Chapter 10: Intro to Recursion

CSE 2010 Week 13

Code and Figures from : “C++ programming: an object-oriented approach” - Forouzan, Behrouz A, Gilberg, Richard, 2020.

Introduction

In previous chapters, we have learned to write functions to perform specific tasks. When the task requires something to be done repeatedly, we have used iteration (loops).

Now we are going to learn how to solve repetitive problems **recursively.**

Repetition vs Recursion

- In a simple iteration, we use a counter to repeat a task (chunk of code) n times.
- In recursion, the function does the task only once, but then it **calls itself** n-1 times to achieve the same goal.

So what exactly is recursion..

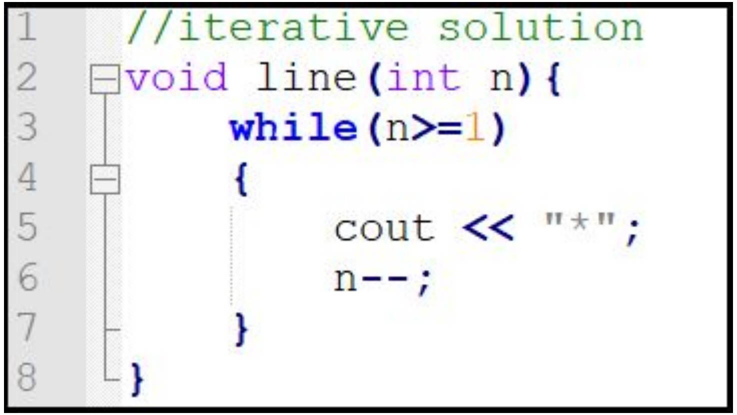
- Recursion is a powerful programming technique that allows us to break up complex computational problems into smaller, more simple ones.
- Recursion even allows us to implement a solution in a way that mirrors our natural (human) way of thinking about a problem.
- We can accomplish recursion by implementing functions that call themselves to complete a task.

There are 2 key requirements to make sure that our recursive functions are successful:

1. Every recursive call must simplify the computation in some way (use smaller values with each call).
2. There must be special cases to handle the simplest computations directly.
   - Each recursive function has a **general case** and **base case**.
   - A base case is the case that will terminate the recursion, while a general case is related to calls that do something and continue the recursion.

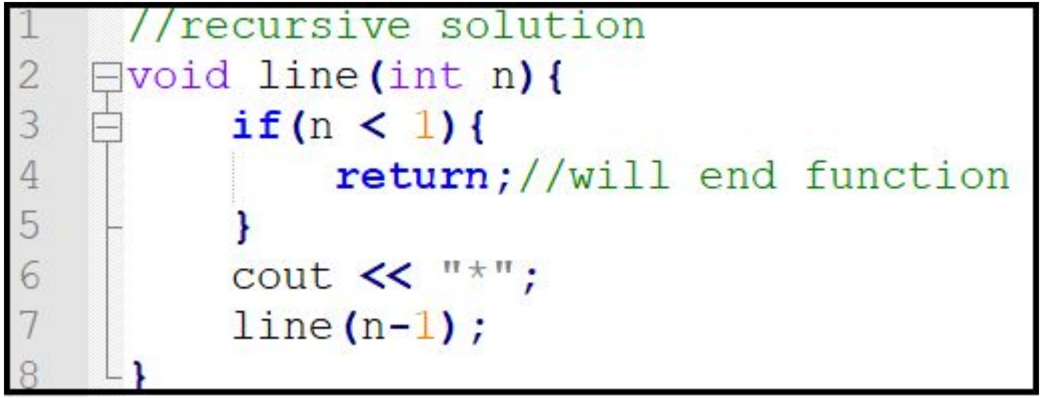
Our first (simple) recursive problem - void recursion

Assume that we need to print *n* asterisks on a line and the value of *n* is known. Let’s see how we can solve this iteratively and recursively.



```
//iterative solution
void line(int n){
    while(n>=1)
    {
        cout << "*";
        n--;
    }
}
```

The function is called a single time, and the loop is controlled by reducing the value of n with each iteration with n--.

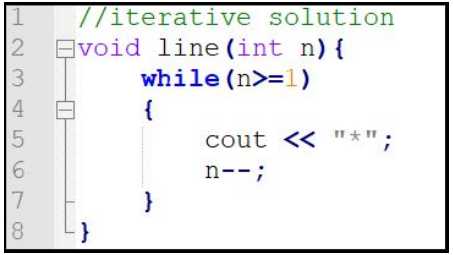


```
//recursive solution
void line(int n){
    if(n < 1){
        return;//will end function
    }
    cout << "*";
    line(n-1);
}
```

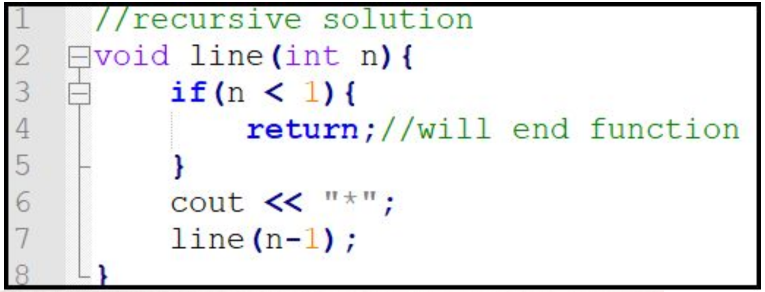
The function is called n-1 times. In each call, except the last where n=0, the function prints a single *.

What are our general and base cases?

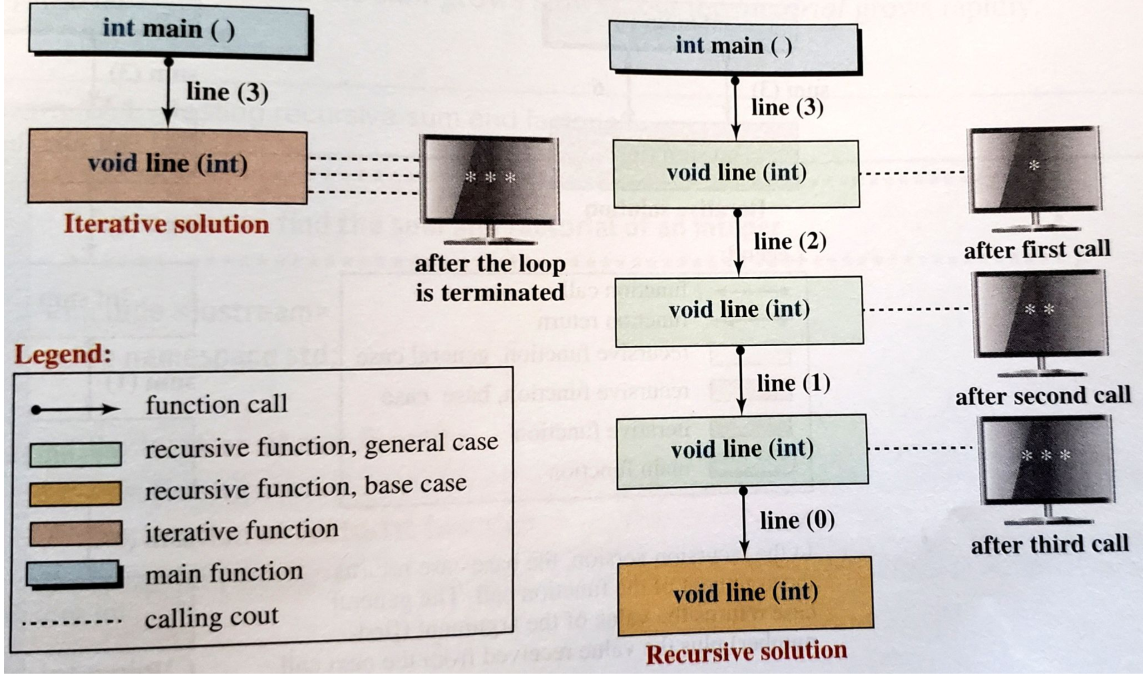
Closer look at what's going on...



```
//iterative solution
void line(int n){
    while(n>=1)
    {
        cout << "*";
        n--;
    }
}
```

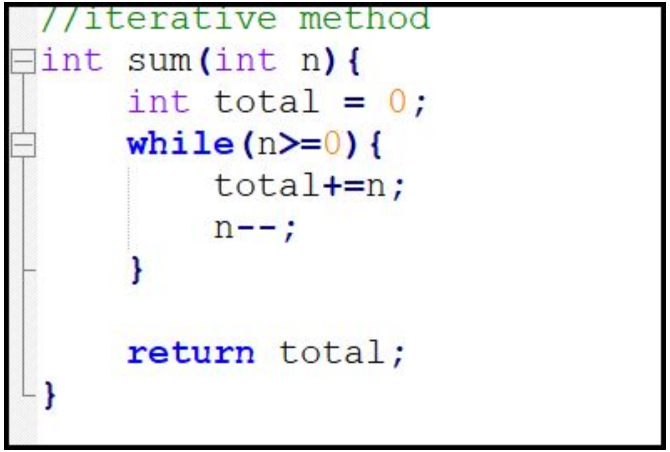


```
//recursive solution
void line(int n){
    if(n < 1){
        return;//will end function
    }
    cout << "*";
    line(n-1);
}
```

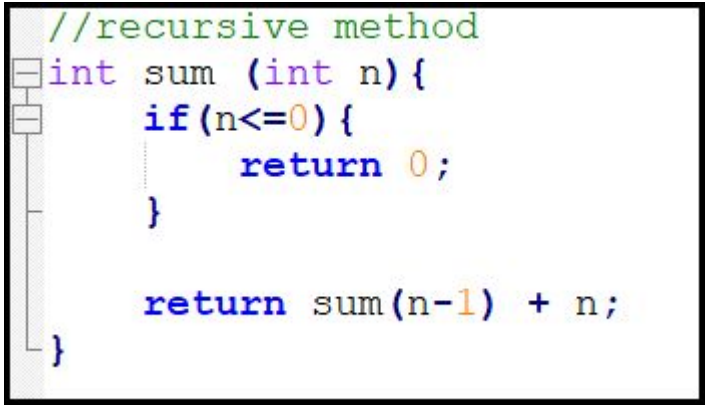


Another example - value returning recursion

Assume that we need to find the sum of all numbers from 0 to *n*, then return the sum to the calling function.



```
//iterative method
int sum(int n){
        int total = 0;
        while(n>=0){
                total+=n;
                n--;
        }

        return total;
}
```



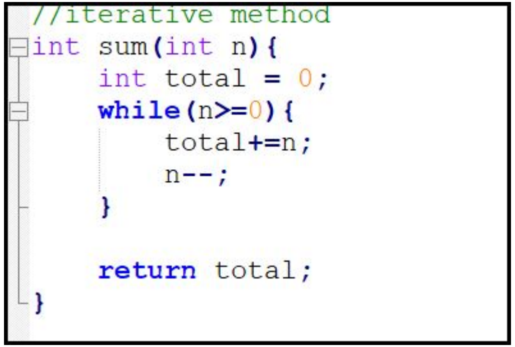
```
//recursive method
int sum (int n){
        if(n<=0){
                return 0;
        }

        return sum(n-1) + n;
}
```

In both programs, we are summing the numbers backward, from *n* to 0.

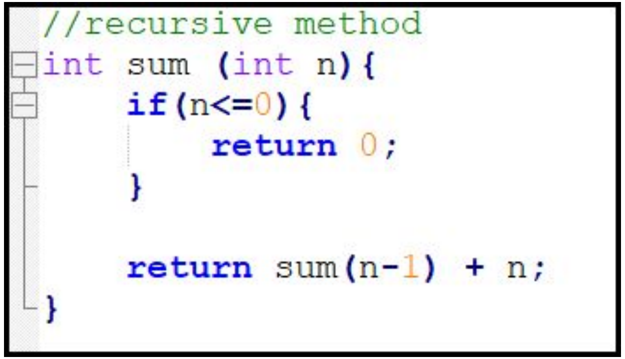
total = n + n-1 + n-2 + ...+ 1 + 0

Base case: n<=0
General case: sum(n-1) + n

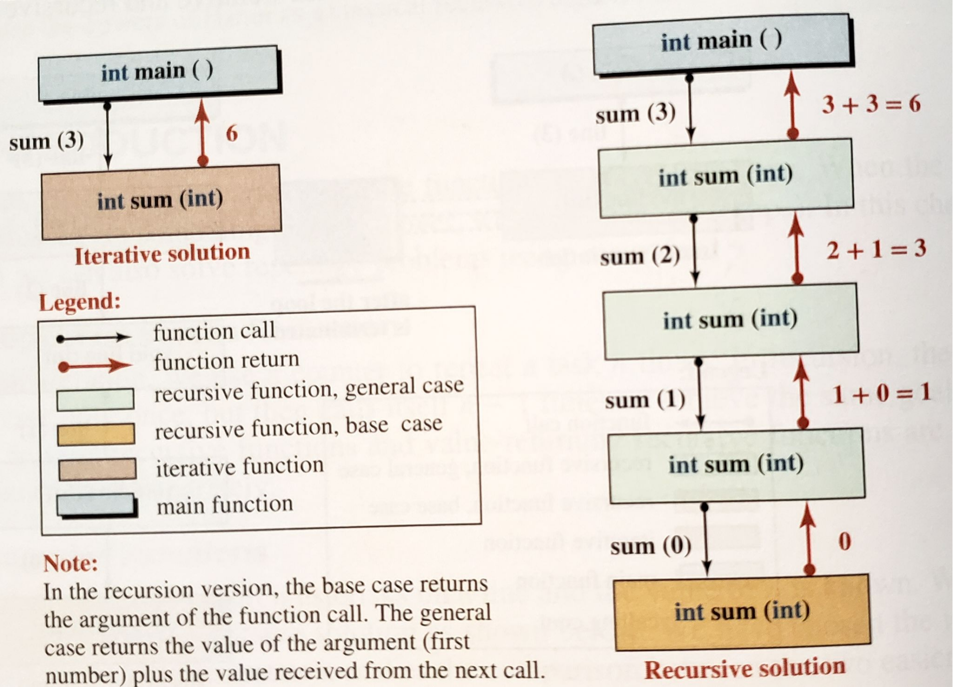
Let's examine the flow of the functions...



```
//iterative method
int sum(int n){
    int total = 0;
    while(n>=0){
        total+=n;
        n--;
    }

    return total;
}
```



```
//recursive method
int sum (int n){
    if(n<=0){
        return 0;
    }

    return sum(n-1) + n;
}
```



Iterative solution

Recursive solution

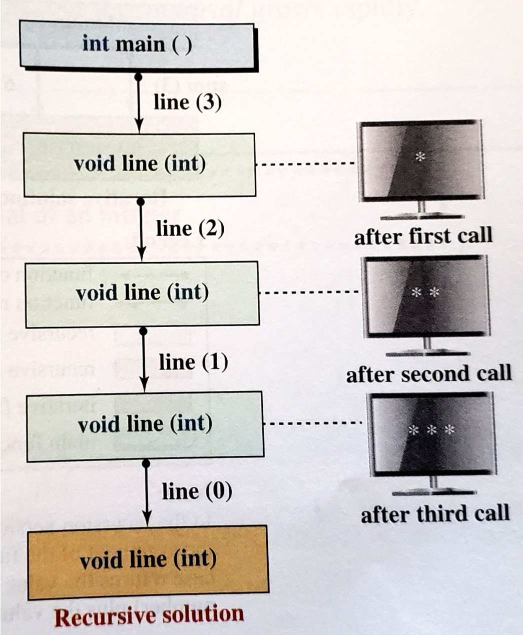
Legend:

- → function call
- → function return
- 🟩 recursive function, general case
- 🟨 recursive function, base case
- 🟫 iterative function
- 🟦 main function

Note:
In the recursion version, the base case returns the argument of the function call. The general case returns the value of the argument (first number) plus the value received from the next call.

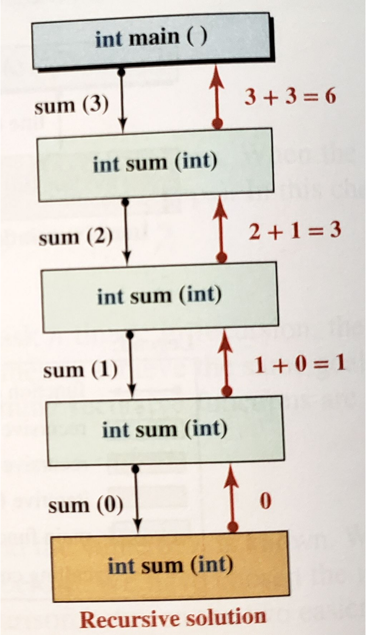
Void vs. Value Returning Recursive Functions

For void recursive functions, general cases are continuously called until a base case is reached. The general case does not need to hold any information.



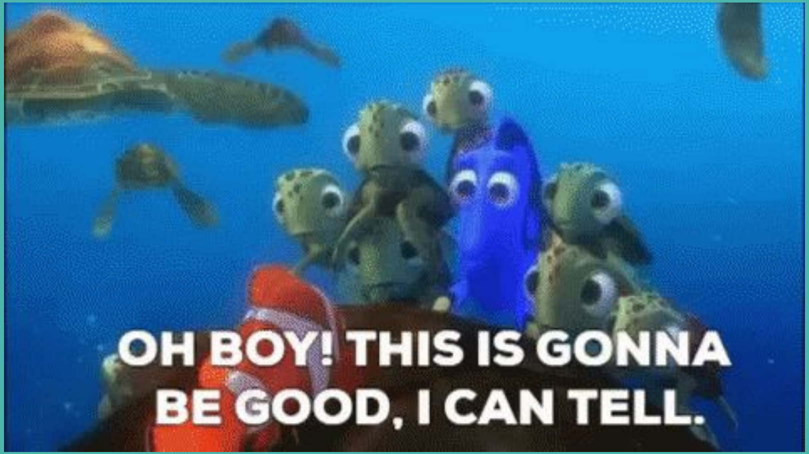
Recursive solution

For value-returning recursive functions, the general cases are called until the base case is reached. Each general case must hold some information until the call to the next step is returned.



Recursive solution

Recursion is one of those concepts that gets easier to understand as you see more examples, so we're going to look over several recursive algorithms.



For now let’s look at Fibonacci Numbers

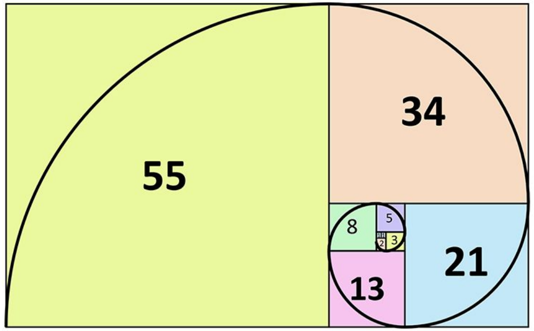
The Fibonacci sequence is a sequence of numbers in which each number is the sum of the previous two numbers.

The first 10 terms of a sequence are:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55

The Fibonacci numbers problem has two base cases and a general case

Base Case: F(0) = 0 & F(1) = 1

General Case: F(n) = F(n-1) + F(n-2)



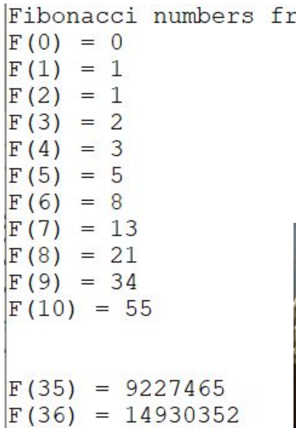
Recursive Implementation to find the nth Fibonacci

The Fibonacci numbers problem has two base cases and a general case

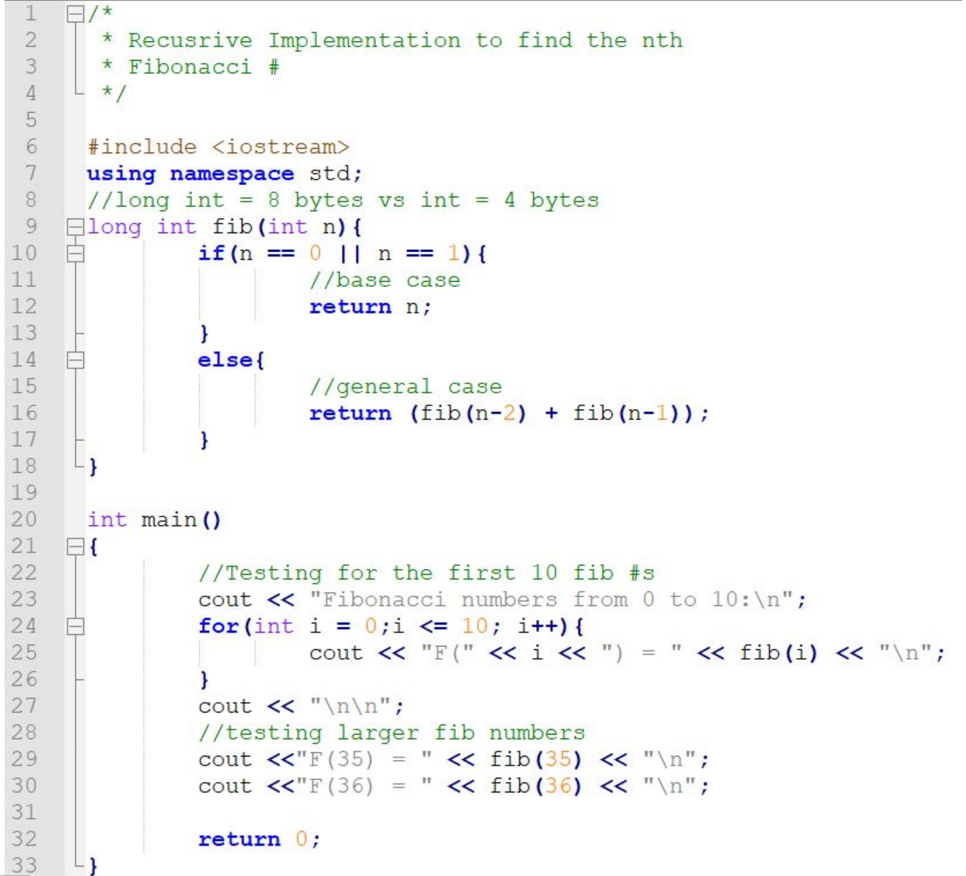
Base Case: F(0) = 0 & F(1) = 1

General Case: F(n) = F(n-1) + F(n-2)

Output

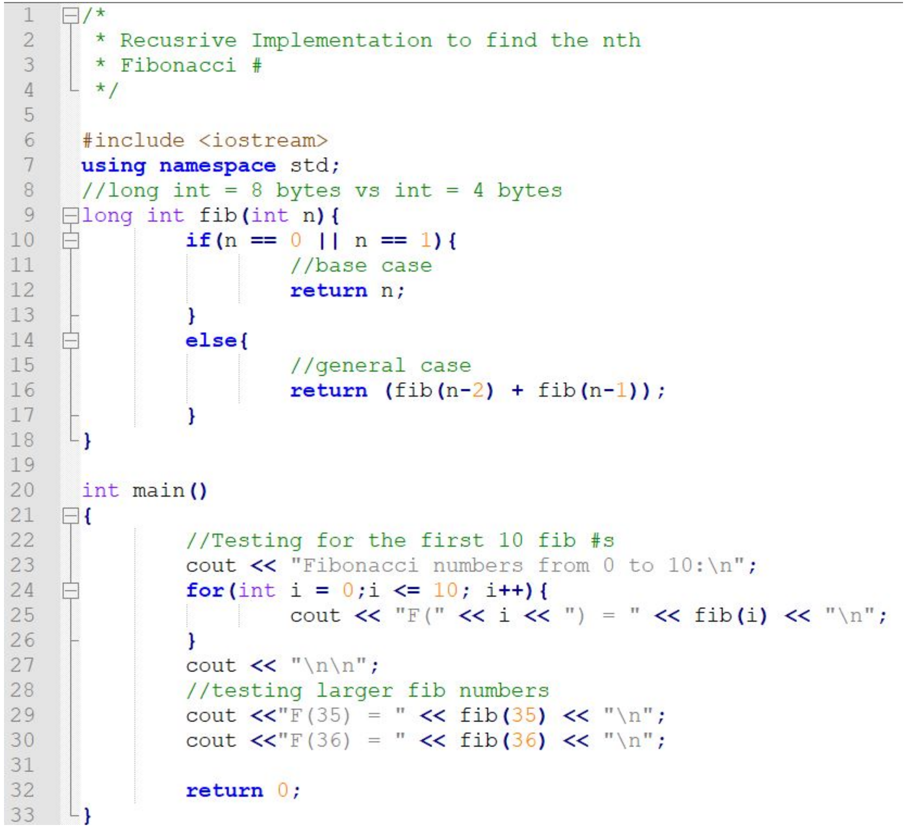


```
Fibonacci numbers from 0 to 10:
F(0) = 0
F(1) = 1
F(2) = 1
F(3) = 2
F(4) = 3
F(5) = 5
F(6) = 8
F(7) = 13
F(8) = 21
F(9) = 34
F(10) = 55


F(35) = 9227465
F(36) = 14930352
```

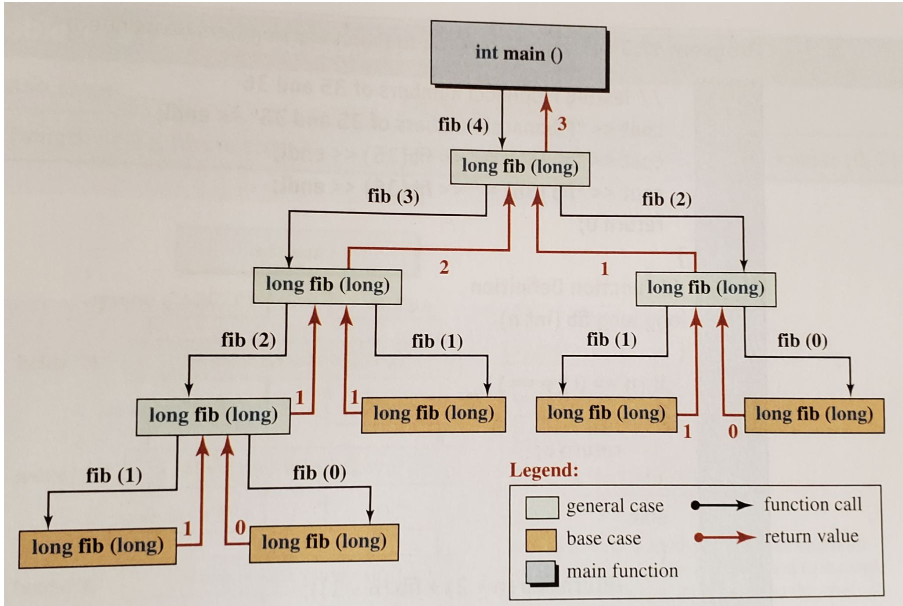


```
/*
 * Recusrive Implementation to find the nth
 * Fibonacci #
 */

#include <iostream>
using namespace std;
//long int = 8 bytes vs int = 4 bytes
long int fib(int n){
        if(n == 0 || n == 1){
                //base case
                return n;
        }
        else{
                //general case
                return (fib(n-2) + fib(n-1));
        }
}

int main()
{
        //Testing for the first 10 fib #s
        cout << "Fibonacci numbers from 0 to 10:\n";
        for(int i = 0;i <= 10; i++){
                cout << "F(" << i << ") = " << fib(i) << "\n";
        }
        cout << "\n\n";
        //testing larger fib numbers
        cout <<"F(35) = " << fib(35) << "\n";
        cout <<"F(36) = " << fib(36) << "\n";

        return 0;
}
```

Recursive Trace



```
/*
 * Recusrive Implementation to find the nth
 * Fibonacci #
 */

#include <iostream>
using namespace std;
//long int = 8 bytes vs int = 4 bytes
long int fib(int n){
        if(n == 0 || n == 1){
                //base case
                return n;
        }
        else{
                //general case
                return (fib(n-2) + fib(n-1));
        }
}

int main()
{
        //Testing for the first 10 fib #s
        cout << "Fibonacci numbers from 0 to 10:\n";
        for(int i = 0;i <= 10; i++){
                cout << "F(" << i << ") = " << fib(i) << "\n";
        }
        cout << "\n\n";
        //testing larger fib numbers
        cout <<"F(35) = " << fib(35) << "\n";
        cout <<"F(36) = " << fib(36) << "\n";

        return 0;
}
```


int main ()
fib (4)
3
long fib (long)
fib (3)
fib (2)
2
1
long fib (long)
long fib (long)
fib (2)
fib (1)
fib (1)
fib (0)
long fib (long)
1
1
long fib (long)
long fib (long)
1
0
long fib (long)
fib (1)
fib (0)
long fib (long)
1
0
long fib (long)
Legend:
general case
base case
main function
function call
return value