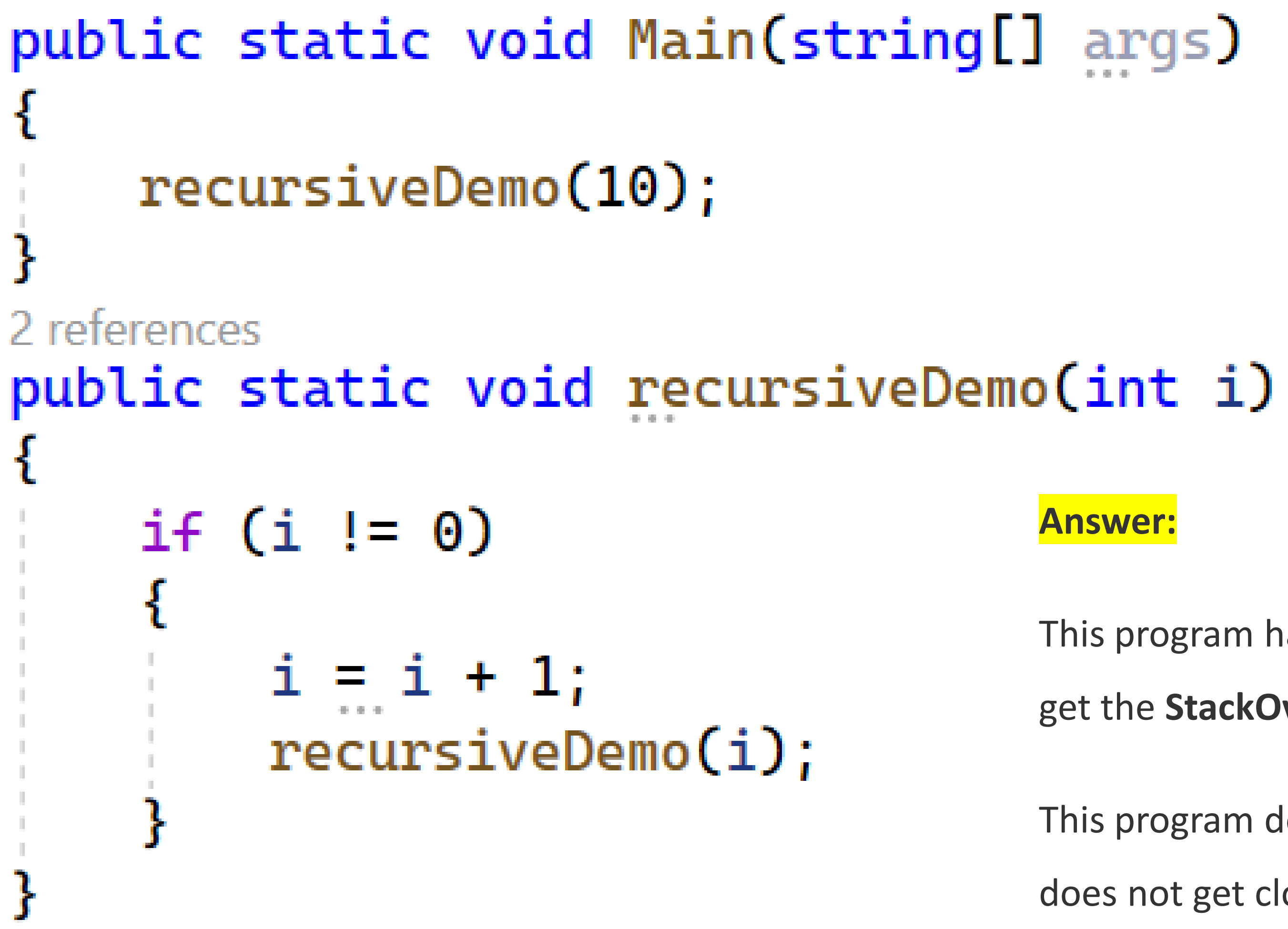
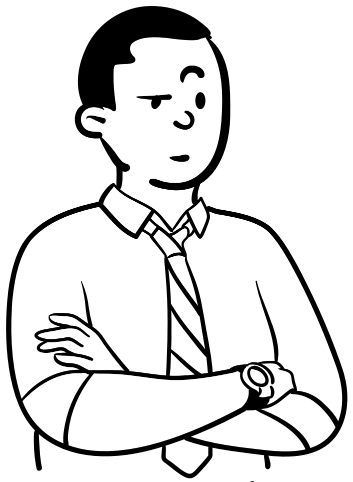
Example:



```
public static void Main(string[] args)
{
    recursiveDemo(10);
}
2 references
public static void recursiveDemo(int i)
{
    if (i != 0)
    {
        i = i + 1;
        recursiveDemo(i);
    }
}
```

Does it hold recursion characteristics????



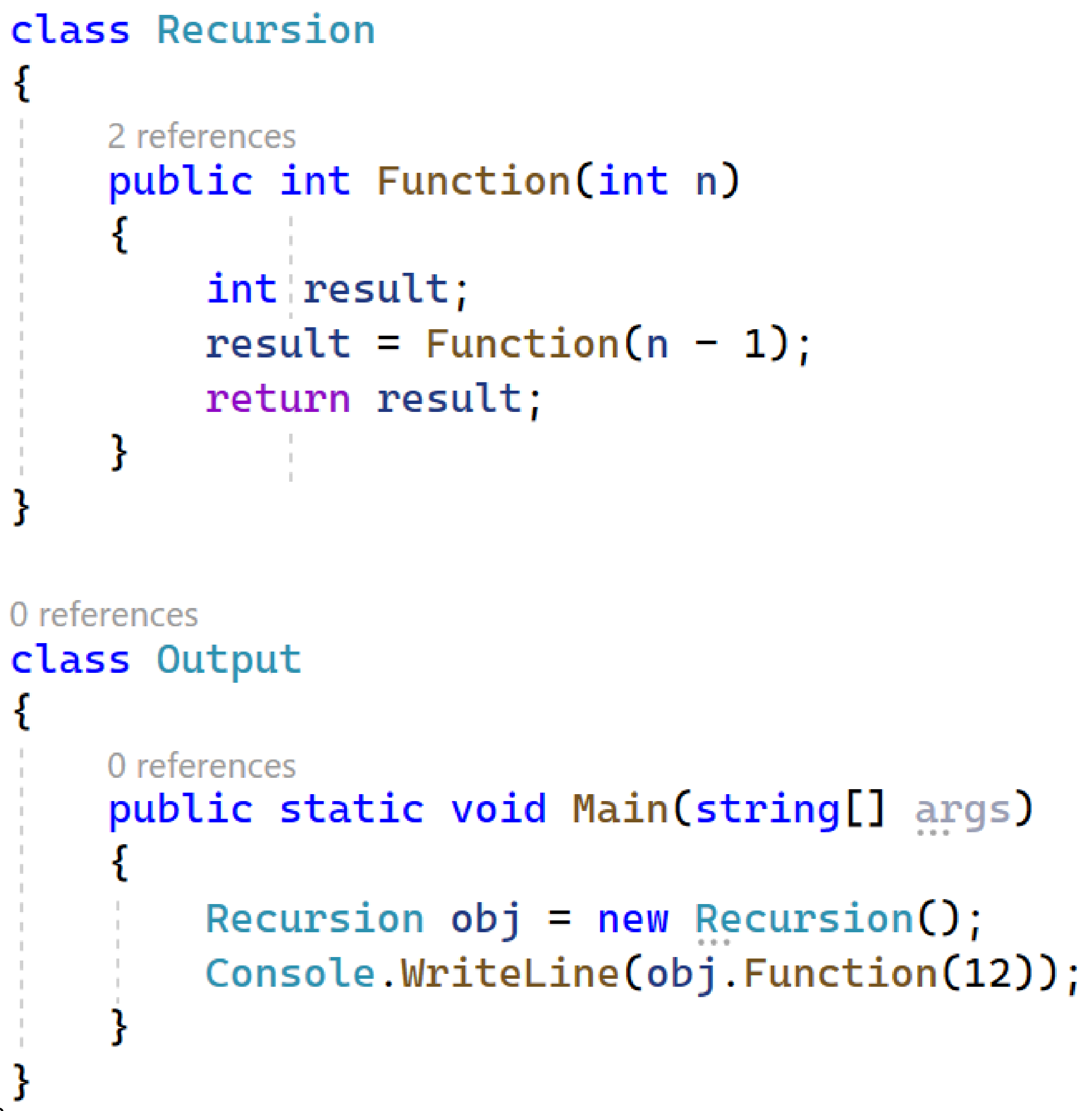
Answer:

This program has a problem. We observe that on running the program, we get the **StackOverflowException**.

This program does have some sort of base case (i==0) but the recursive call does not get closer to the base case. It goes away from base case leading to StackOverflowException due to the wrong logic of incrementing i, instead of decrementing it.

SDU

What will be the output of the following C# program?



```
class Recursion
{
    2 references
    public int Function(int n)
    {
        int result;
        result = Function(n - 1);
        return result;
    }
}

0 references
class Output
{
    0 references
    public static void Main(string[] args)
    {
        Recursion obj = new Recursion();
        Console.WriteLine(obj.Function(12));
    }
}
```

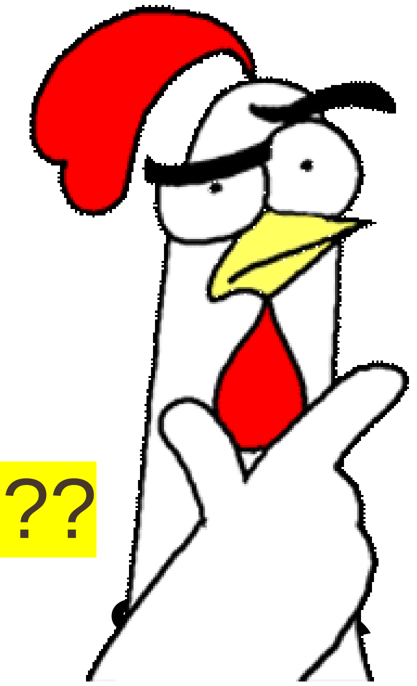
Answer:
Since the base case of the recursive function Function() is not defined hence infinite loop occurs and results in StackOverflowException.

Printing a message n times

nPrintln("Welcome", 5);

```
public static void nPrintln(String message, int times) {
  if (times >= 1) {
    System.out.println(message);
    nPrintln(message, times - 1);
  }
}
```

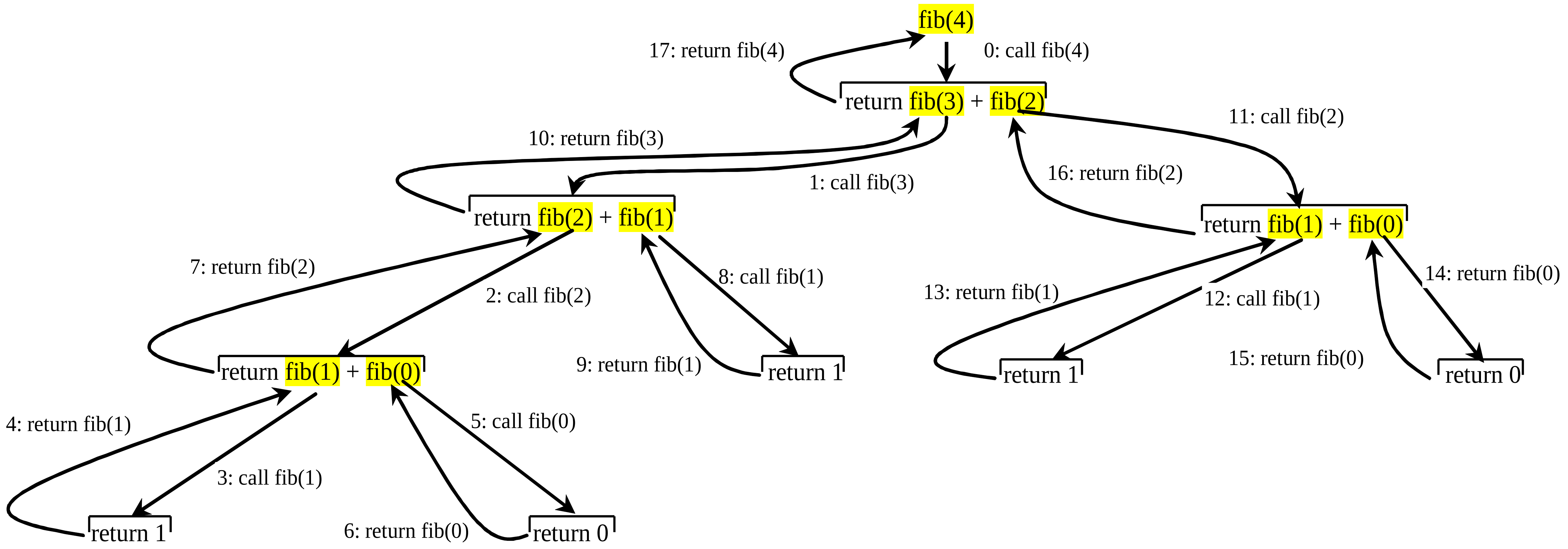
Can you identify the base case????



Answer:
The base case is times == 0

Fibonacci Numbers

Do you see any problem?



Answer: You could easily see that here are many duplicated recursive calls. For instance, **Fib(2)** is called twice, **Fib(1)** three times, and **Fib(0)** twice.

This implies that although Its true that we can apply recursion to many problem, if doesn't necessarily mean that they are the most efficient at solving problems

Recursive Helper Methods

- The IsPalindrome method is not efficient, because it creates a new string for every recursive call.
- To avoid creating new strings, use a helper method
- Recurvise helper methods usually takes more parameters than their primary methods

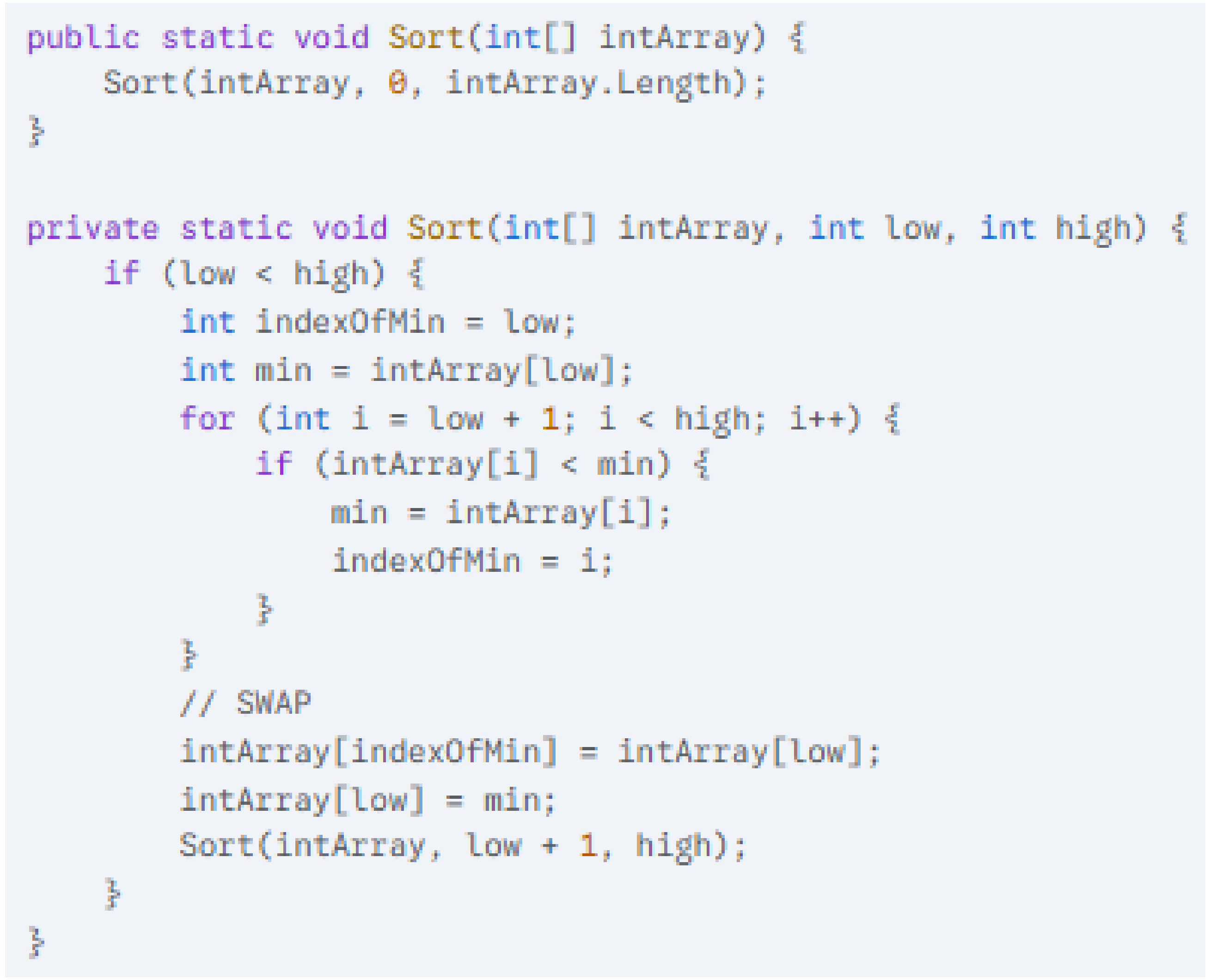
```
public static bool IsPalindrome(string s)
{
 return IsPalindrome(s, 0, s.Length - 1);
}

public static bool IsPalindrome(string s, int low, int high)
{
    if (high <= low) // Base case
        return true;
    else if (s[low] != s[high]) // Base case
        return false;
    else
        return IsPalindrome(s, low + 1, high - 1);
}
```

Recursive helper method

What is special about **IsPalindrome()** method?

Answer: Two overloaded **isPalindrome** methods

Selection Sort example



```
public static void Sort(int[] intArray) {
    Sort(intArray, 0, intArray.Length);
}

private static void Sort(int[] intArray, int low, int high) {
    if (low < high) {
        int indexOfMin = low;
        int min = intArray[low];
        for (int i = low + 1; i < high; i++) {
            if (intArray[i] < min) {
                min = intArray[i];
                indexOfMin = i;
            }
        }
        // SWAP
        intArray[indexOfMin] = intArray[low];
        intArray[low] = min;
        Sort(intArray, low + 1, high);
    }
}
```

Can you spot a primary method and a recursive helper method?

Answer:
Primary method= Sort(int[] intArray)
Recursive helper method= Sort(int[] intArray, int low, int high)