CS 2050
Computer Science II

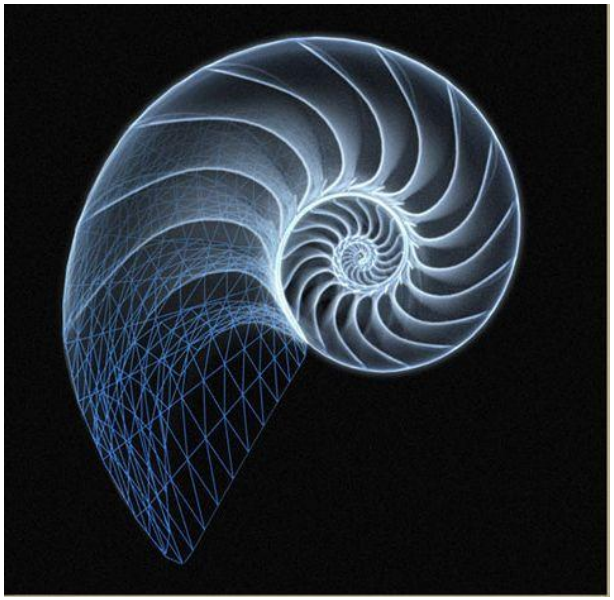
Lesson 06

METROPOLITAN STATE UNIVERSITY℠
OF DENVER
LIVES TRANSFORMED

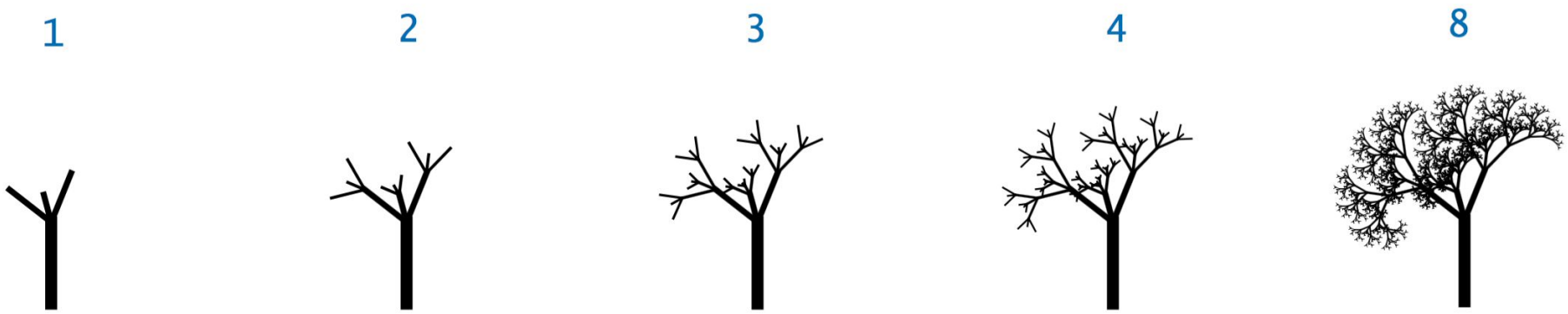
Agenda

- Recursion
- The Call Stack

Recursion



Recursion



Recursion

- Recursion occurs when you have a definition that uses itself
- The most common applications of recursion are found in mathematics and CS
- We say that a function has a recursive definition if the function is applied to define itself

Recursion

- Example - Factorial:

```
n! = n x (n-1) x … x 1
```

Recursion

- Example - Factorial:

```
n! = n * (n - 1)!
```

Recursion

- Example - Factorial:

```
n! = n * (n - 1)!, if n > 0

n! = 1, if n = 0
```

Recursion

- Example - Factorial:

```
n! = n * (n - 1)!, if n > 0

n! = 1, if n = 0
```

base case or
exit condition

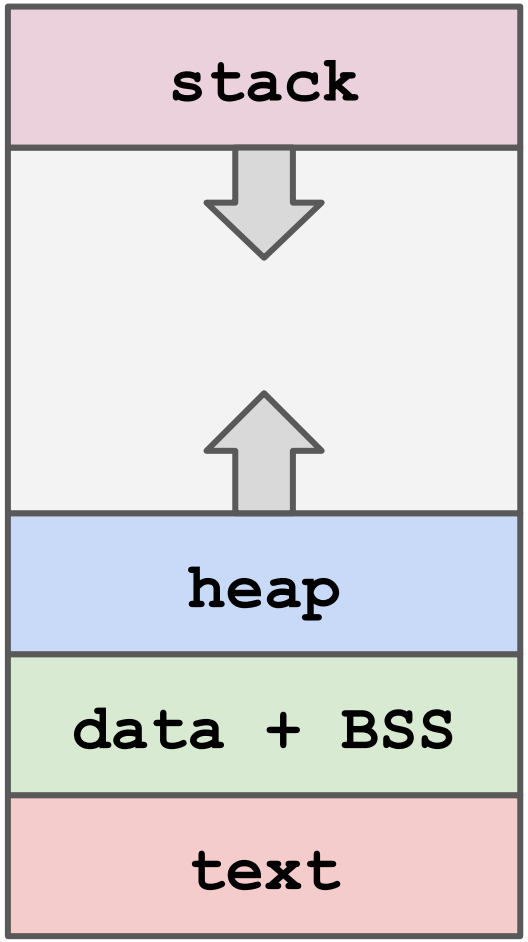
Recursion

The Call Stack

```
static int factorial(int n) {
    if (n == 0)
        return 1;
    return n * factorial( n: n - 1);
}
```

The Call Stack

- The call stack is an internal data structure used by the O.S. to keep track of function calls in a program
- Other names: execution stack, run-time stack, program stack, control stack



Memory Segments of a Running Program

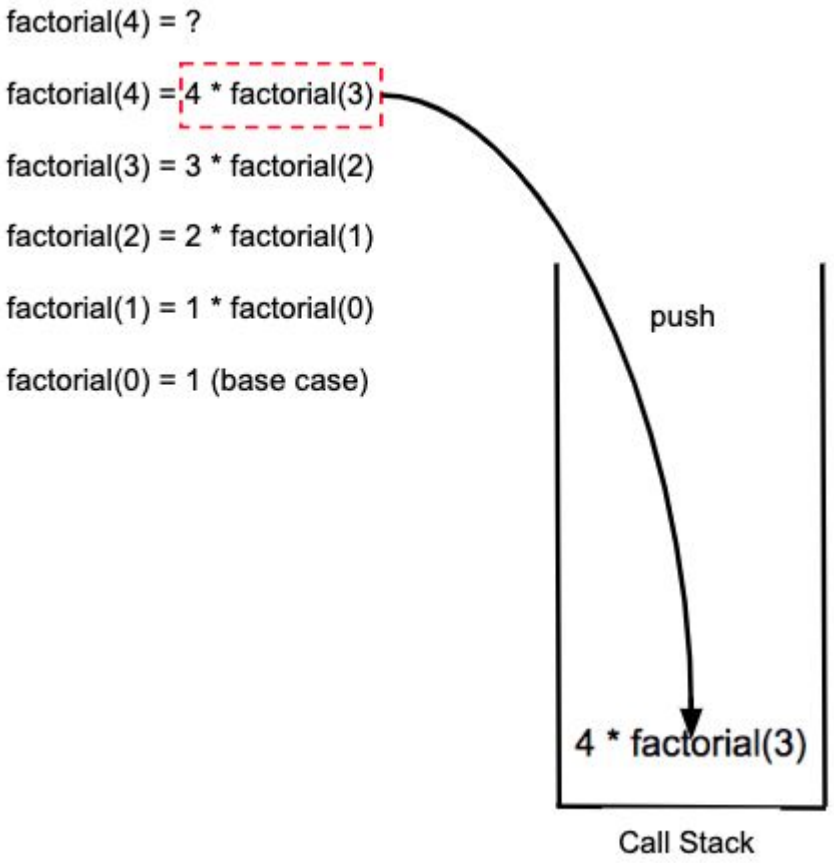
The Call Stack

- Its main purpose is to keep track of the point to which each active function should return control when it finishes executing
- The call stack is made of “stack frames”

The Call Stack

- A new “stack frame” is created and pushed onto the call stack every time your program makes a function call
- Conversely, a “stack frame” is popped out of the call stack whenever a function in your program resumes execution (i.e. return)

The Call Stack



The Call Stack

factorial(4) = ?

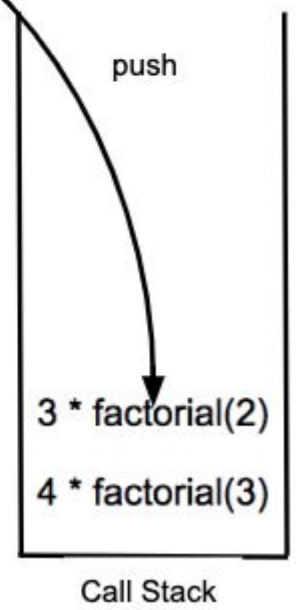
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



The Call Stack

factorial(4) = ?

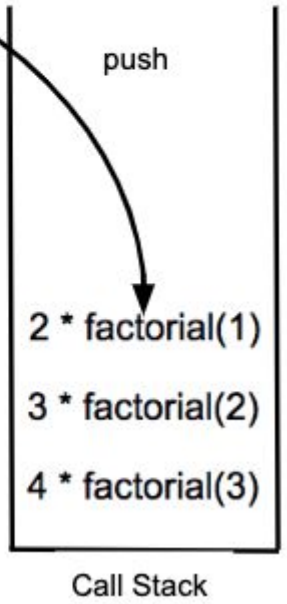
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



The Call Stack

factorial(4) = ?

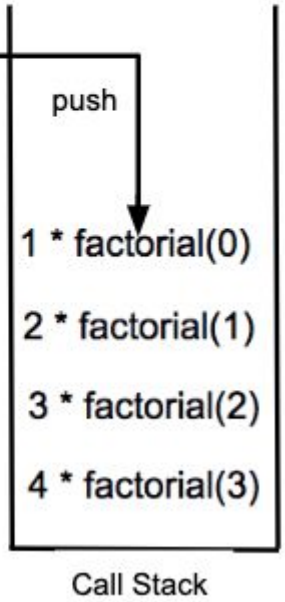
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



The Call Stack

factorial(4) = ?

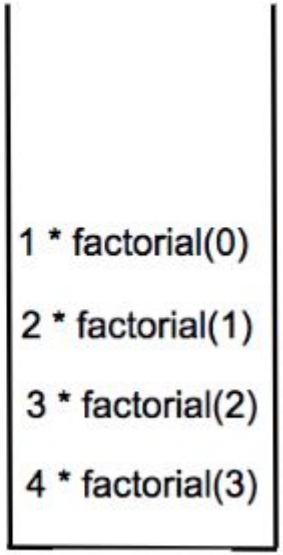
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * factorial(0)

factorial(0) = 1 (base case)



Call Stack

The Call Stack

factorial(4) = ?

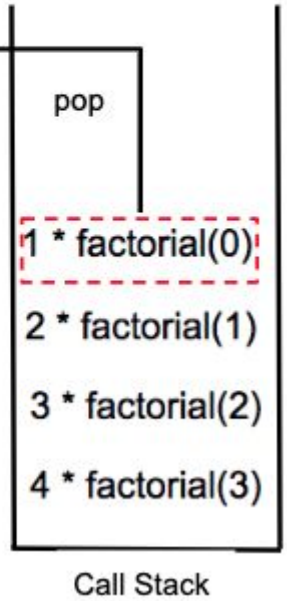
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * factorial(1)

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



The Call Stack

factorial(4) = ?

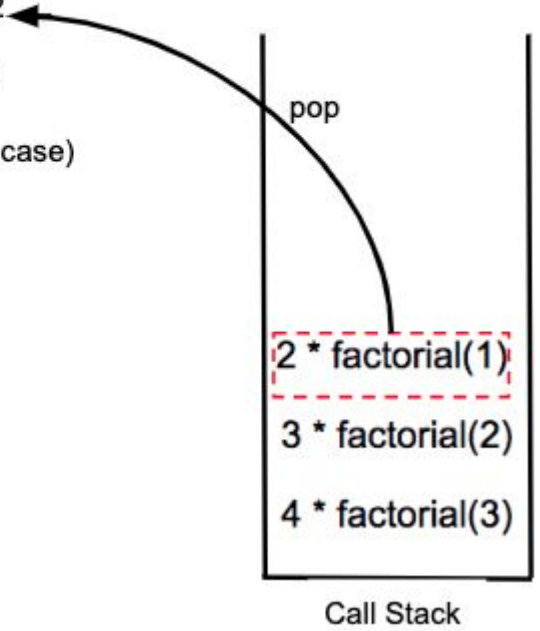
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * factorial(2)

factorial(2) = 2 * 1 = 2

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



The Call Stack

factorial(4) = ?

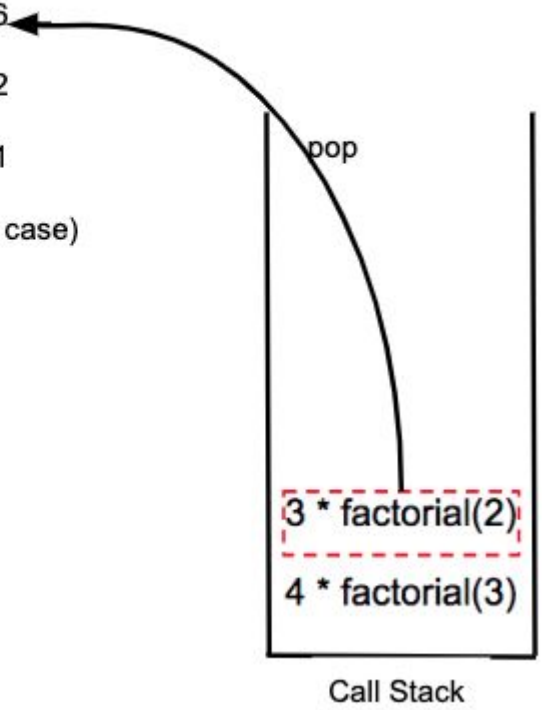
factorial(4) = 4 * factorial(3)

factorial(3) = 3 * 2 = 6

factorial(2) = 2 * 1 = 2

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



The Call Stack

factorial(4) = ?

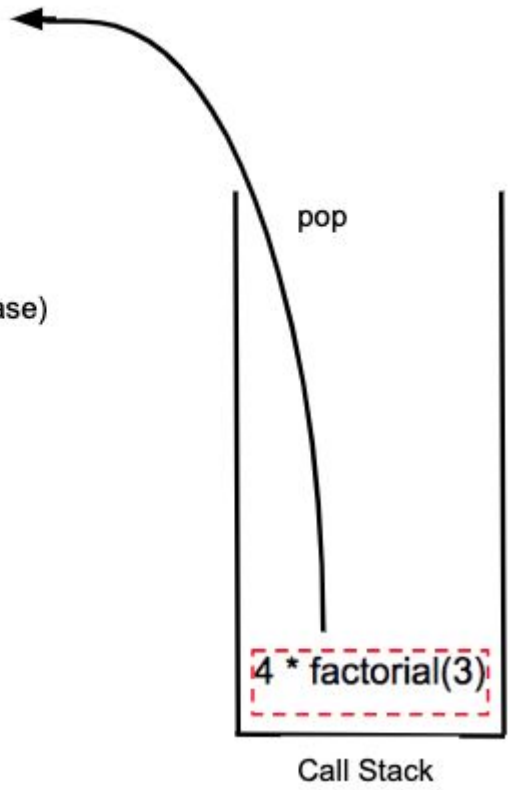
factorial(4) = 4 * 6 = 24

factorial(3) = 3 * 2 = 6

factorial(2) = 2 * 1 = 2

factorial(1) = 1 * 1 = 1

factorial(0) = 1 (base case)



The Call Stack

- The “stack overflow” error occurs when the call stack is exhausted because of the number of function calls made recursively without the function ever reaching its base case