable “a”, which takes input at the runtime and that value is passed to the Fibonacci function. The output for the following program is shown in figure 2.



```
Enter a number
5
The term in fibonacci sequence at position 5 is 8
PS D:\Business\code playground\C++ course> cd "d:\Business
Enter a number
6
The term in fibonacci sequence at position 6 is 13
PS D:\Business\code playground\C++ course> █
```

Figure 2: Fibonacci Recursive Function Output

As shown in figure 2, 1st we input the value “5” and it gives us the Fibonacci number at that place which is “8”. 2nd we input the value “6” and it gives us the Fibonacci number at that place which is “13”.

One thing to note here is that recursive functions are not always the best option. They perform well in some problems but not in every problem.

Code as described/written in the video

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

int fib(int n){
    if(n<2){
        return 1;
    }
    return fib(n-2) + fib(n-1);
}

// fib(5)
// fib(4) + fib(3)
// fib(2) + fib(3) + fib(2) + fib(3)

int factorial(int n){
    if (n<=1){
        return 1;
    }
    return n * factorial(n-1);
}

// Step by step calculation of factorial(4)
// factorial(4) = 4 * factorial(3);
// factorial(4) = 4 * 3 * factorial(2);
// factorial(4) = 4 * 3 * 2 * factorial(1);
```

