



Oregon State  
University

COLLEGE OF ENGINEERING | School of Electrical Engineering  
and Computer Science

# CS 161

## Introduction to CS I

### Lecture 21

- Recursion



# Assignment 5 – Treasure Chest

- Define your own struct (type) with at least 4 attributes
- Create a program to store items of that type in a treasure chest and keep track of the total collection value
- User can:
  - Add item
  - Remove item
  - Display item
  - Add an item with random properties
  - <Your choice here>

No live demo  
(README.txt instead)

# Midterm 2

- Midterm 2: content through **week 7** (but no structs)
- Review questions (and solutions) are on course website
- Bring your questions to class on Wednesday
  - Stuck on pointers? Functions? 2D arrays?
- Review session: Thursday 2/27, 6-7 p.m., **LINC 228**
- Midterm: Friday 2/28, 2-2:50 p.m., **LINC 100**
- Format: true/false, multiple choice, one page short answer
  - Scantron sheet: fill in bubbles with #2 pencil
- Bring to midterm: **student ID and #2 pencil(s)**

# Let's calculate factorials

- Mathematical definition

$$0! := 1;$$

$$\begin{aligned} n! &:= n * (n-1) * \dots * 1 \\ &:= n * (n-1)! \quad \text{if } n > 0 \end{aligned}$$

# Iterative factorial

**Iterative definition:**

$\text{factorial}(0) := 1;$

$\text{factorial}(n) := n * (n-1) * (n-2) * \dots * 3 * 2 * 1;$

See lec21-factorial-iterative.cpp

```
1. int factorial(int n) {  
2.     int fact;  
3.     if (n==0)  
4.         fact = 1;  
5.     else  
6.         for (fact=n; n > 1; n--)  
7.             fact = fact * (n-1);  
8.     return fact;  
9. }
```

CS 161

# Computing Factorial Iteratively

```
factorial(4) = 4 * 3  
            = 12 * 2  
            = 24 * 1  
            = 24
```

```
factorial(0) = 1;  
factorial(n) = n*(n-1)*...*2*1;
```

# Recursion

- A recursive definition includes a mention of itself
  - "My descendants are my children + all of my children's descendants."
  - "My keys are located in this room or in some other room."
- A recursive function includes [at least one] call to itself
  - Base case: when to stop (simplest case)
  - Recursive step: a general statement that reduces the task (eventually) to a base case

# Recursive Factorial

See lec21-factorial-recursive.cpp

## Recursive definition:

**Base case:**  $\text{factorial}(0) = 1$ ;

**Recursive step:**  $\text{factorial}(n) = n * \text{factorial}(n-1)$ ;

```
1. int factorial(int n) {  
2.     if (n == 0) /* Base case */  
3.         return 1;  
4.     else  
5.         /* recursive call */  
6.         return n * factorial(n - 1);  
7. }
```

# Computing Factorial Recursively

```
factorial(4) = 4 * factorial(3)
              = 4 * (3 * factorial(2))
              = 4 * (3 * (2 * factorial(1)))
              = 4 * (3 * ( 2 * (1 * factorial(0))))
              = 4 * (3 * ( 2 * (1 *1)))
              = 4 * (3 * ( 2 * 1))
              = 4 * (3 * 2)
              = 4 * 6
              = 24
```

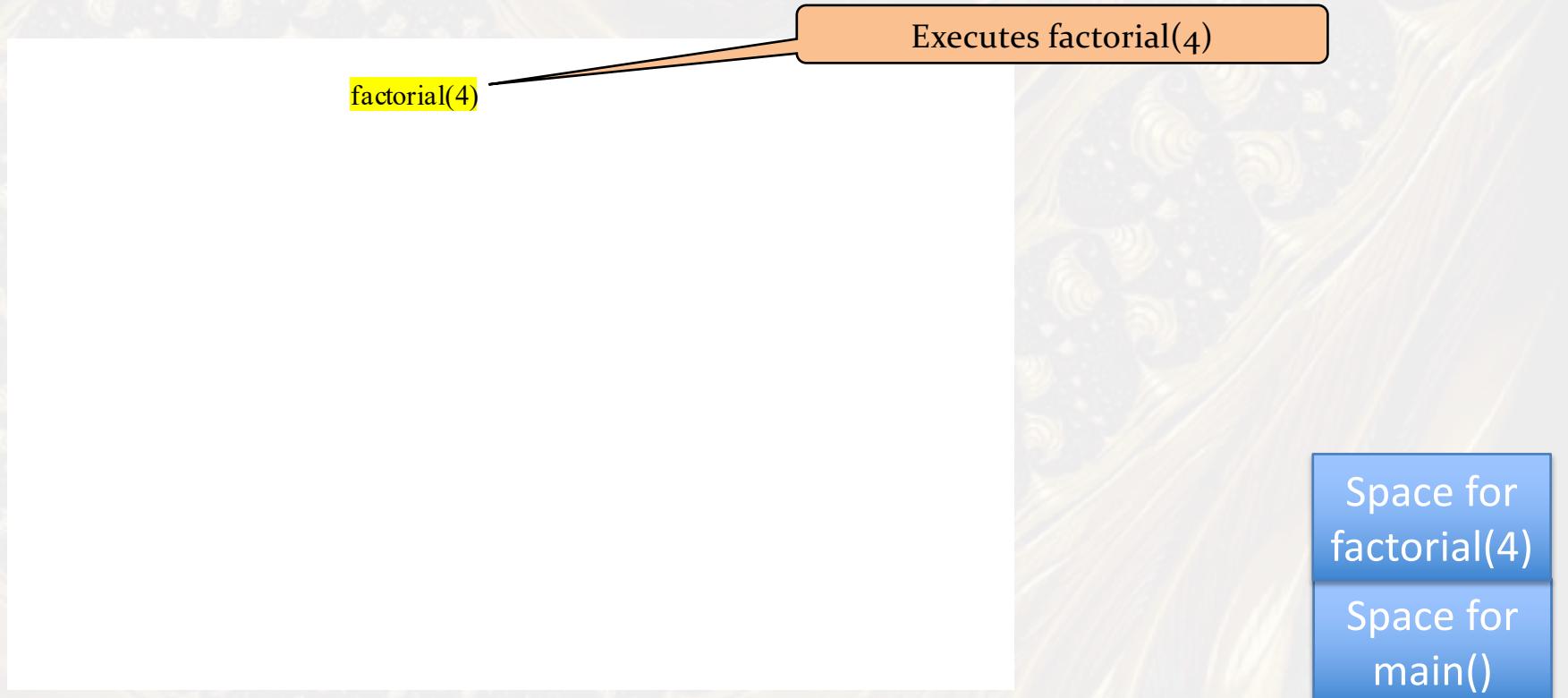
```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

# Differences

- Pros
  - Readability
- Cons
  - Efficiency
  - Memory

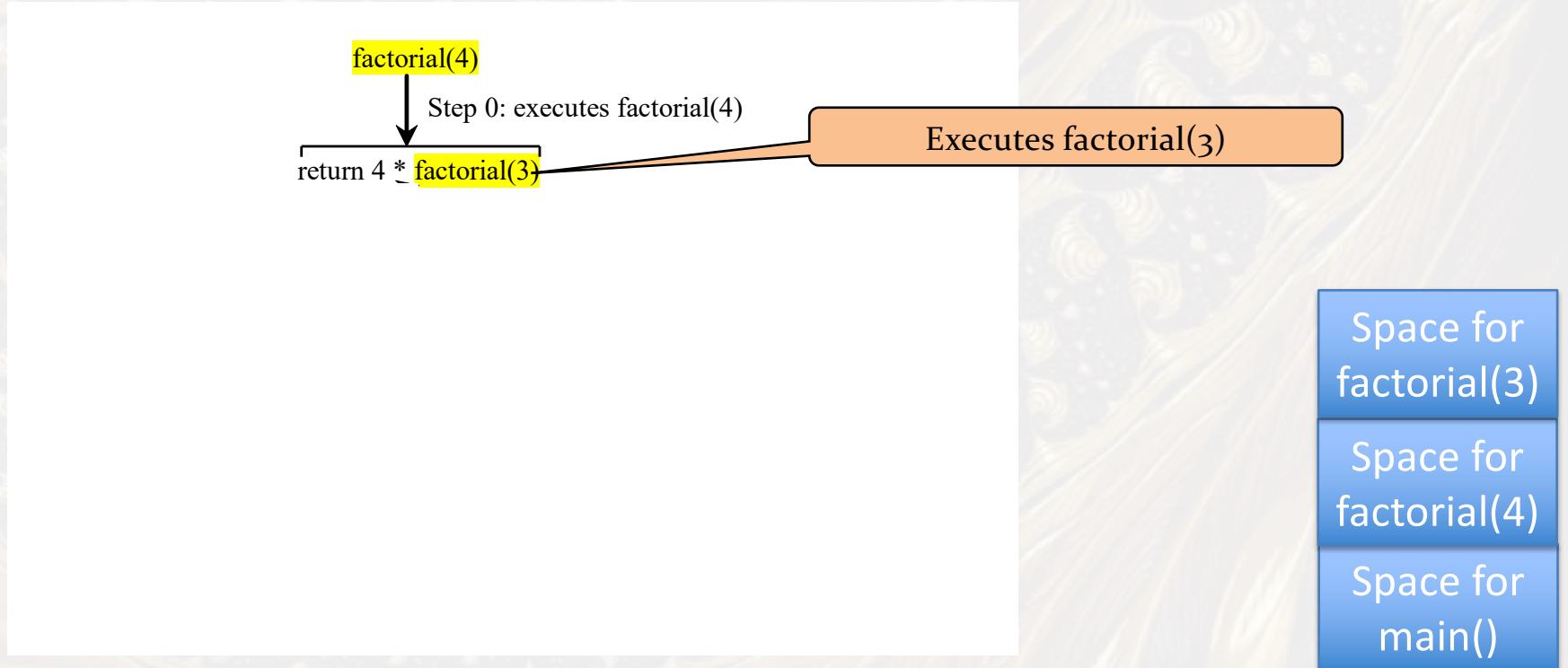
# Recursive Factorial

Stack



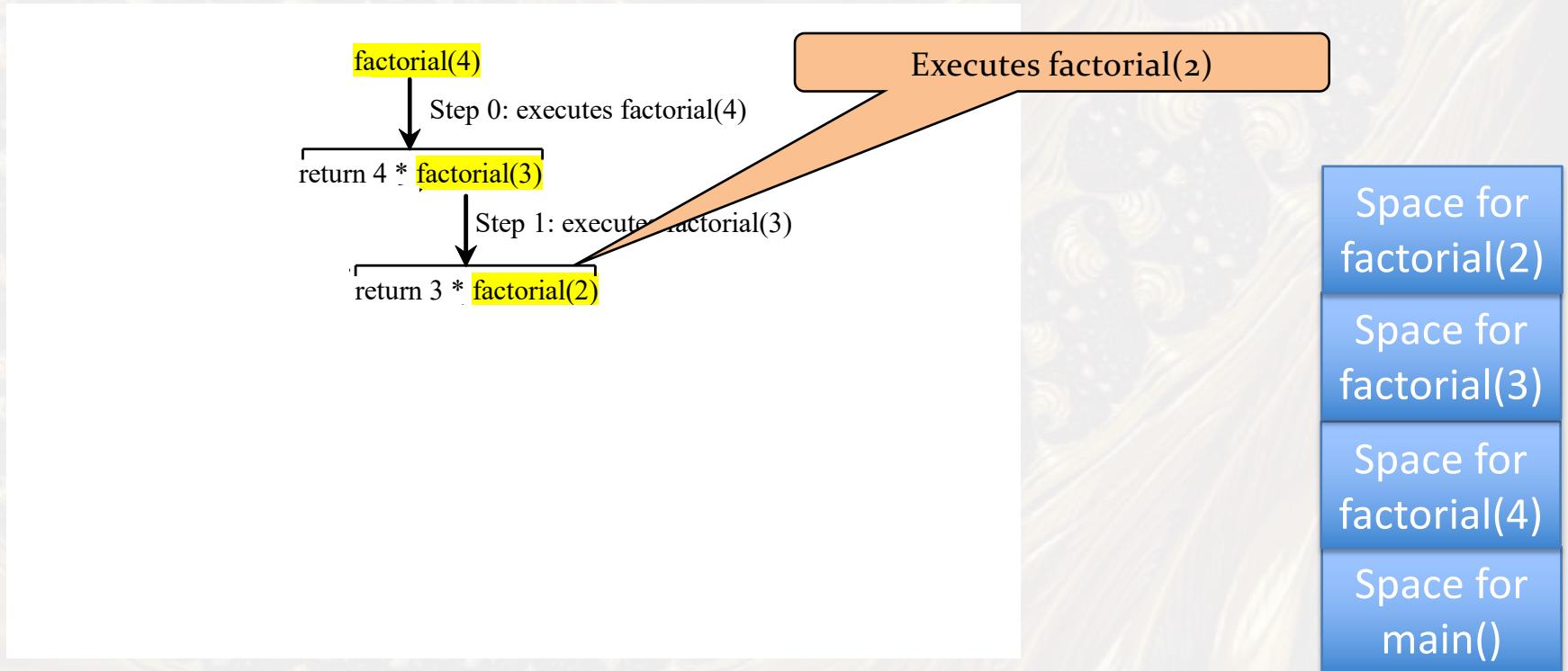
# Recursive Factorial

Stack



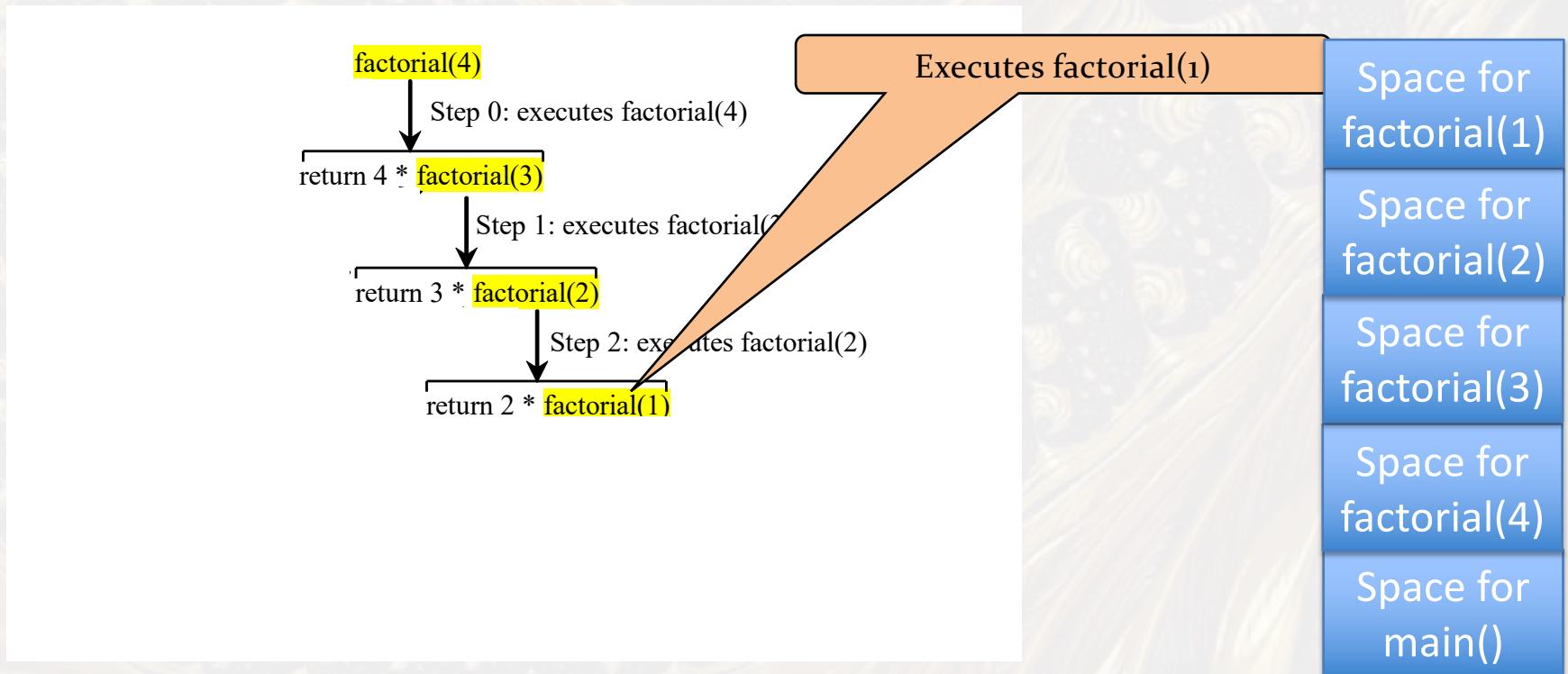
# Recursive Factorial

Stack



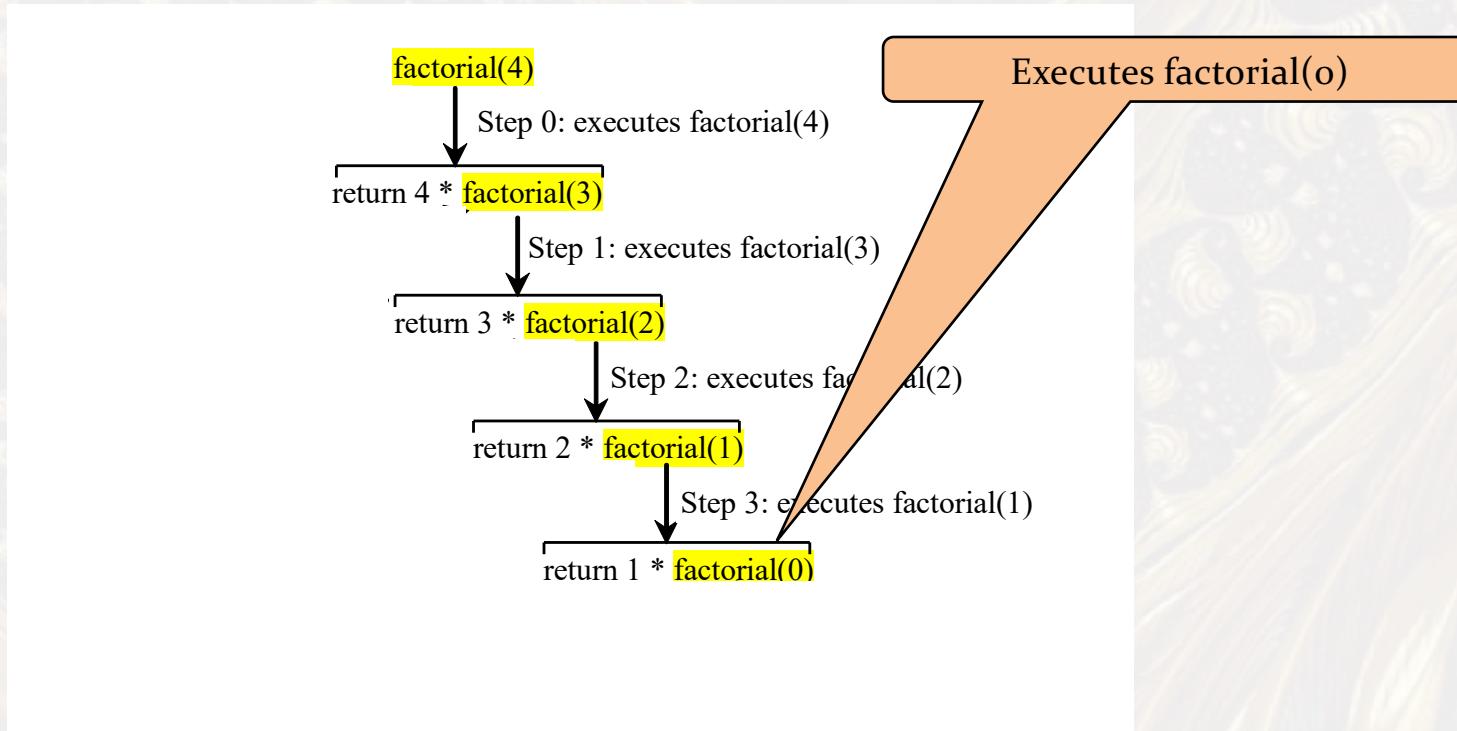
# Recursive Factorial

Stack



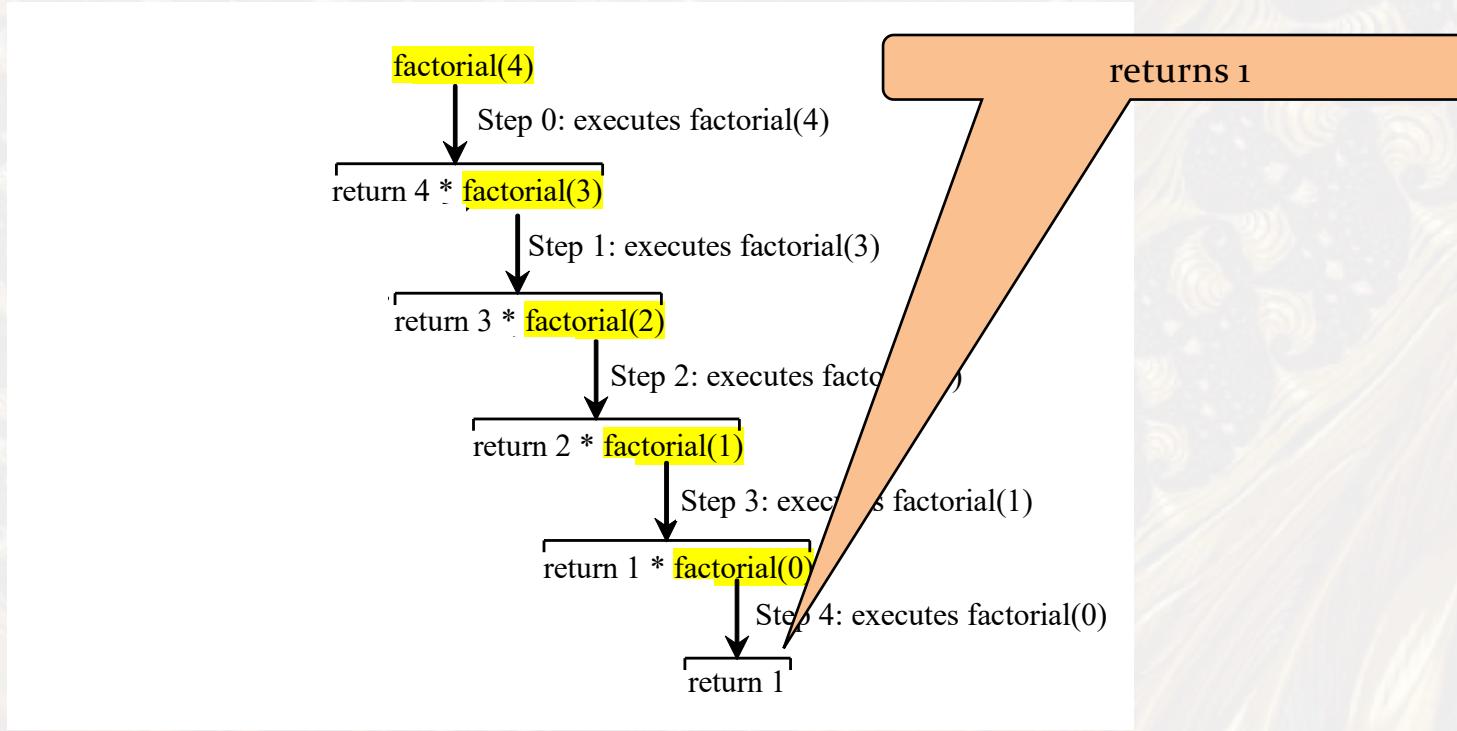
# Recursive Factorial

Stack



# Recursive Factorial

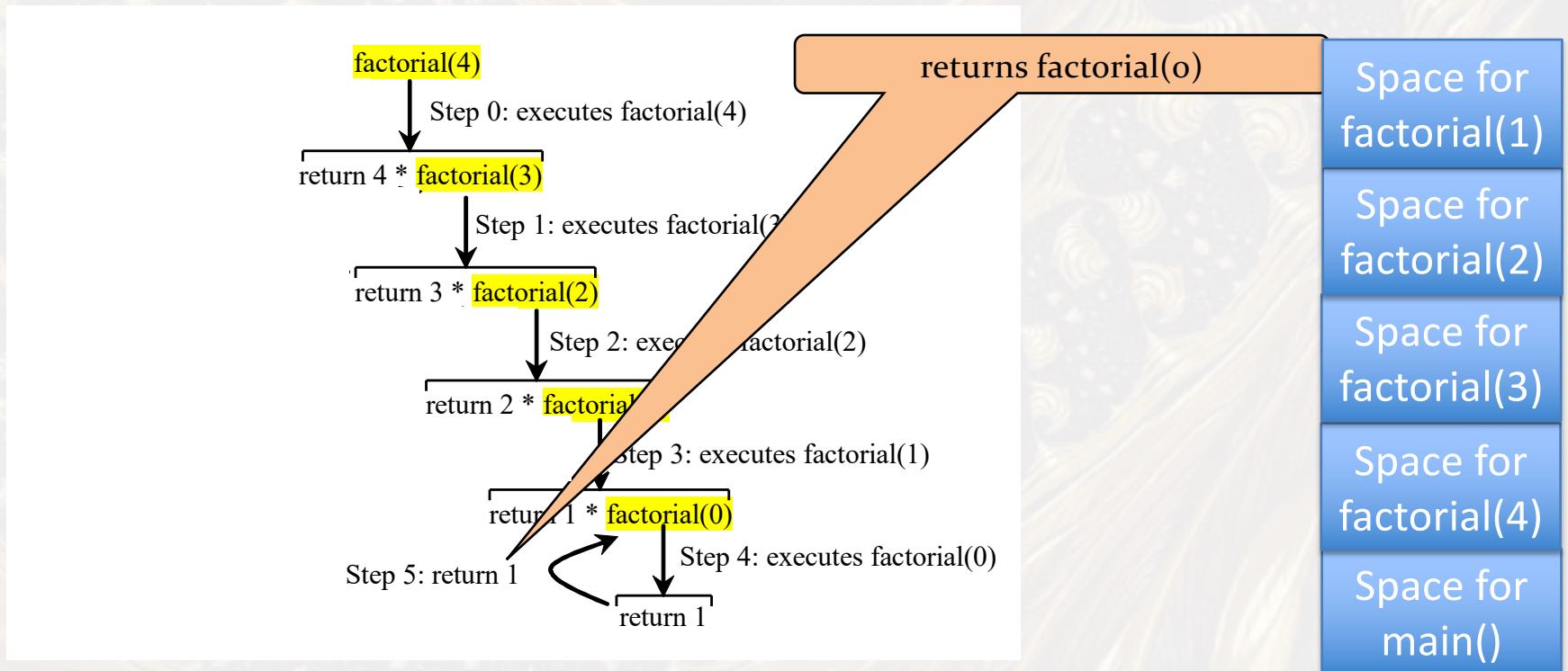
**Stack**



Space for factorial(0)
Space for factorial(1)
Space for factorial(2)
Space for factorial(3)
Space for factorial(4)
Space for main()

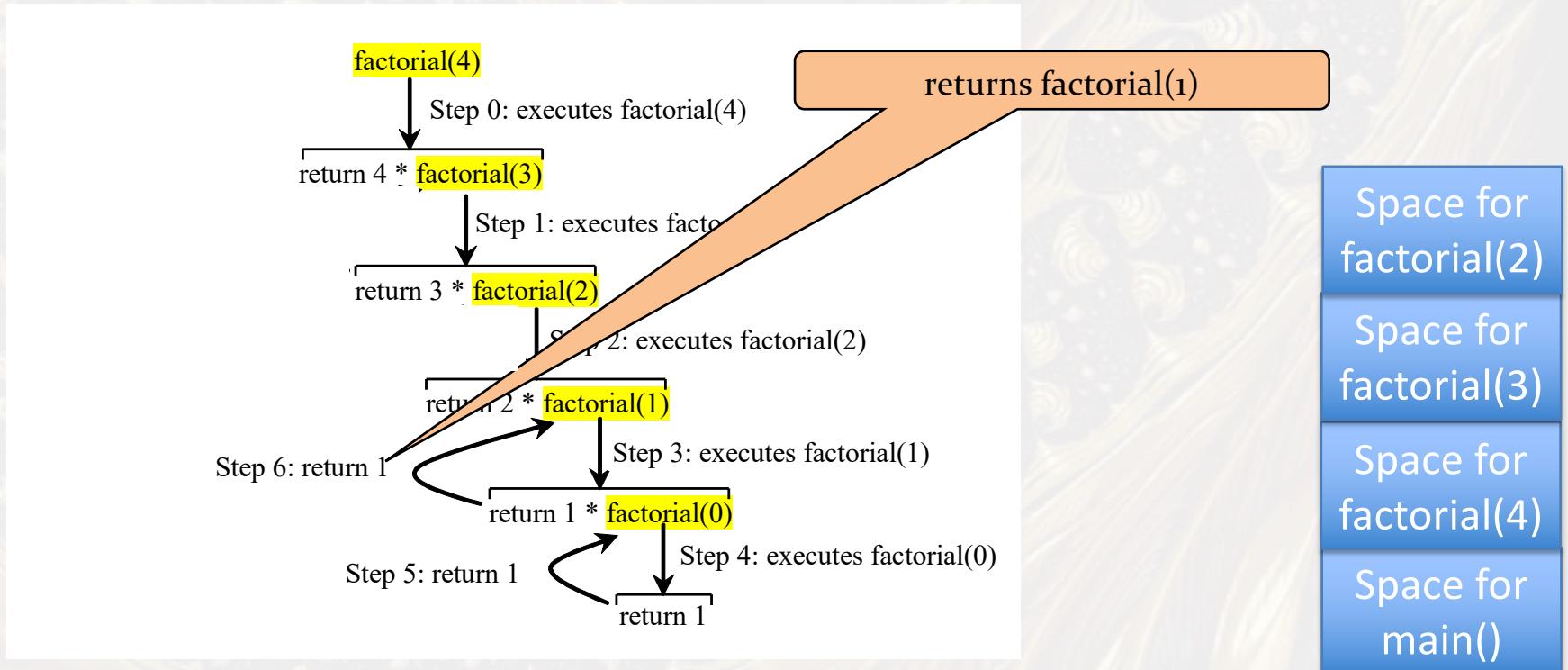
# Recursive Factorial

Stack



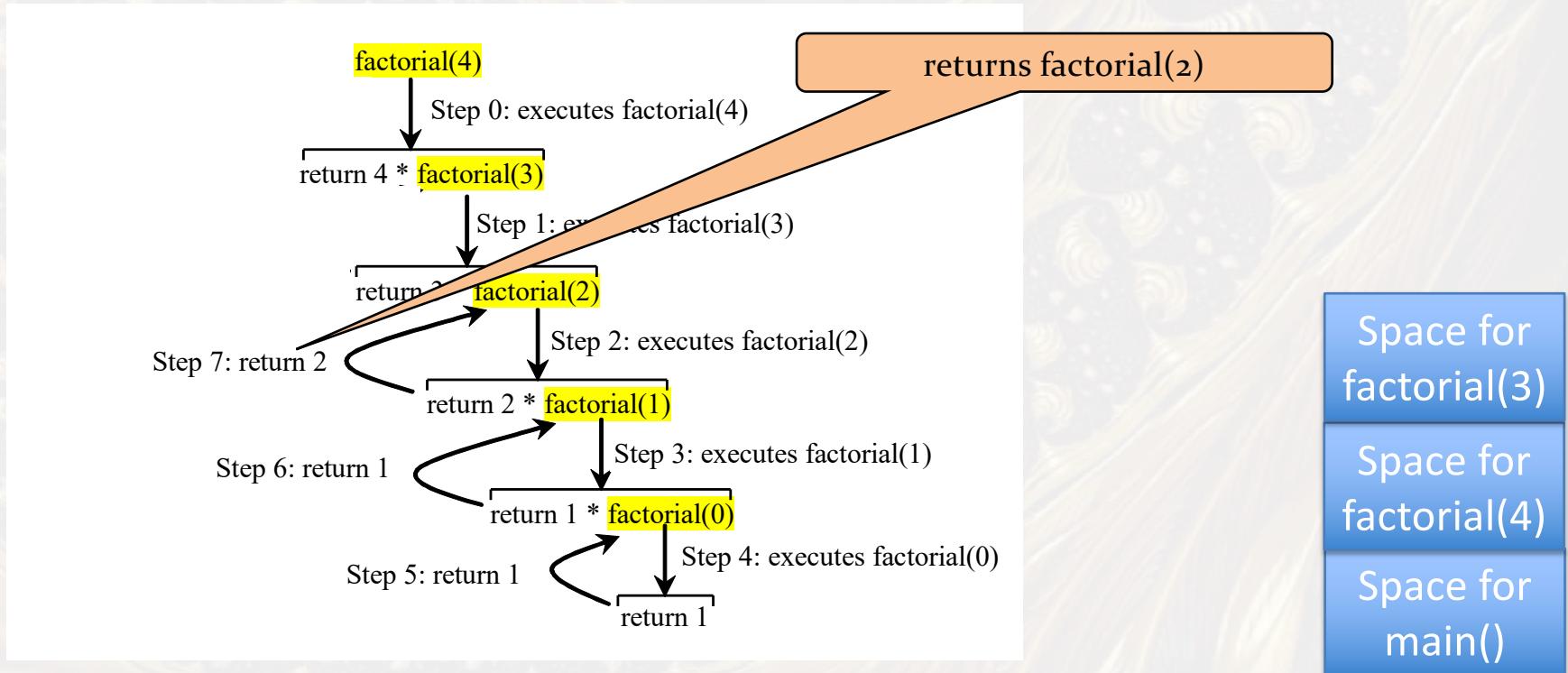
# Recursive Factorial

Stack



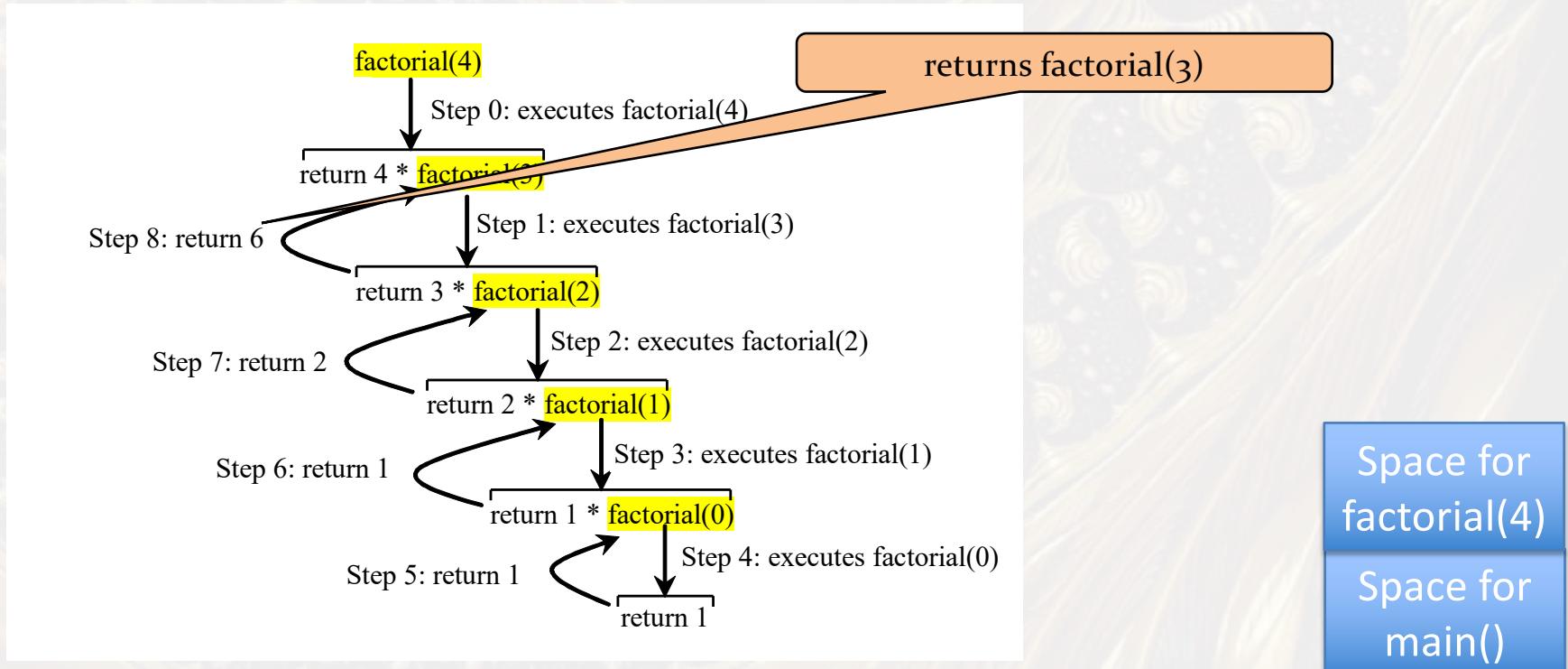
# Recursive Factorial

Stack



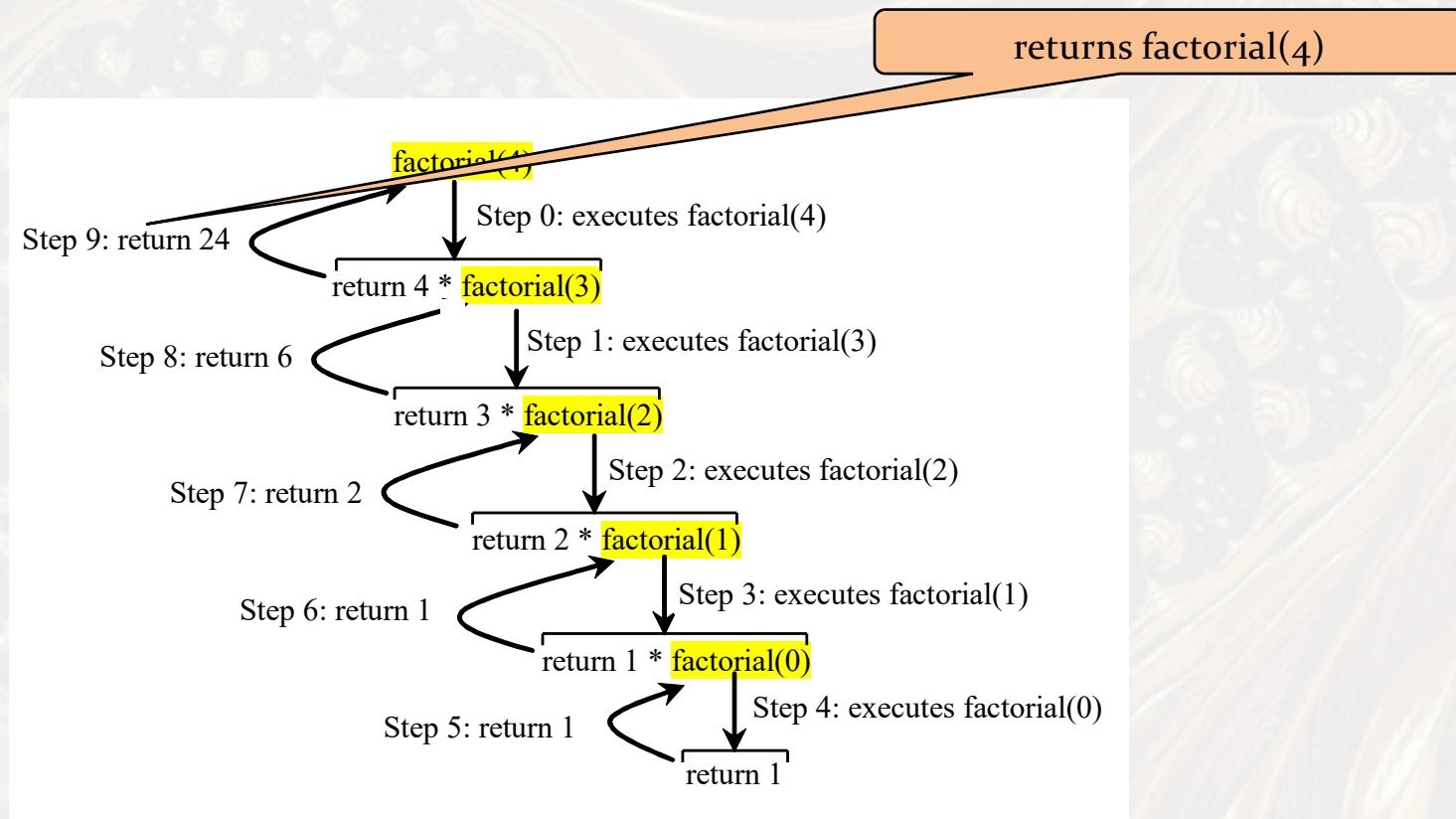
# Recursive Factorial

Stack



# Recursive Factorial

Stack



# When is recursion useful?

- Problems that have a "nested" or recursive structure and would be hard to write in an iterative fashion
  - Water in a river is the sum of the water from each tributary
- Recursion breaks the problem into one small step + "the rest of the solution"



## Your turn: descendants

- "My number of descendants is my number of children + the sum of my children's descendants."
  - What is the **base case**?
  - What is the **recursive step**?

## Your turn: descendants

- "My number of descendants is my number of children + the sum of my children's descendants."
  - What is the **base case**?
    - No children: #descendants = 0
  - What is the **recursive step**?
    - $\# \text{descendants} = \# \text{children} + \# \text{descendants}(\text{child1}) + \# \text{descendants}(\text{child2}) + \dots$
  - This would be quite difficult to do in an iterative way!

## Your turn: exponents

- Compute **base<sup>exp</sup>** in a recursive function
  - What is the function prototype?
  - What is the **base case**?
  - What is the **recursive step**?

## Your turn: exponents

- Compute **base<sup>exp</sup>** in a recursive function called pwr()
  - What is the function prototype?
    - int pwr(int base, int exp);
  - What is the **base case**?
  - What is the **recursive step**?

## Your turn: exponents

- Compute **base<sup>exp</sup>** in a recursive function called pwr()
  - What is the function prototype?
    - int pwr(int base, int exp);
  - What is the **base case**?
    - exp = 0: return 1
  - What is the **recursive step**?
    - exp > 0: return base \* pwr(base, exp-1)

# Exponent implementation

See `lec21-power-recursive.cpp`

```
1. int pwr(int base, int exp) {  
2.     if (exp == 0) /* base case */  
3.         return 1;  
4.     else  
5.         /* recursive call */  
6.         return base * pwr(base, exp - 1);  
7. }
```

# Gotchas

- Failure to specify base case => stack overflow
- Failure to reach base case => stack overflow
  - Problem doesn't get smaller

```
int myfun(int n) {  
    if (n == 0)  
        return 0;  
    else  
        return myfun(n);  
}
```

# What vocabulary did we learn today?

- Recursion
- Base case
- Inductive step

# What ideas and skills did we learn today?

- How to design solutions with recursive definitions
- How to translate a recursive definition into a recursive function
- Merits of iteration versus recursion

# Week 8 begins!

- Attend lab (laptop required)
- Read Rao lesson 7 (pp. 158-161)

Read Miller lecture 8:

<http://www.doc.ic.ac.uk/~wjk/C++Intro/RobMillerL8.html>

- Start on design for **Assignment 5** (due **Sunday, March 1**)

See you Wednesday (midterm review)!

- Bring your questions about material from weeks 1-7