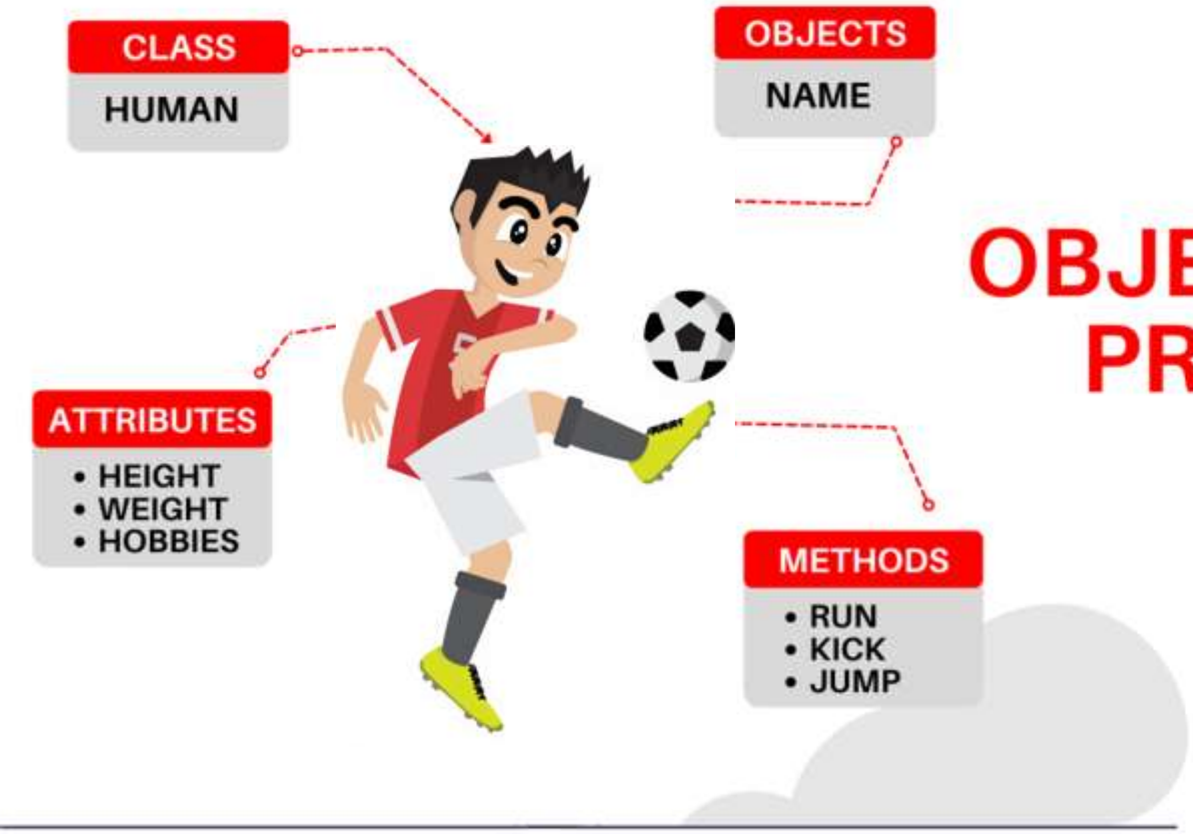
National University of Computer and Emerging Sciences

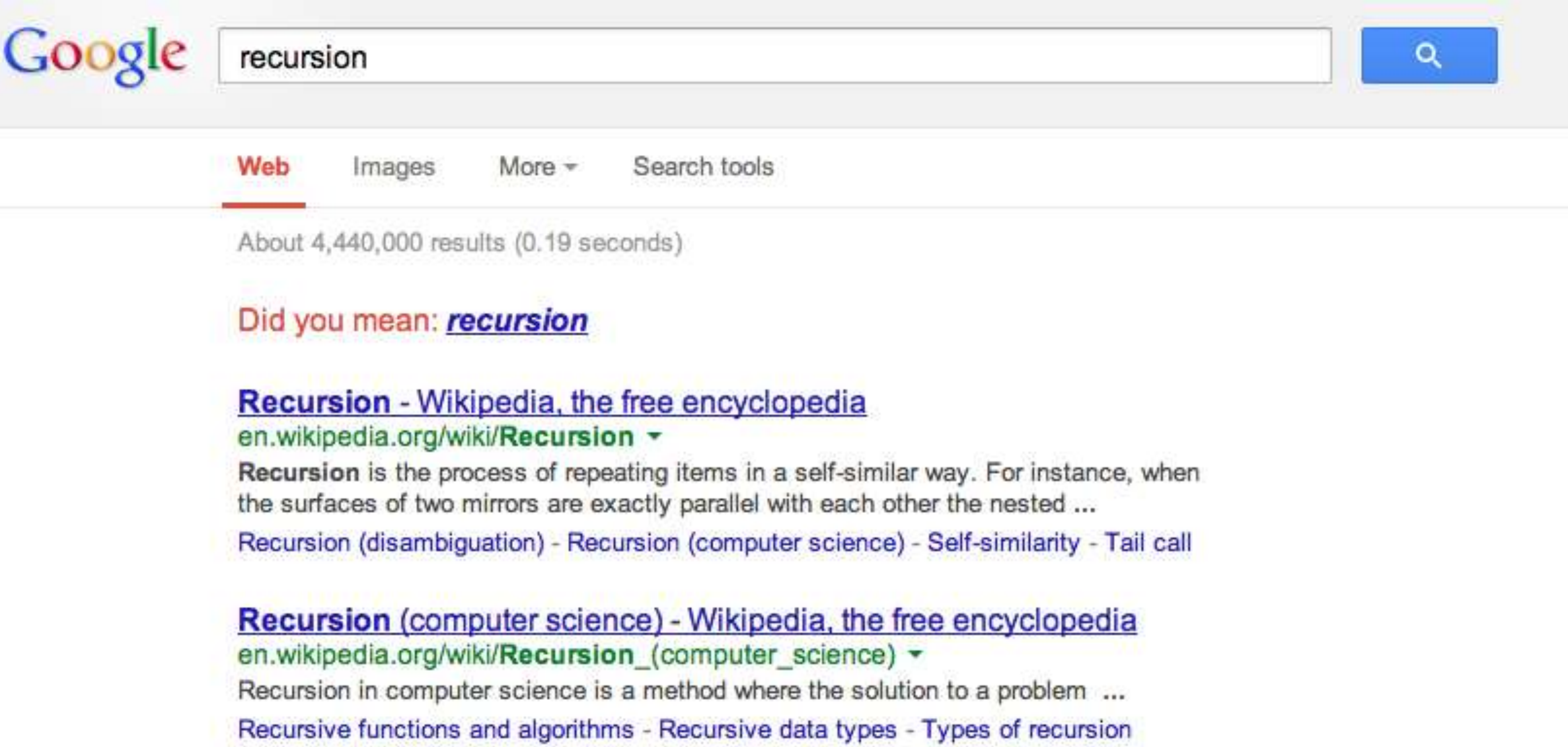


OBJECT-ORIENTED PRORAMMING

Summer 2023

Pir Sami Ullah Shah

Lecture # 4 Recursion



About 4,440,000 results (0.19 seconds)

Did you mean: ***recursion***

Recursion - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Recursion**

Recursion is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other the nested ...

Recursion (disambiguation) - Recursion (computer science) - Self-similarity - Tail call

Recursion (computer science) - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Recursion**_(computer_science)

Recursion in computer science is a method where the solution to a problem ...

Recursive functions and algorithms - Recursive data types - Types of recursion

Recursive Function

- A function that calls itself is called recursive function

```
void Message()
{
        cout << "This is a recursive function.\n";
        Message();
}
```

- What is the problem with the above function?

Recursion

- The function message() is like an infinite loop because there is no code to stop it from repeating

Recursive Function – Number of Repetitions

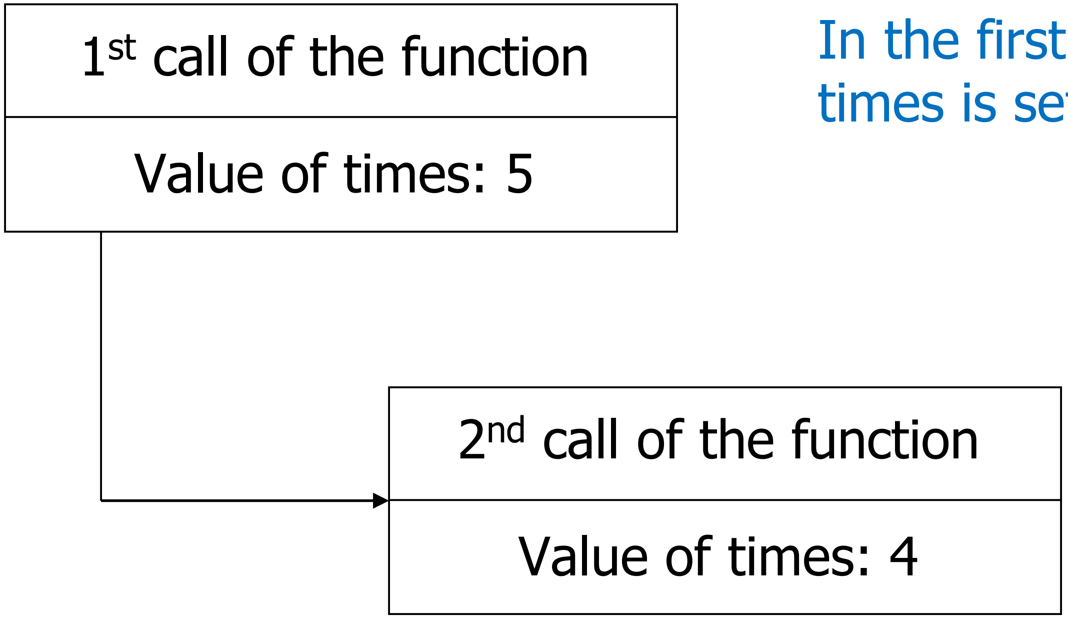
- Recursive function must have some algorithm (i.e., logic) to control the number of times it repeats

```
void Message(int times)
{
   if (times > 0) //base case
   {
      cout << "This is a recursive function.\n";
      Message(times - 1);
   }
   return;
}
```

- Modification to `Message` function
  - Receive an `int` argument to **control the number to times to call itself**
  - For each recursive call, the parameter controlling the recursion should **move closer to the base case (converge)**

Recursive Function – Execution (1)

- Each time the function is called, a new instance of the **`times`** parameter is created
  - Suppose program invokes the function as **`Message(5)`**



1st call of the function

Value of times: 5

In the first call to function, times is set to 5.

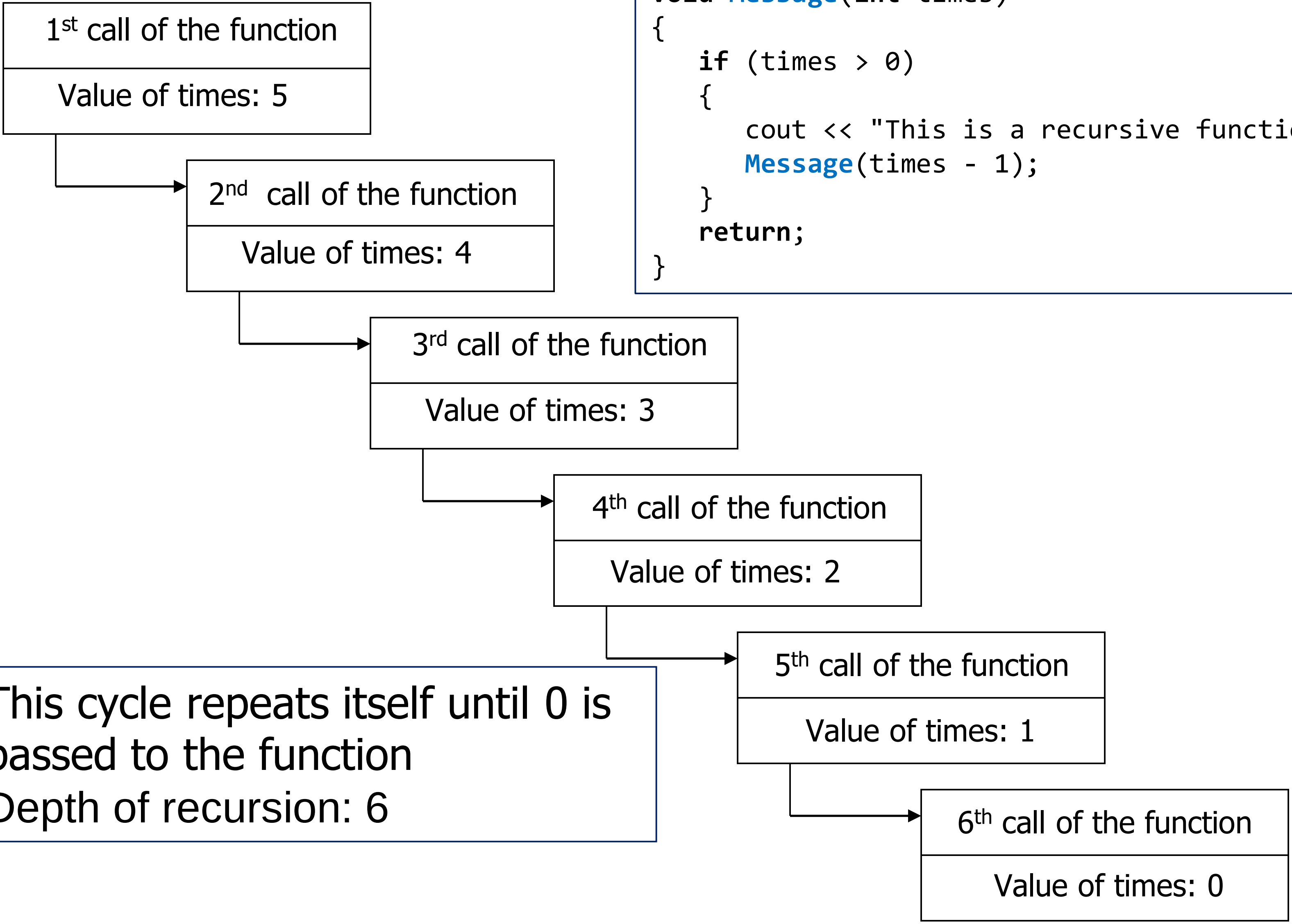
2nd call of the function

Value of times: 4

When the function calls itself, a new instance of times is created with the value 4.

Recursive Function – Execution (2)

```
void Message(int times)
{
    if (times > 0)
    {
        cout << "This is a recursive function.\n";
        Message(times - 1);
    }
    return;
}
```



1st call of the function

Value of times: 5

2nd call of the function

Value of times: 4

3rd call of the function

Value of times: 3

4th call of the function

Value of times: 2

5th call of the function

Value of times: 1

6th call of the function

Value of times: 0

- This cycle repeats itself until 0 is passed to the function
- Depth of recursion: 6

Program Output

```
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
This is a recursive function.
```

What Happens When Called?

- Each time a recursive function is called, a new copy of the function runs, with new instances of parameters and local variables being created

- As each copy finishes executing, it returns to the copy of the function that called it

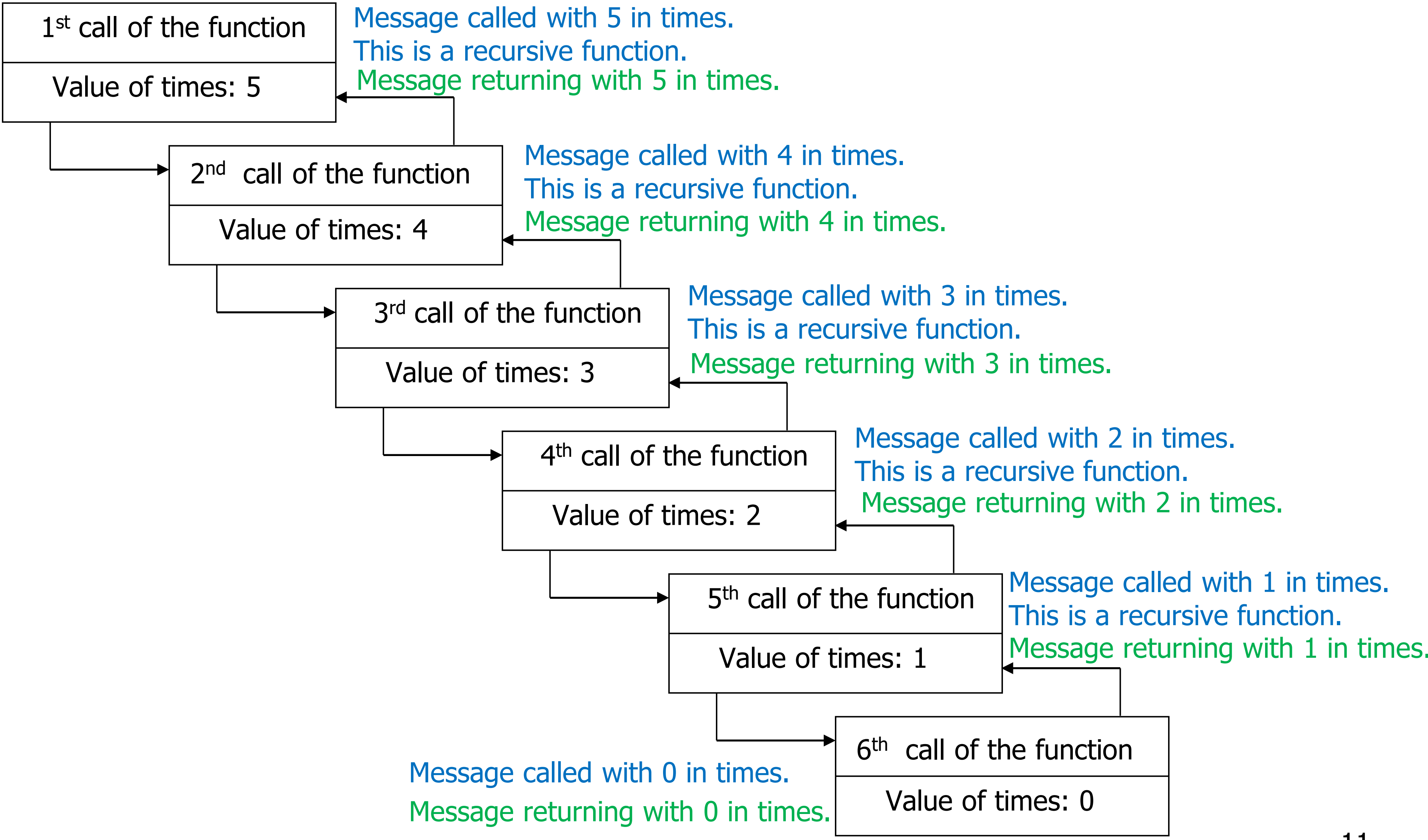
- When the first copy finishes executing, it returns to the part of the program that made the initial call to the function

Recursive Function – Modification

- Statements after the recursive invocation of the function

```
void Message(int times)
{
   cout << "Message called with " << times <<" in times.\n";
   if (times > 0) {
      cout << "This is a recursive function.\n";
      Message(times - 1);
   }
   cout << "Message returning with " << times;
   cout << " in times.\n";
}
```

Recursive Function – Execution (3)



1st call of the function

Value of times: 5

Message called with 5 in times.
This is a recursive function.
Message returning with 5 in times.

2nd call of the function

Value of times: 4

Message called with 4 in times.
This is a recursive function.
Message returning with 4 in times.

3rd call of the function

Value of times: 3

Message called with 3 in times.
This is a recursive function.
Message returning with 3 in times.

4th call of the function

Value of times: 2

Message called with 2 in times.
This is a recursive function.
Message returning with 2 in times.

5th call of the function

Value of times: 1

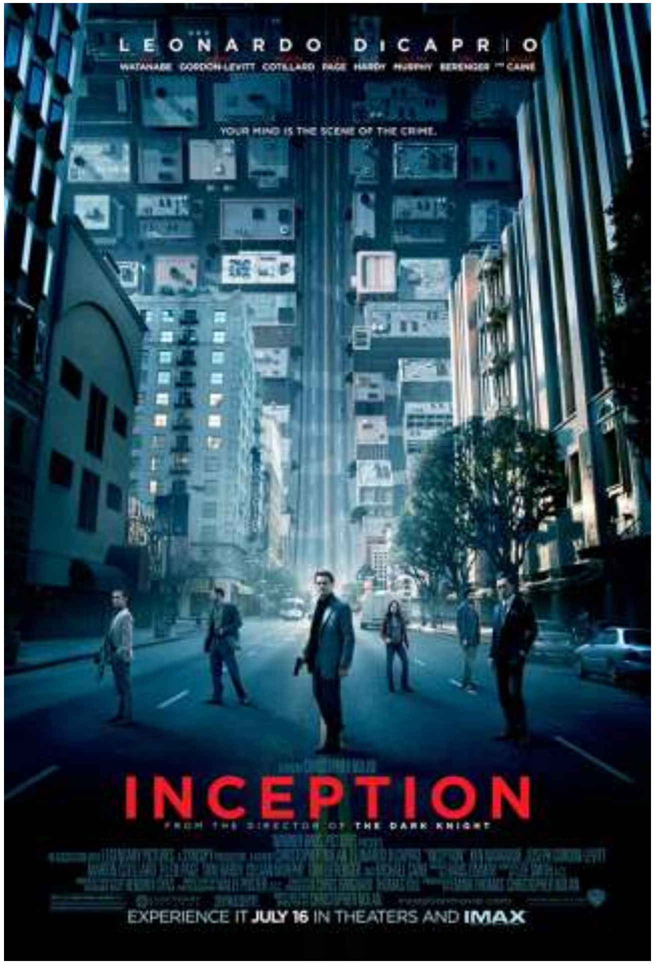
Message called with 1 in times.
This is a recursive function.
Message returning with 1 in times.

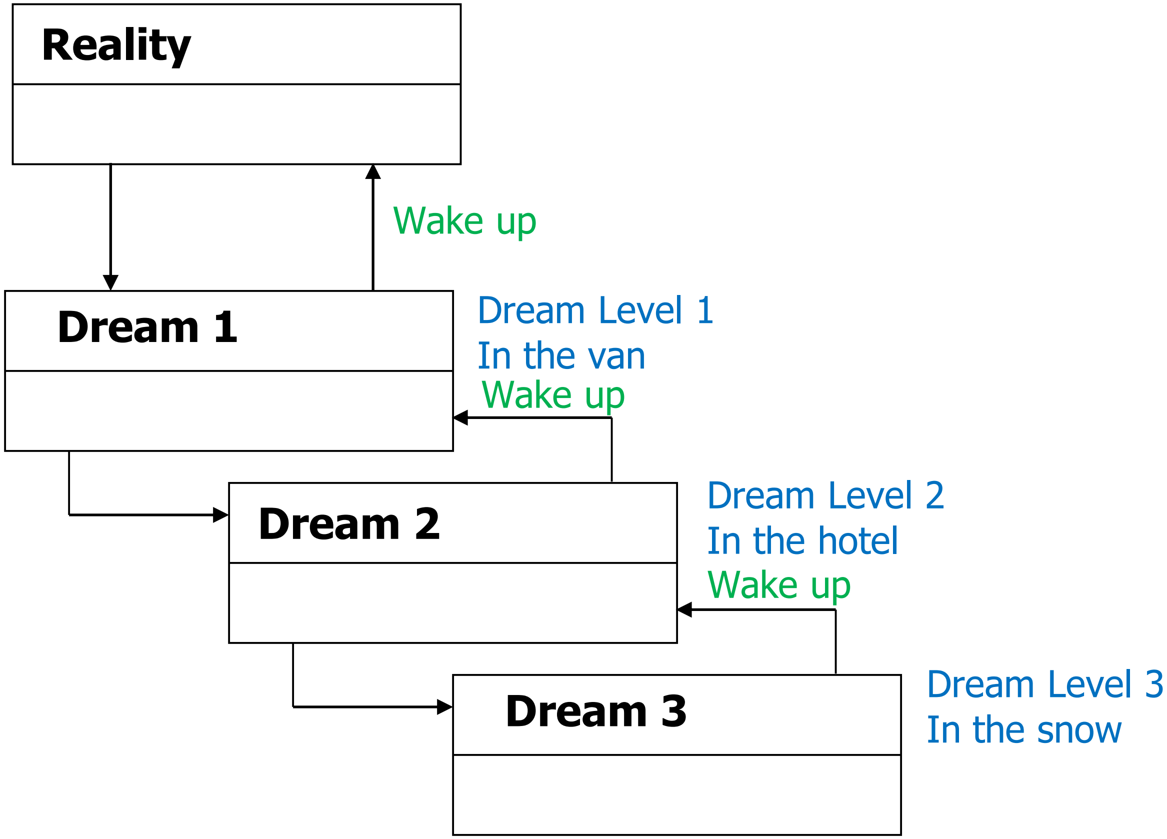
6th call of the function

Value of times: 0

Message called with 0 in times.
Message returning with 0 in times.

Recursion


LEONARDO DICAPRIO
WATANABE GORDON-LEVITT COTILLARD PAGE HARDY MURPHY BERENGER AND CAINE
YOUR MIND IS THE SCENE OF THE CRIME.
INCEPTION
FROM THE DIRECTOR OF THE DARK KNIGHT
EXPERIENCE IT JULY 16 IN THEATERS AND IMAX



Reality

Wake up

Dream 1

Dream Level 1
In the van

Wake up

Dream 2

Dream Level 2
In the hotel

Wake up

Dream 3

Dream Level 3
In the snow

Recursion

Recursion

- Solving a problem by reducing it to a smaller version of itself

- A properly written recursive function must
  - Handle the base cases, and
  - Recursive cases (convergence to the base case)

- Failure to properly handle the base case or converge to the base case may result in infinite recursion

- Very Important: NO LOOPS!!! (for, while, do-while)

Recursion

- To solve problem recursively

```
1. Define the base case(s)
2. Define the recursive case(s)
   a) Divide the problem into smaller sub-problems
   b) Solve the sub-problems
   c) Combine results to get answer
```

Sub-problems solved as a recursive call to the same function

- Sub-problem must be smaller than the original problem
  - Otherwise recursion never terminates

Print Numbers in Descending Order

```
void printDes(int n) {

    if ( n <= 0 ) //Base condition
           return;

    cout << n << " "; //Prints number n

    printDes(n-1); //Recursive call
}
```

print(10) produces ⍰ 10 9 8 7 6 5 4 3 2 1

Print Numbers in Ascending Order

```
void printAsc(int n) {

    if ( n <= 0 ) //Base condition
           return;

    printAsc(n-1); //Recursive call

    cout << n << " "; //Prints number n
}
```

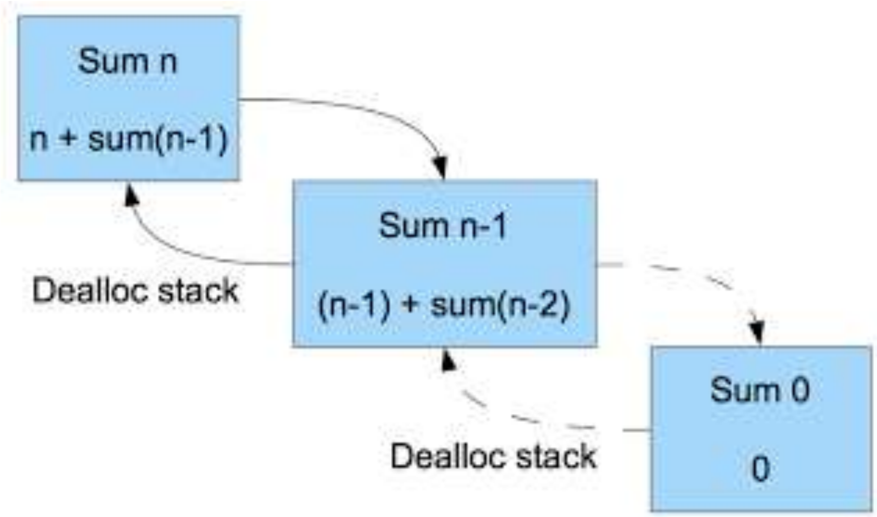
print(10) produces ⍰ 1 2 3 4 5 6 7 8 9 10

Example: Sum function (Iterative)

```
//Our initial total is zero
int total = 0;

//We want the sum from 1 + 2 + ... + 9 + 10
int n = 10;

/* The following for loop will calculate the
summation from 1 – n */
for ( int i = 1; i <= n; i++ ) {
     total = total + i;
}
```

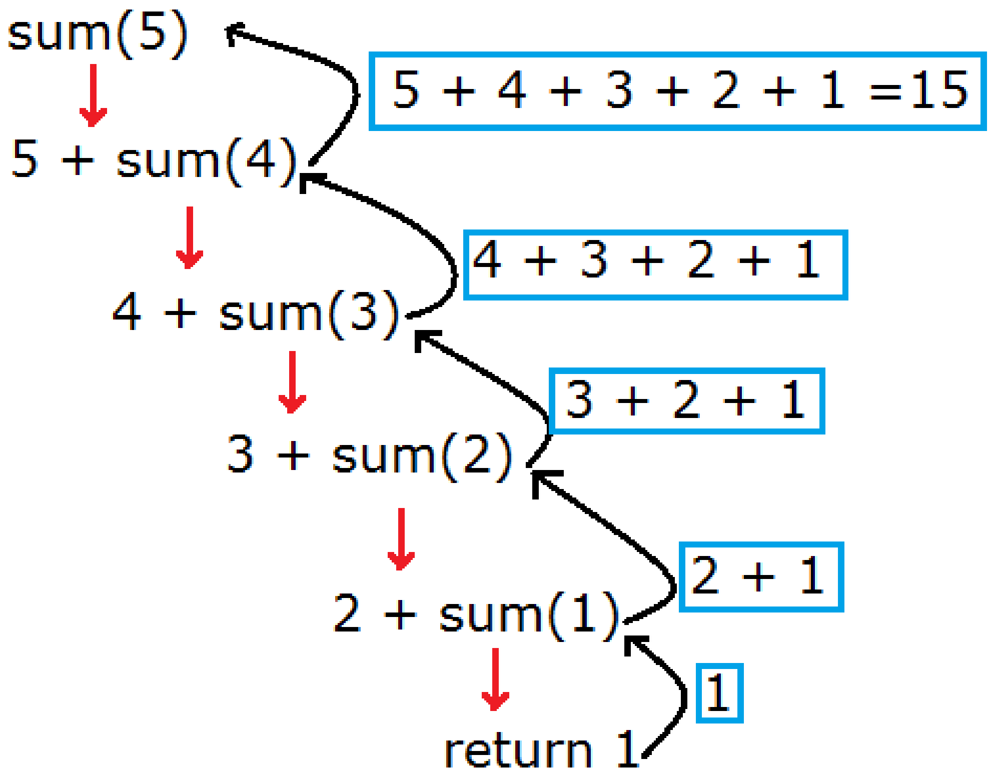
Example: Sum function (Recursive)

```
int sum(int n) {

     if ( n <= 0 ) //base case
          return 0;
     else //recursive call
         return n + sum(n-1);
}
```



Example: Sum Function

- sum(5) = 5 + 4 + 3 + 2 + 1 = 15



sum(5)

5 + sum(4)

4 + sum(3)

3 + sum(2)

2 + sum(1)

return 1

1

2 + 1

3 + 2 + 1

4 + 3 + 2 + 1

5 + 4 + 3 + 2 + 1 =15

Example: Multiply Function

- Multiplication is basically just addition

10 * 5 = 10 + 10 + 10 + 10 + 10 = 50

Added 10, 5 times

7 * 4 = 7 + 7 + 7 + 7 = 28

Added 7, 4 times

Write a recursive function that performs multiplication

Example: Multiply Function

```
int multiply (int n, int times){

 if(times == 0) //base case
     return 0;
 else           //recursive case
     return n + multiply(n,times-1);

}
```

Example: Factorial Function (1)

- A mathematical definition: For a non-negative integer n

$$fac(n) = \begin{cases} 1 & if\ n \leq 1 \\ n \times fac(n-1) & otherwise \end{cases}$$

- Factorial is defined in terms of itself
  - Defined in cases: a base case and a recursive case

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

Example: Factorial Function (2)

- Suppose the factorial function is invoked as `fac(5)`

```
fac(5)
5 * fac(4)
5 * 4 * fac(3)
5 * 4 * 3 * fac(2)
5 * 4 * 3 * 2 * fac(1)
5 * 4 * 3 * 2 * 1
5 * 4 * 3 * 2
5 * 4 * 6
5 * 24
120
```

Example: Character count

```
// Function prototype
int numChars(char, char*, int);

int main()
{
   char array[] = "abcddddef";

   /* Display the number of times the letter 'd'
  appears in the string. */

   cout << "The letter d appears "
   << numChars('d', array, 0) << " times.\n";

   return 0;
}
```

Example: Character count

```
int numChars(char search, char * str, int index) {

if (*(str + index) == '\0') {  //Base case
     return 0;
}

else if (*(str + index) == search){ //Recursive case
     return 1 + numChars(search, str,index+1);  }

else  {  // Recursive case
     return 0 + numChars(search, str, index+1);
     }
}
```

Example: Fibonacci Series

Fibonacci series:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, ...
```

- Except the first two numbers, each term is the sum of the two preceding terms

- **Recursive solution:**

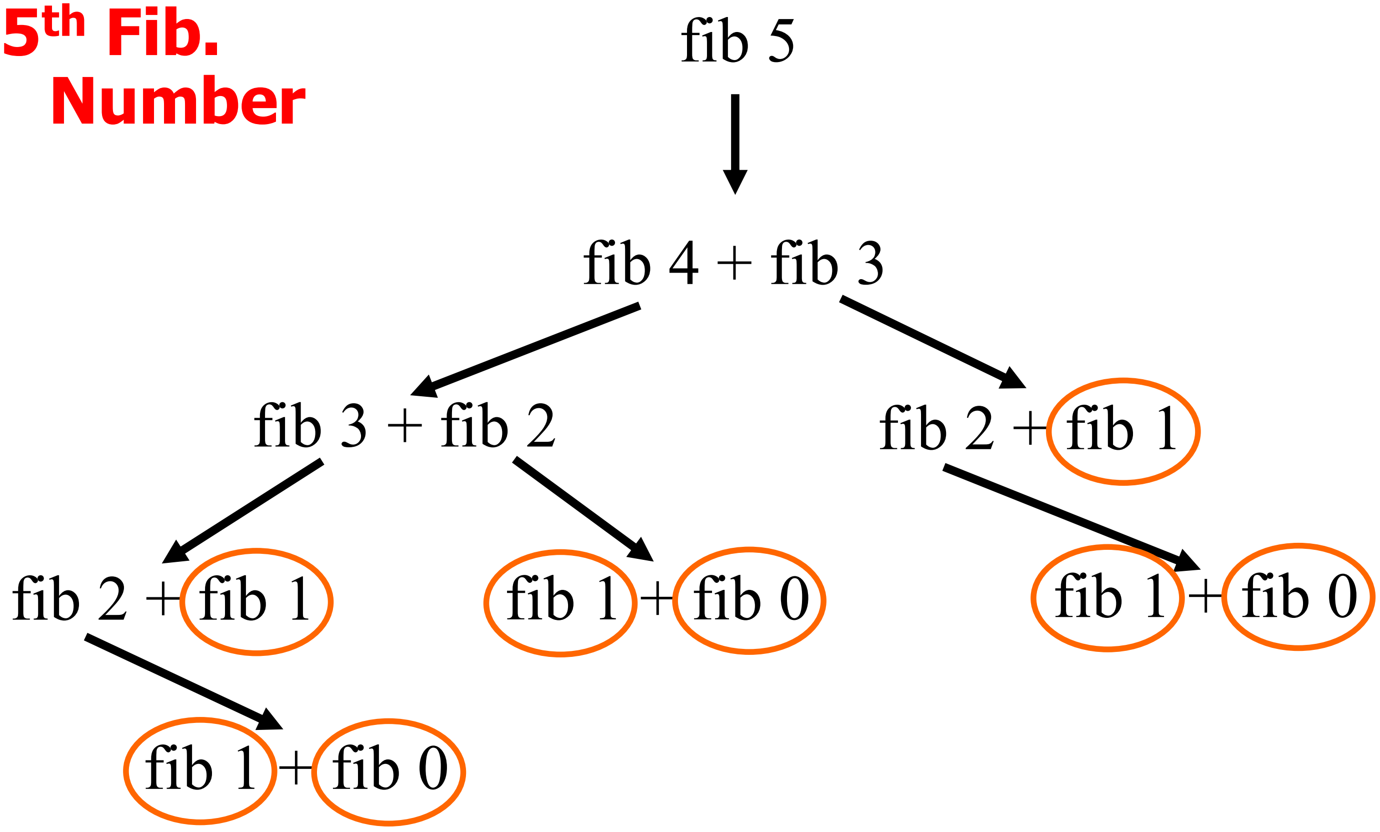
```
fib(n) = fib(n - 1) + fib(n - 2);
```

- **Base cases:**

```
n == 0, n == 1
```

Recursion

5th Fib. Number



fib 5

fib 4 + fib 3

fib 3 + fib 2

fib 2 + fib 1

fib 2 + fib 1

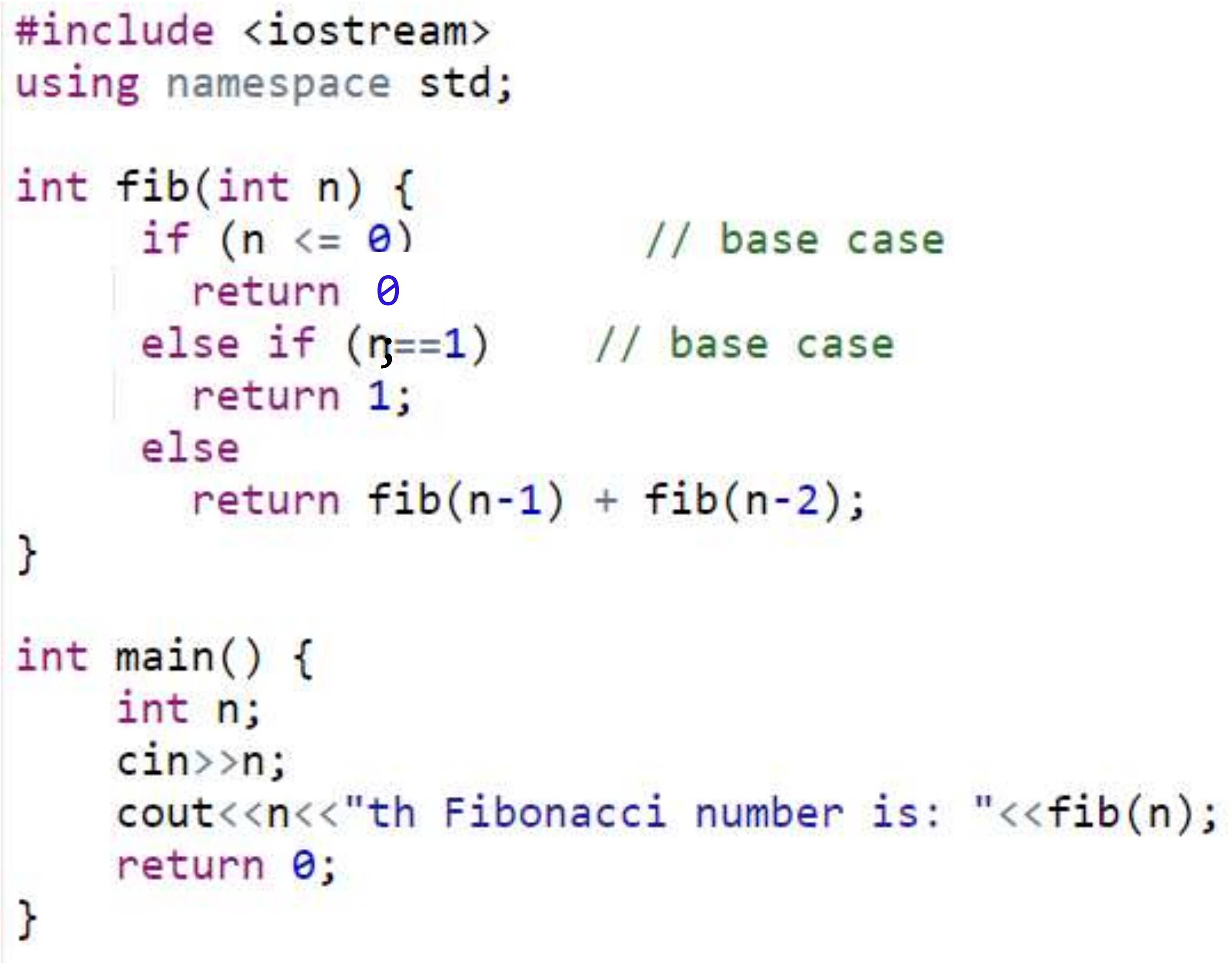
fib 1 + fib 0

fib 1 + fib 0

fib 1 + fib 0

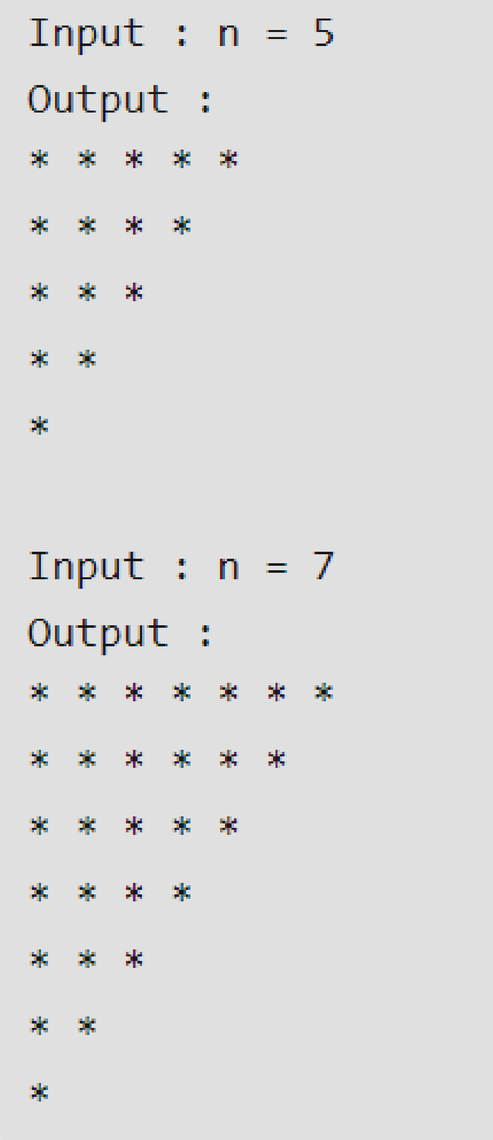
= 3

Recursive Fibonacci Function



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

int fib(int n) {
     if (n <= 0)             // base case
       return  0
     else if (n==1)     // base case
       return 1;
     else
       return fib(n-1) + fib(n-2);
}

int main() {
    int n;
    cin>>n;
    cout<<n<<"th Fibonacci number is: "<<fib(n);
    return 0;
}
```

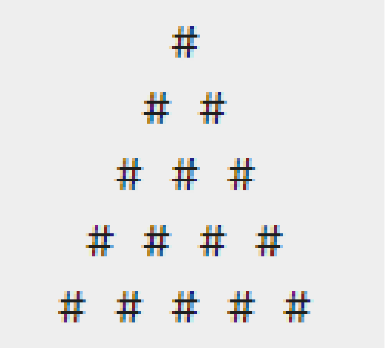
Printing Patterns using Recursion



```
Input : n = 5
Output :
* * * * *
* * * *
* * *
* *
*

Input : n = 7
Output :
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

```
void printCols(int nCols) {
    if (nCols > 0) {
        cout << "* ";
        printCols(nCols - 1);
    }
}
void printRows(int nrows) {
    if (nrows > 0) {
        printCols(nrows);
        cout << endl;
        printRows(nrows - 1);
    }
}
void main()
{     printRows(7);        }
```

Printing Patterns using Recursion



```
    #
   # #
  # # #
 # # # #
# # # # #
```

```
void printSpace(int nSpc) {
    if (nSpc > 0) {
        cout << " ";
        printSpace(nSpc - 1);
    }
}

void printCols(int nCols) {
    if (nCols > 0) {
        cout << “# ”;
        printCols(nCols - 1);
    }
}
```

```
printRows(int total,int n) {
    if (n > 0) {
        printSpace(n);
        printCols(total-(n-1));
        cout << endl;
        printRows(total,n - 1);
    }
}
int main()
{
    printRows(5,5);
    return 0;
}
```

Stack Overflow (1)

- Recursive functions cannot use statically allocated local variables
  - Each instance of the function needs its own copies of local variables

- Most modern languages allocate local variables for functions on the run-time stack
- Calling a recursive function many times or with large arguments may result in stack overflow

```
$ java Fac 10000
Exception in thread "main" java.lang.StackOverflowError
at Fac.facIter(Fac.java:35)
at Fac.facIter(Fac.java:38)
at Fac.facIter(Fac.java:38)
...
```

Recursion vs Iteration

- Recursive algorithms can also be coded with iterative control structures, but which is best to use?

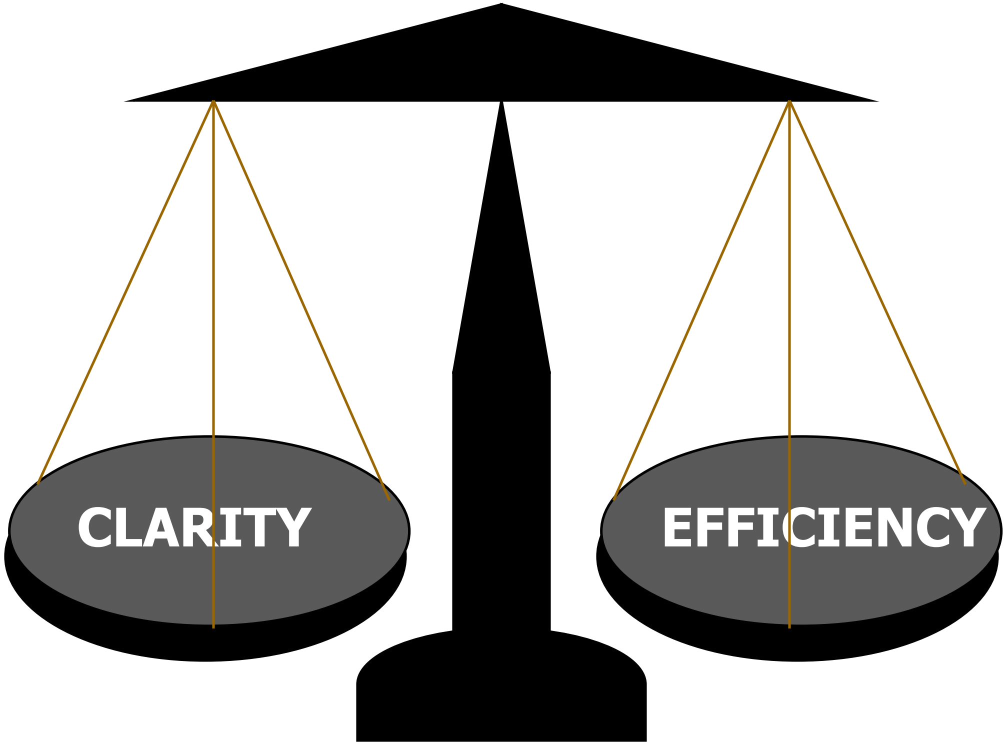
- There are advantages and disadvantages to each approach

- Recursion disadvantages:
  - Less efficient than iterative algorithms.
  - Majority of repetitive programming tasks are best done with loops.
- Recursion advantages:
  - Code clarity
  - Easier to code some algorithms using recursion, e.g. quicksort

Recursion or Iteration?

- If there is sufficient memory available for recursion

- Primarily a design decision. If a problem is more easily solved with a loop, that should be the approach you take. If recursion results in a better design, that is the choice you should make.



Any Question So Far?