Chapter 18 Recursion

Original slides by
Daniel Liang
Modified slides by
Anas Arram

Motivations

To use recursion is to program using recursive methods—that is, to use methods that invoke themselves. Recursion is a useful programming technique.

In some cases, it enables you to develop a natural, straightforward, simple solution to an otherwise difficult problem.

This chapter introduces the concepts and techniques of recursive programming and illustrates with examples of how to "think recursively."

Recursion

Find the sum of numbers, from 0 to *n*.

Loop

```
int sum(int n){
    int sum = 0;
    for(int i = 0; i < n; i++){
        sum += i;
    }
    return sum;
}
```

Recursion

```
int sum(int n){
    if(n==0){
        return 0;
    }
    return n + sum(n-1);
}
```

Recursion

n = 5

```
int sum(int n){
    if(n==0){
        return 0;
    }
    return n + sum(n-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

```
int sum(int 0){
    if(0==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

```
int sum(int 0){
    if(0==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

0

```
int sum(int 0){
    if(0==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

0

0

```
int sum(int 0){
    if(0==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

Recursion

```
int sum(int 5){
    if(5==0){
        return 0;
    }
    return 5 + sum(5-1);
}
```

```
int sum(int 4){
    if(4==0){
        return 0;
    }
    return 4 + sum(4-1);
}
```

```
int sum(int 3){
    if(3==0){
        return 0;
    }
    return 3 + sum(3-1);
}
```

```
int sum(int 2){
    if(2==0){
        return 0;
    }
    return 2 + sum(2-1);
}
```

1

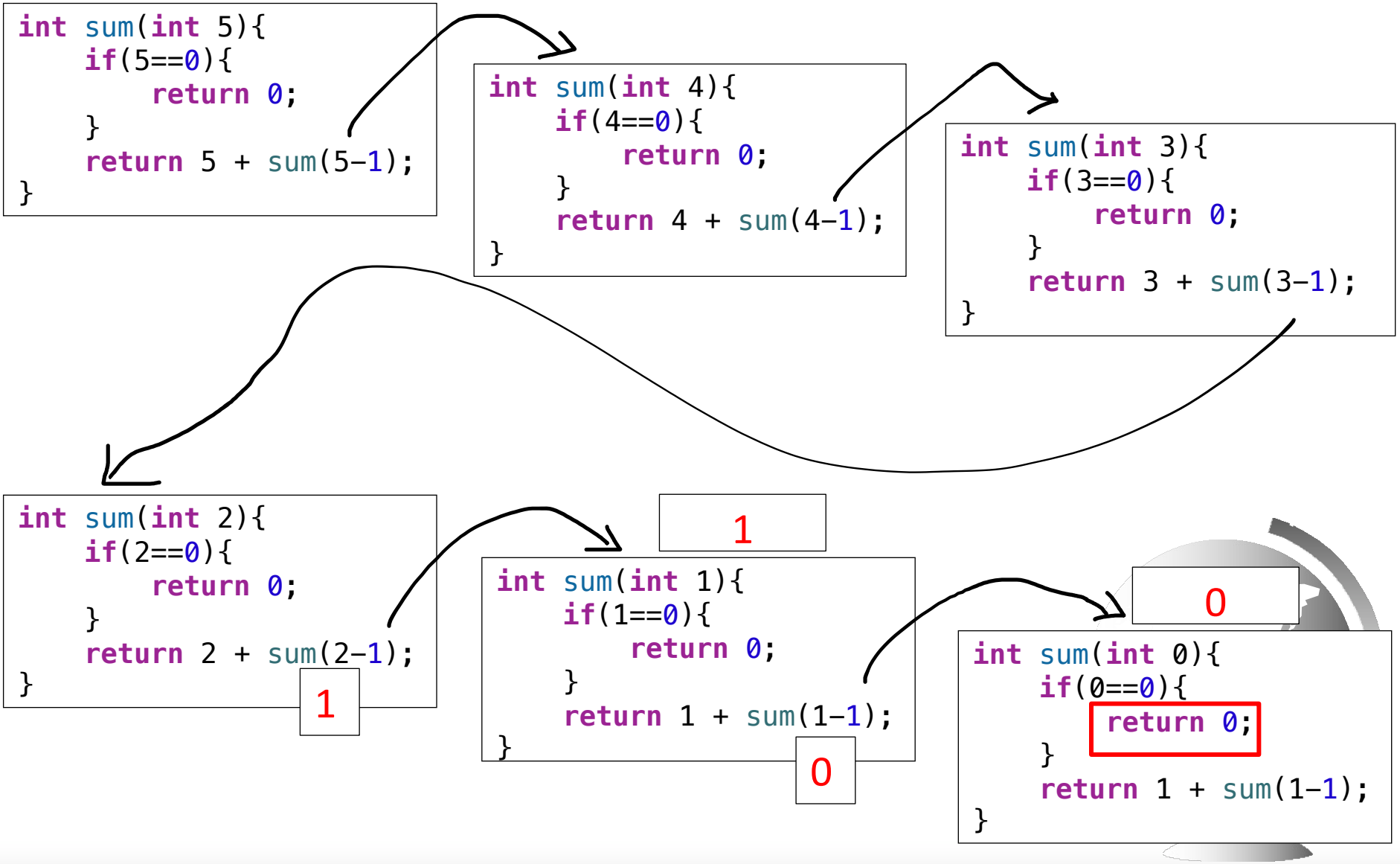
```
int sum(int 1){
    if(1==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

0

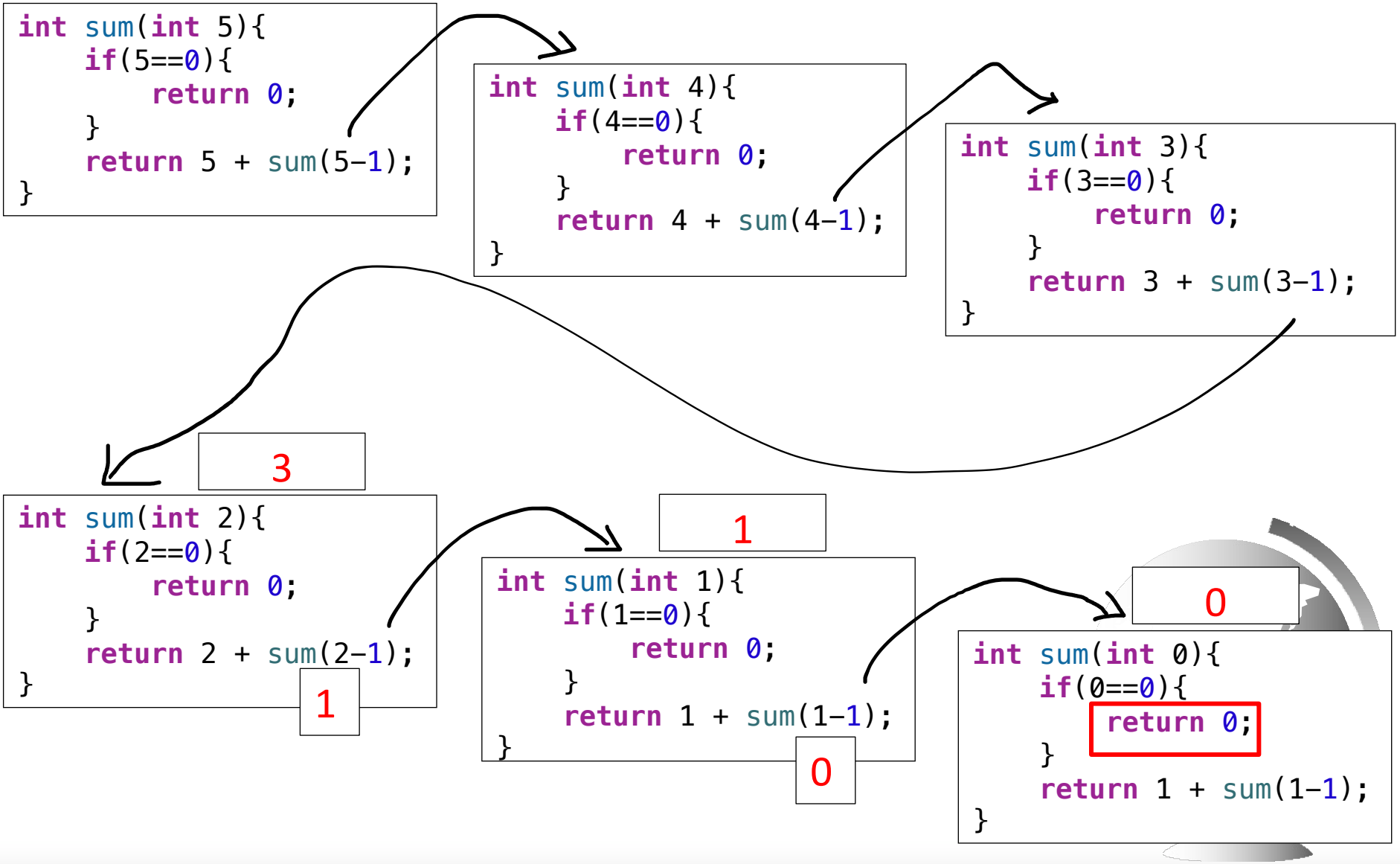
0

```
int sum(int 0){
    if(0==0){
        return 0;
    }
    return 1 + sum(1-1);
}
```

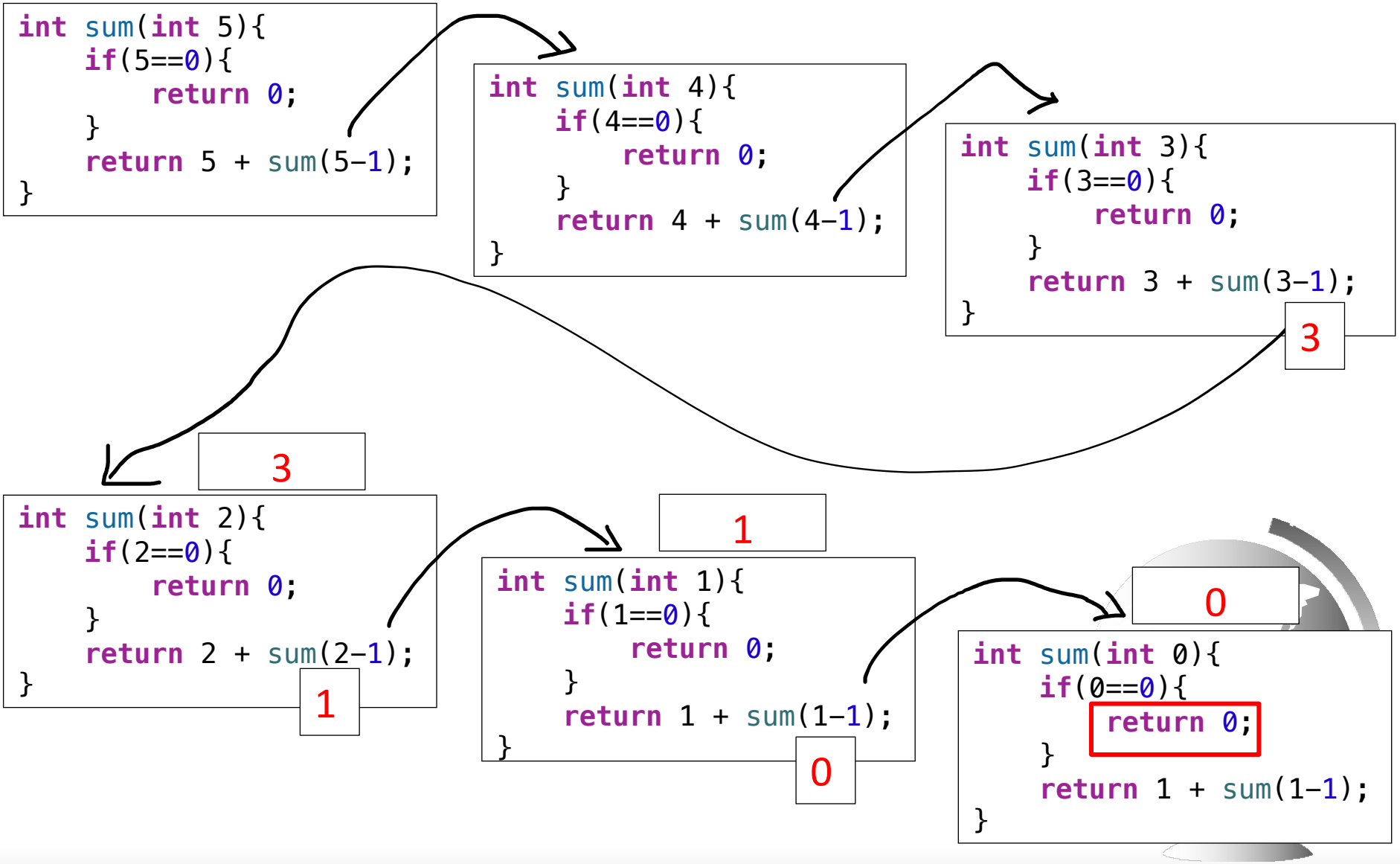
Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

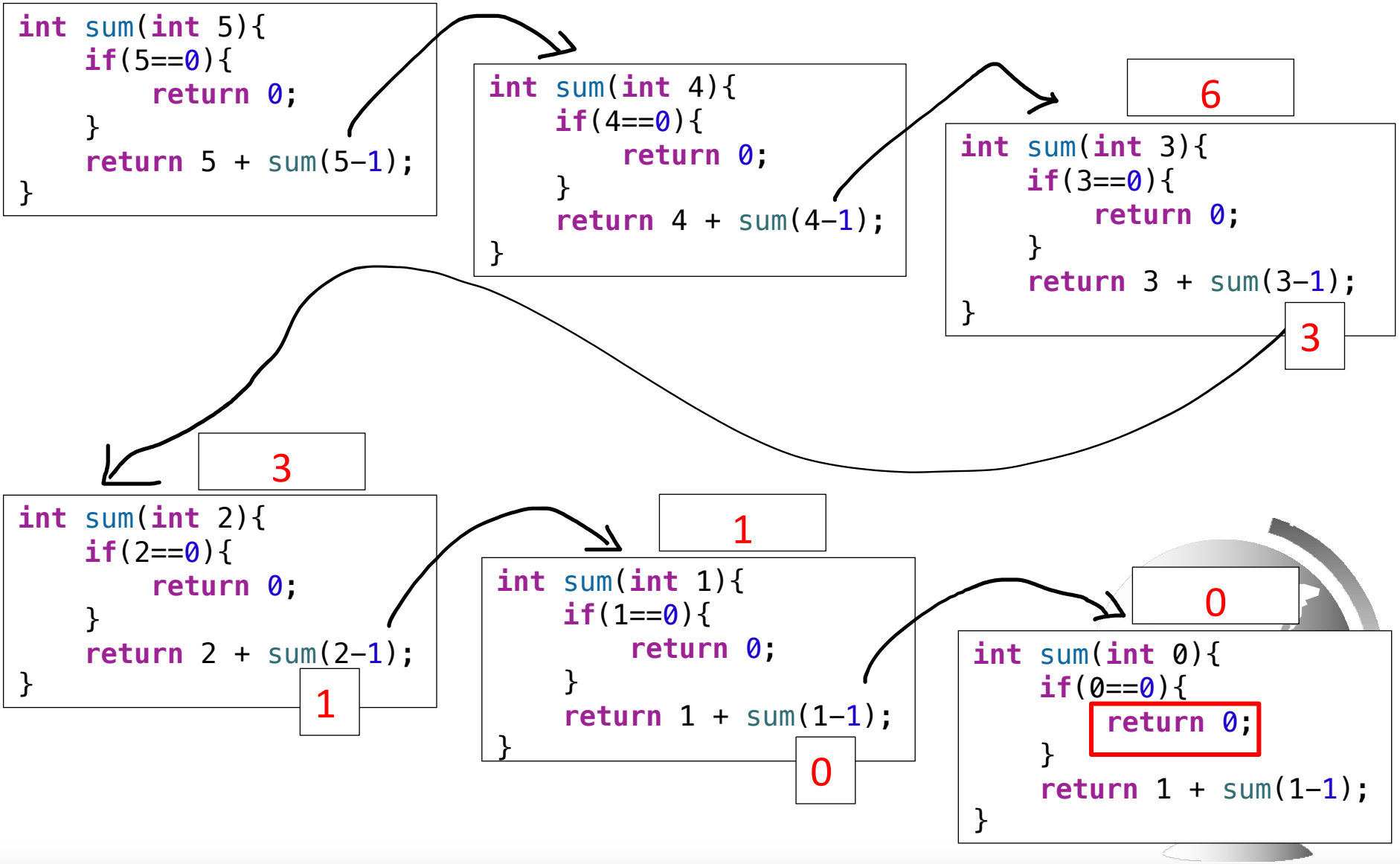
Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

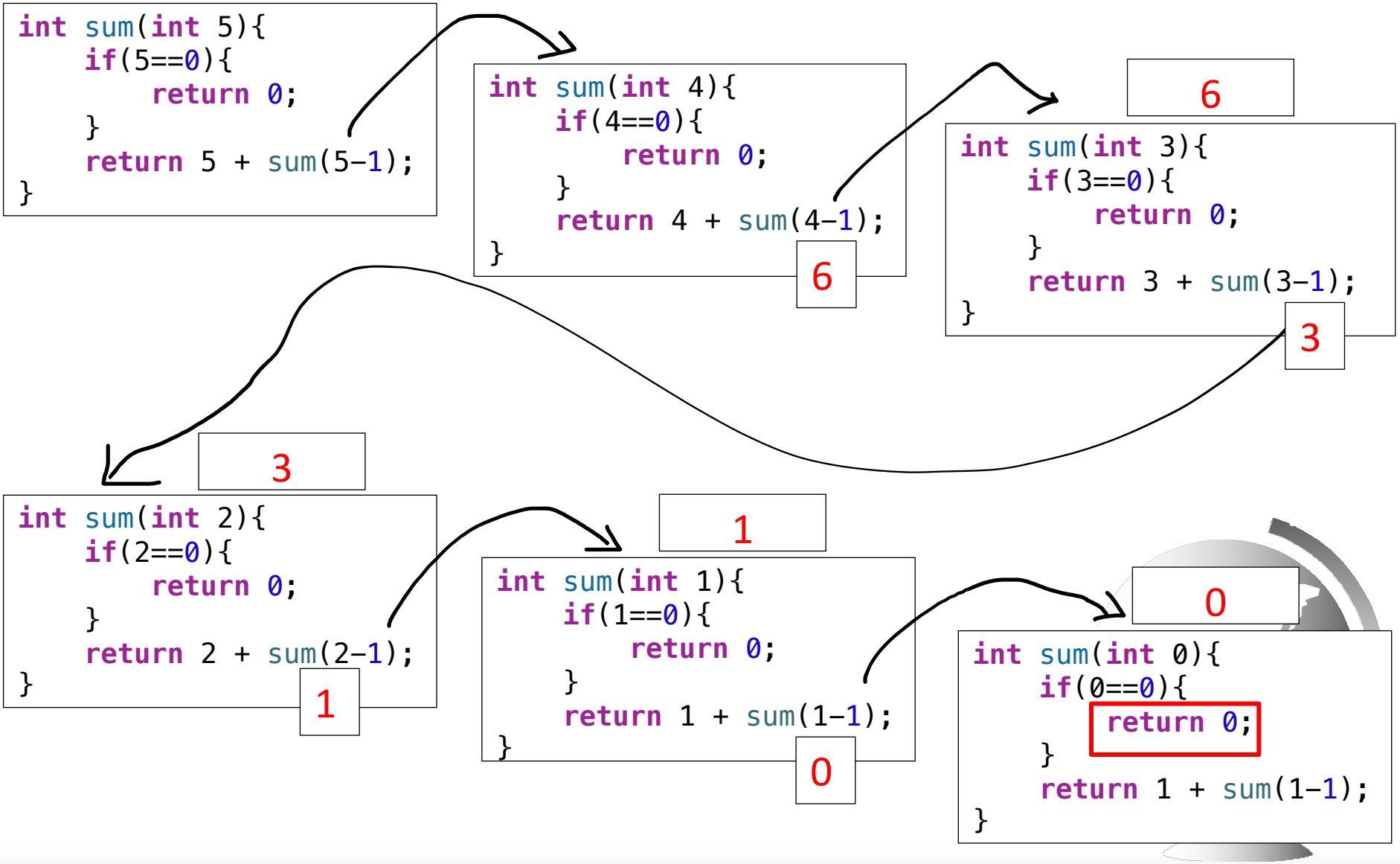
Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

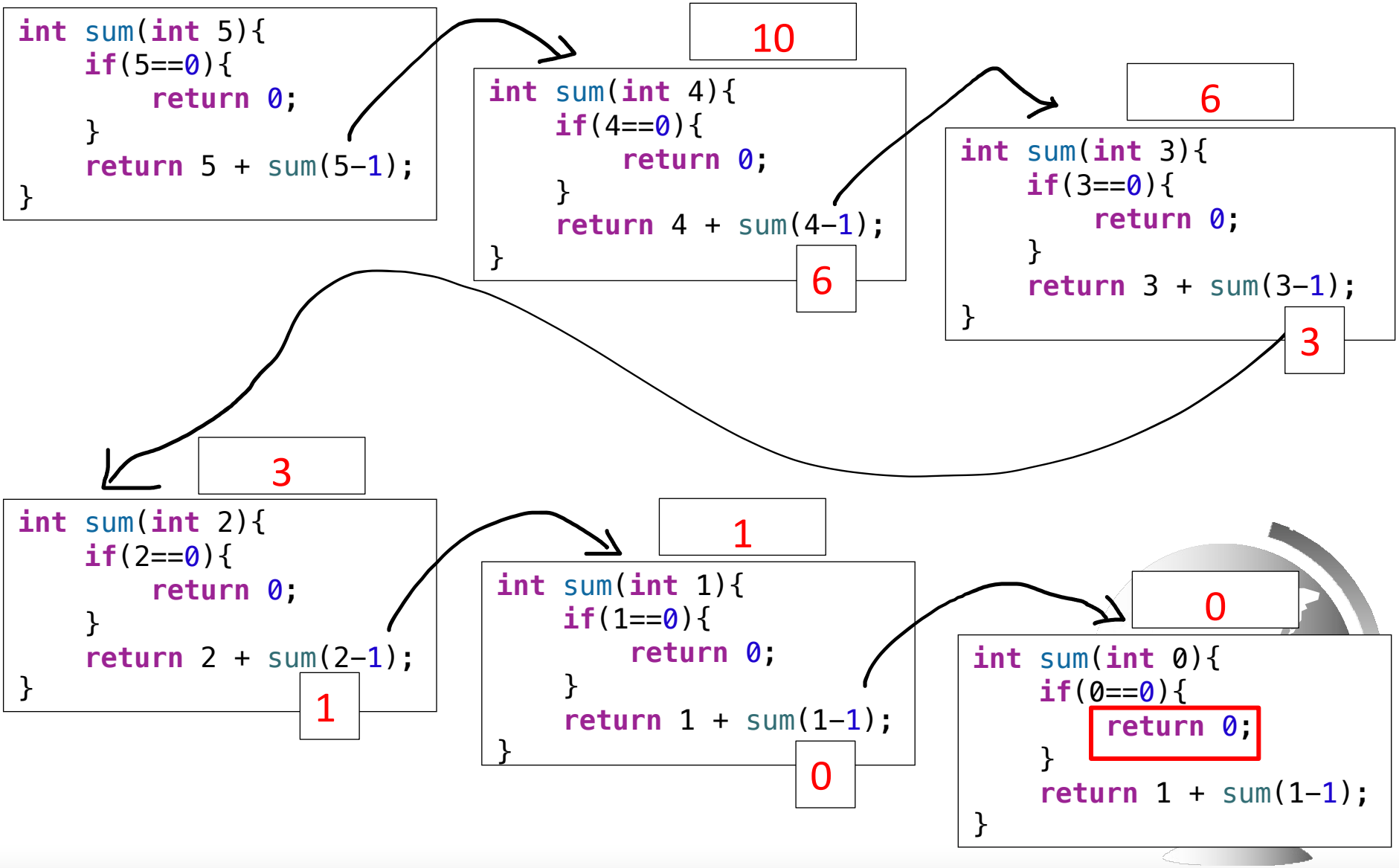
Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
6
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

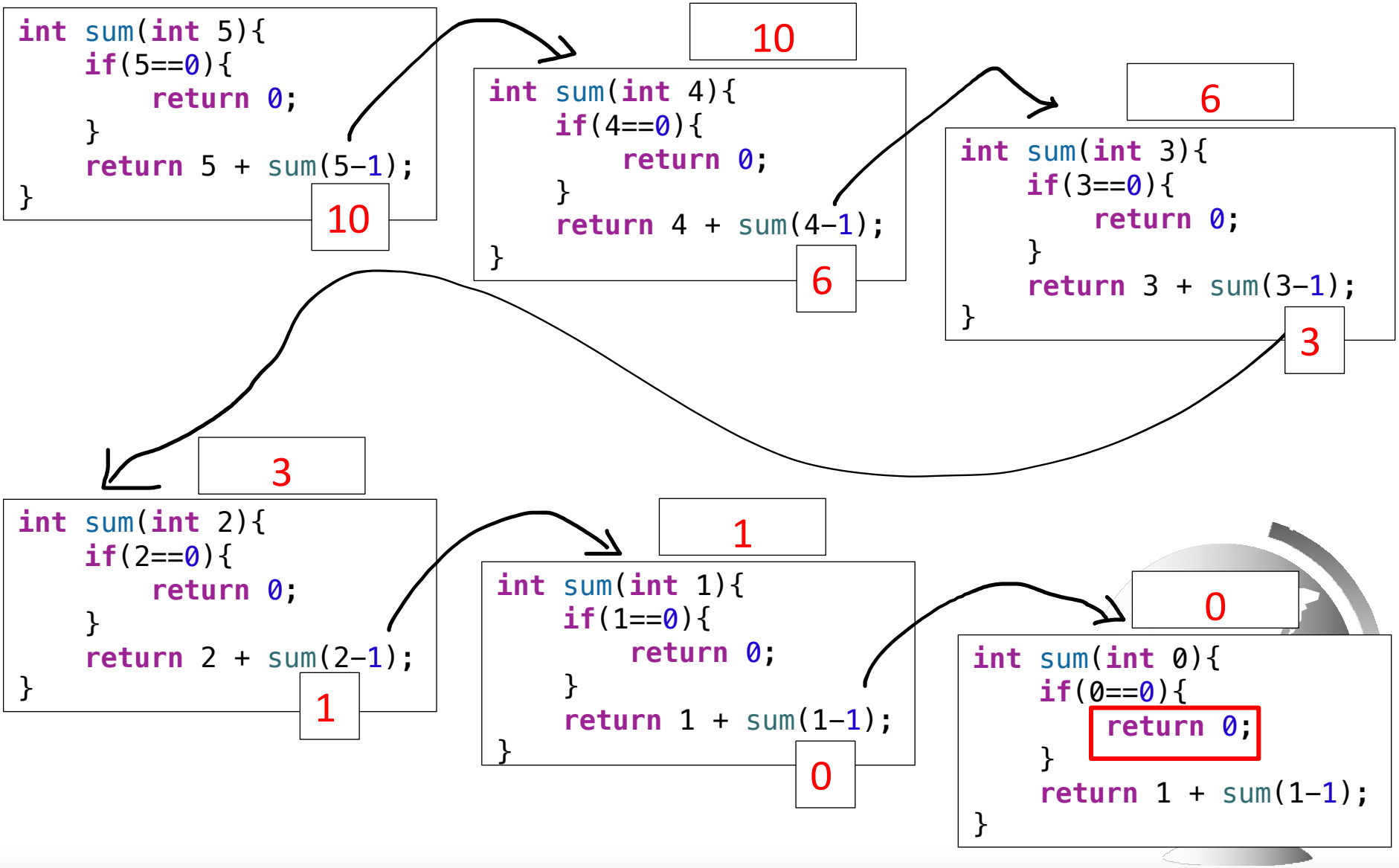
Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
6
6
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

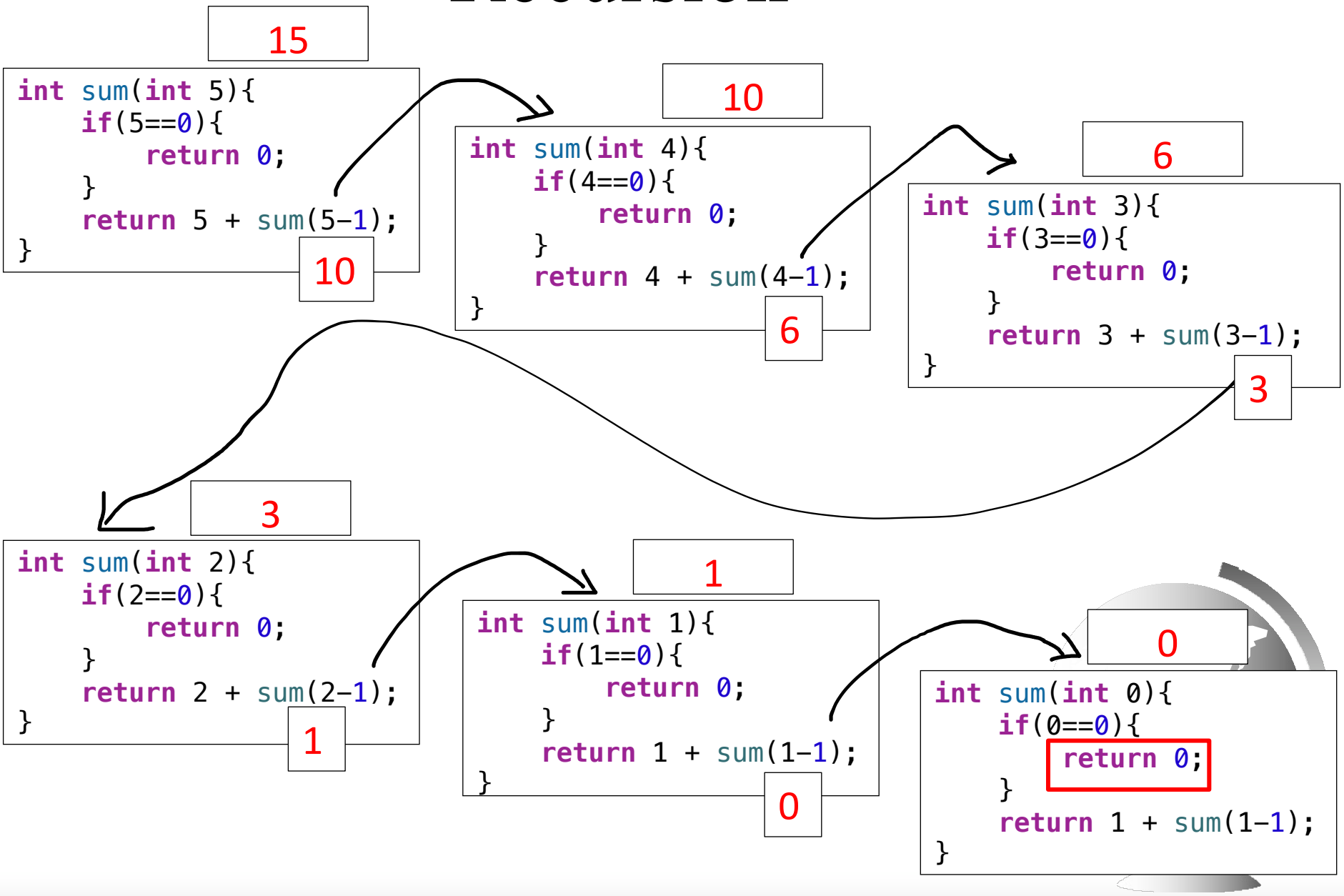
Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
10
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
6
6
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

Recursion


int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
10
10
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
6
6
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

Recursion


15
int sum(int 5){
if(5==0){
return 0;
}
return 5 + sum(5-1);
}
10
10
int sum(int 4){
if(4==0){
return 0;
}
return 4 + sum(4-1);
}
6
6
int sum(int 3){
if(3==0){
return 0;
}
return 3 + sum(3-1);
}
3
3
int sum(int 2){
if(2==0){
return 0;
}
return 2 + sum(2-1);
}
1
1
int sum(int 1){
if(1==0){
return 0;
}
return 1 + sum(1-1);
}
0
0
int sum(int 0){
if(0==0){
return 0;
}
return 1 + sum(1-1);
}

Factorial

```
//recursive method
public int factorial(int n) {
 if (n == 0) // Base case
   return 1;
 else
   return n * factorial(n - 1); // Recursive call
}
```

//recursive definition

n! = n * (n-1)!

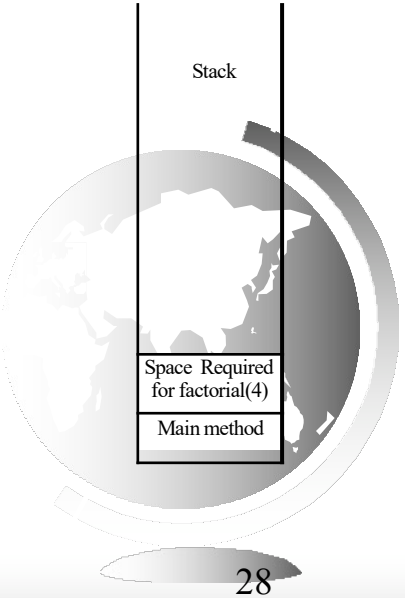
0! = 1

animation

Trace Recursive factorial

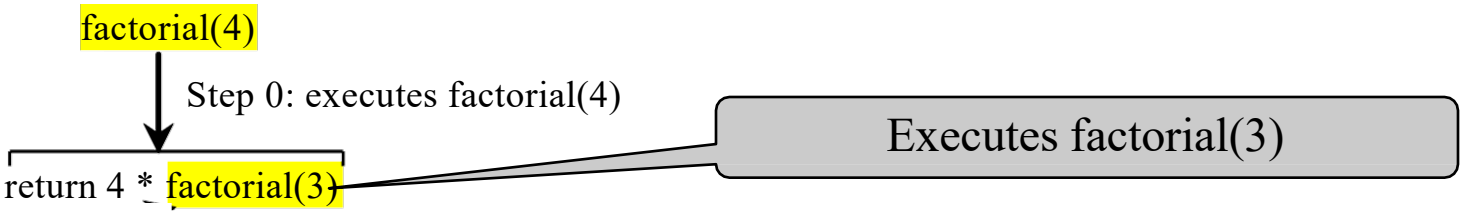
Executes factorial(4)

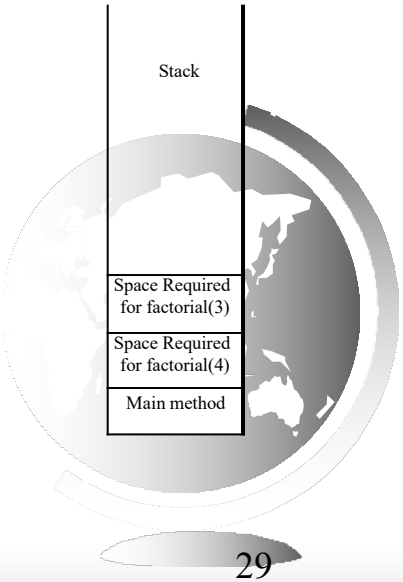
factorial(4)



animation

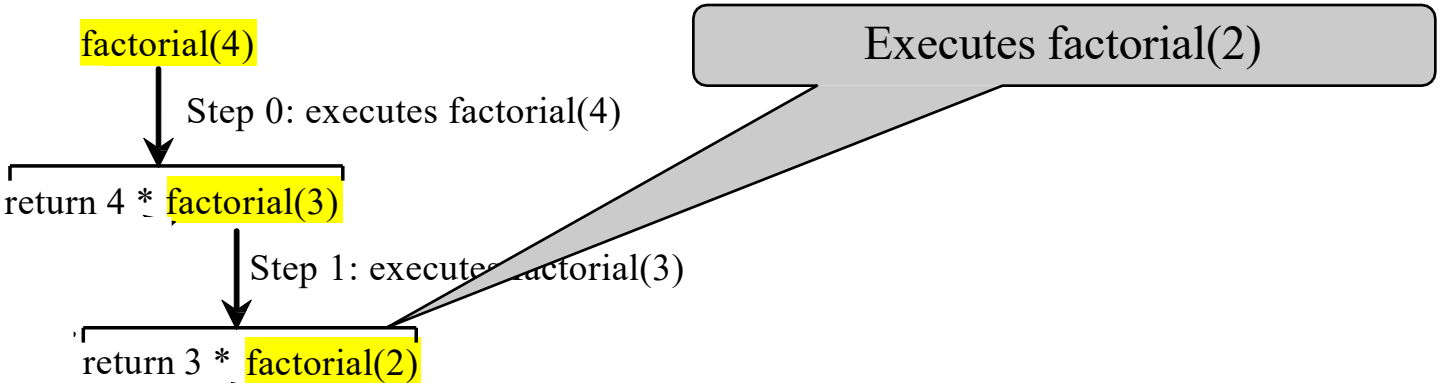
Trace Recursive factorial

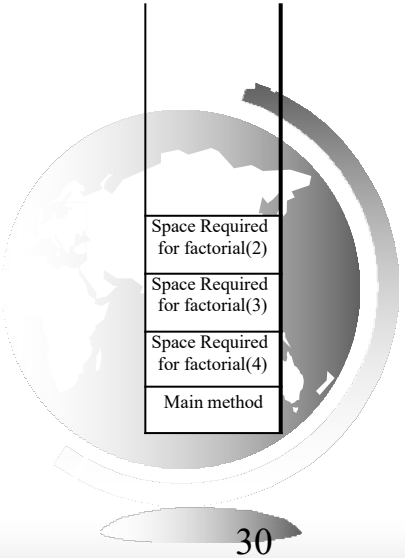




animation

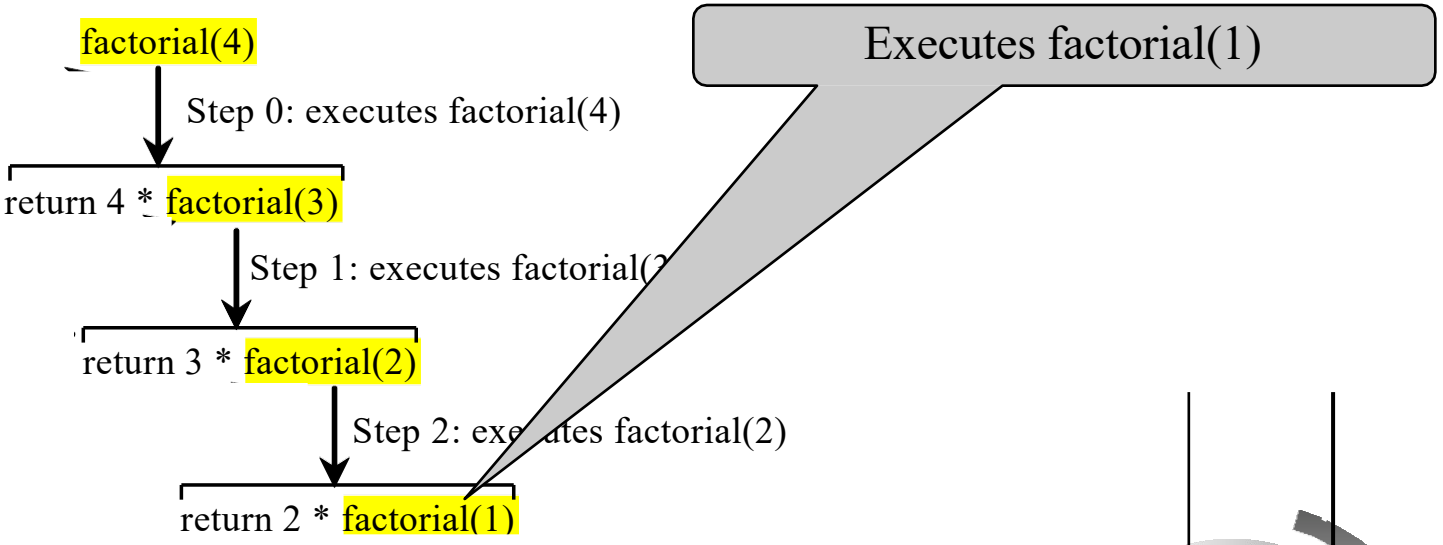
Trace Recursive factorial

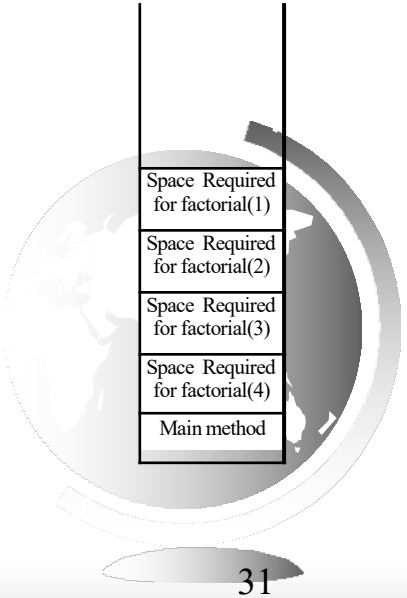




animation

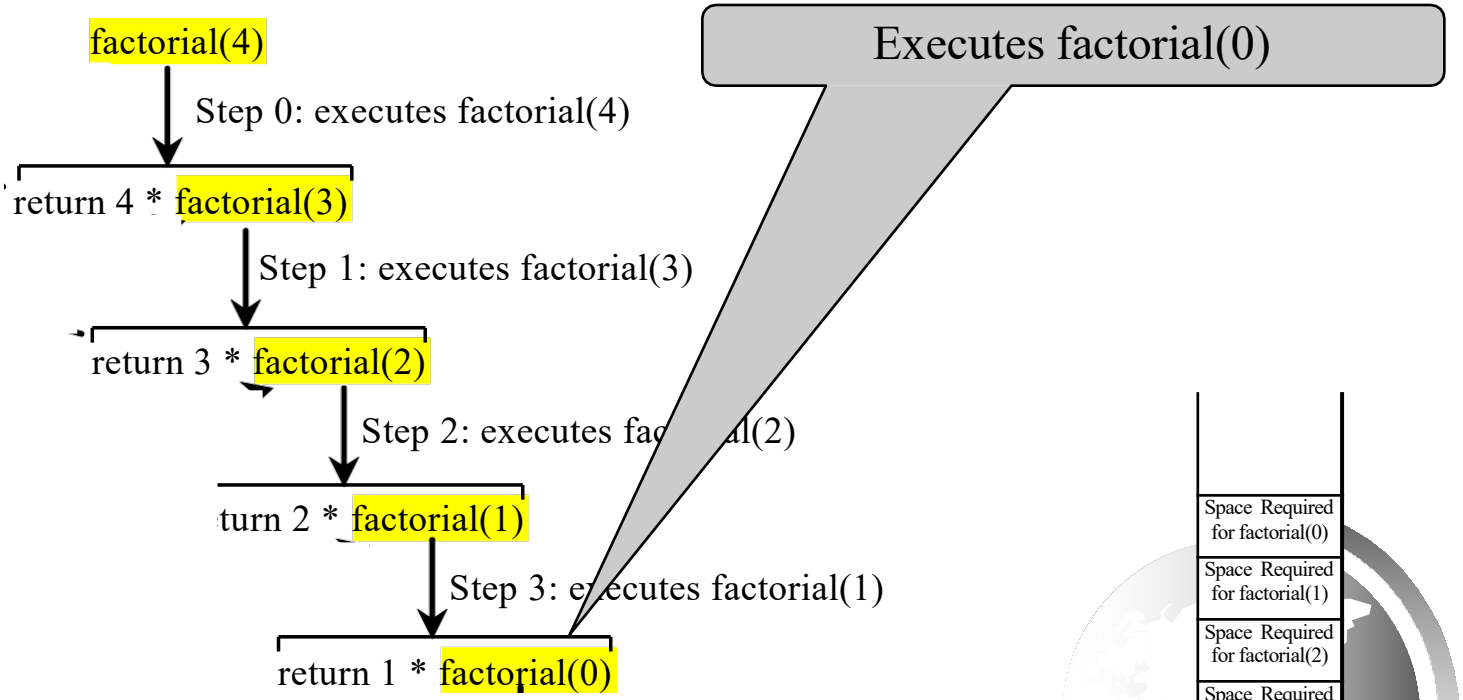
Trace Recursive factorial





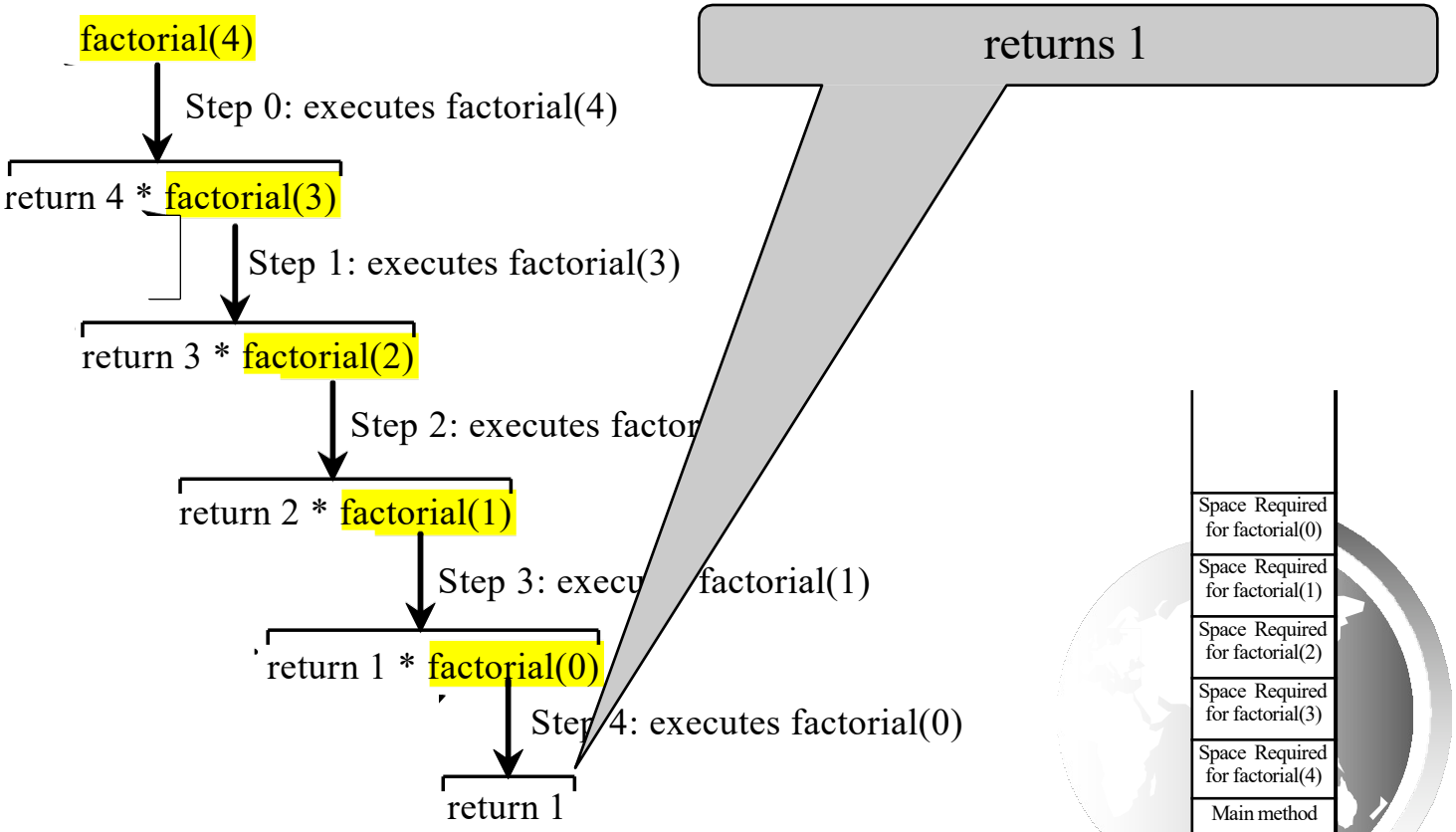
animation

Trace Recursive factorial



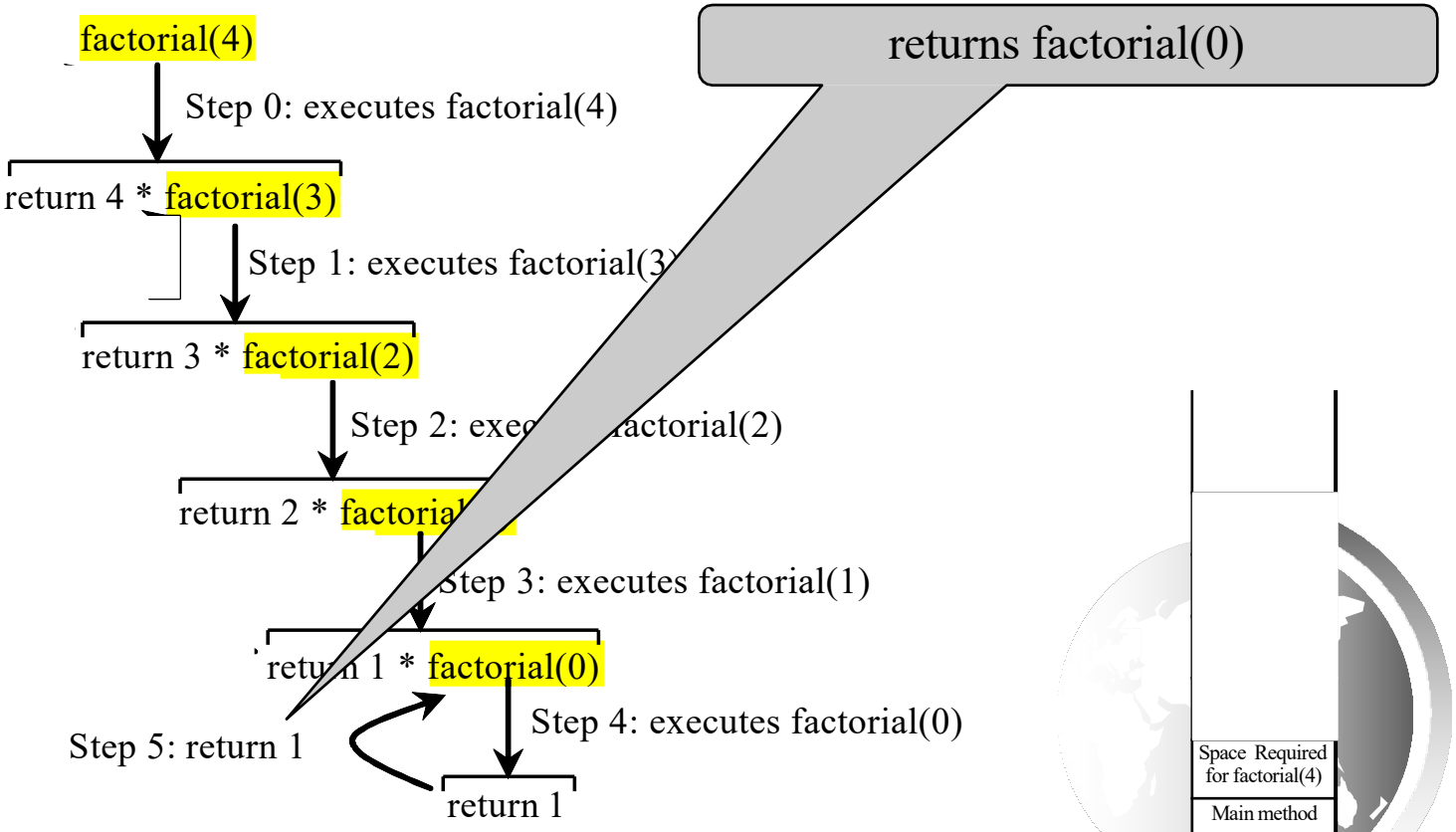
animation

Trace Recursive factorial



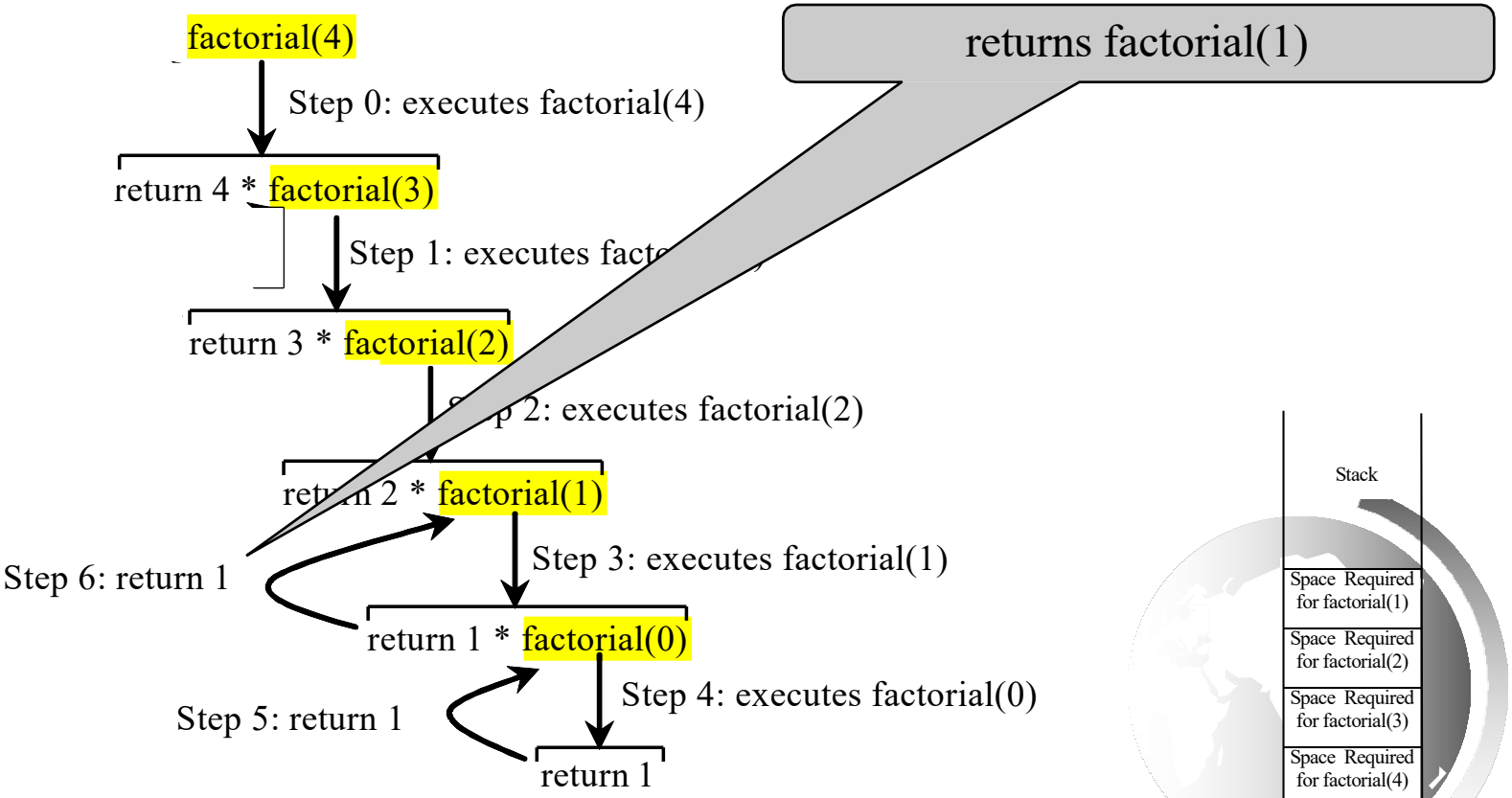
animation

Trace Recursive factorial



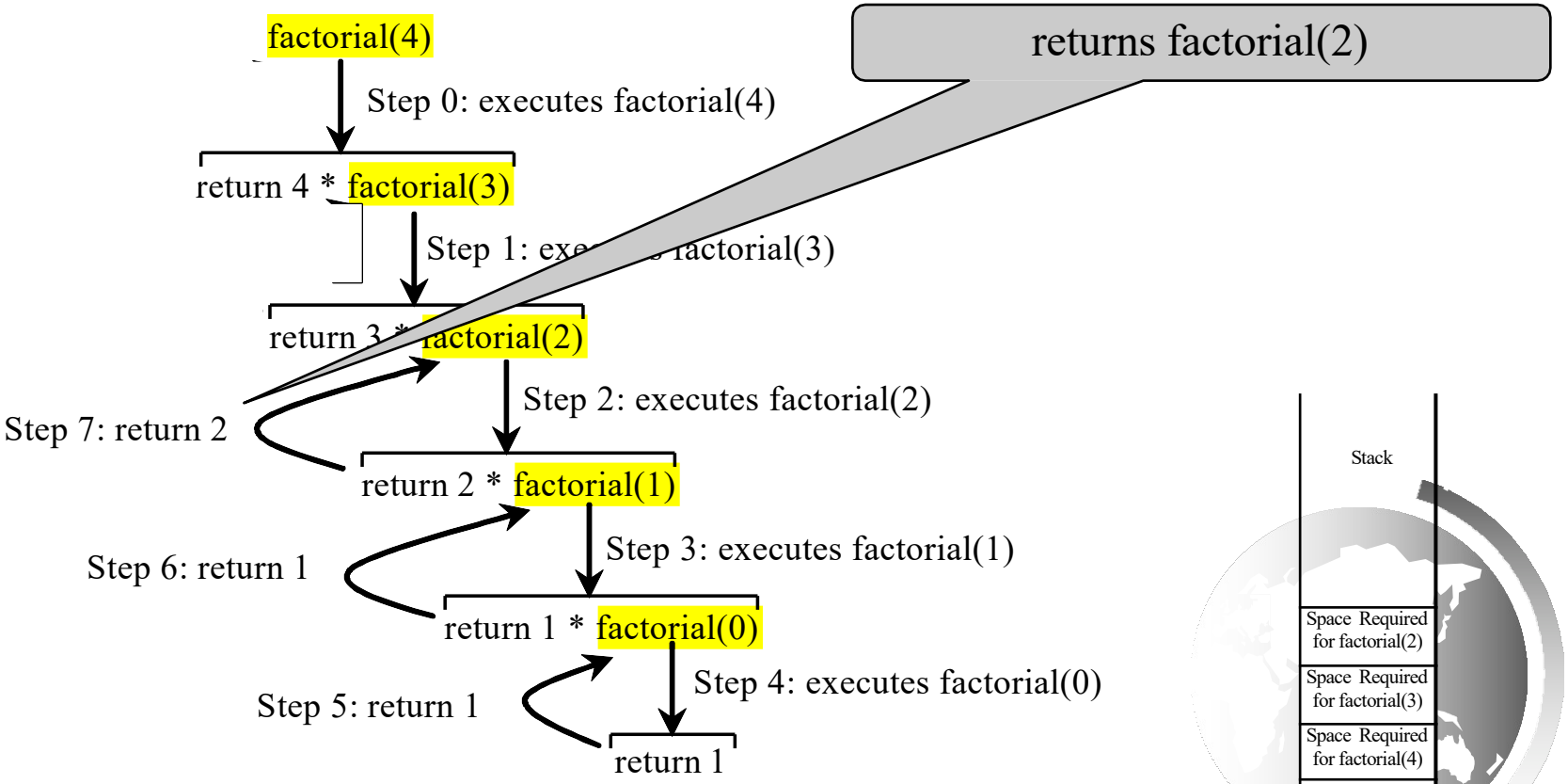
animation

Trace Recursive factorial



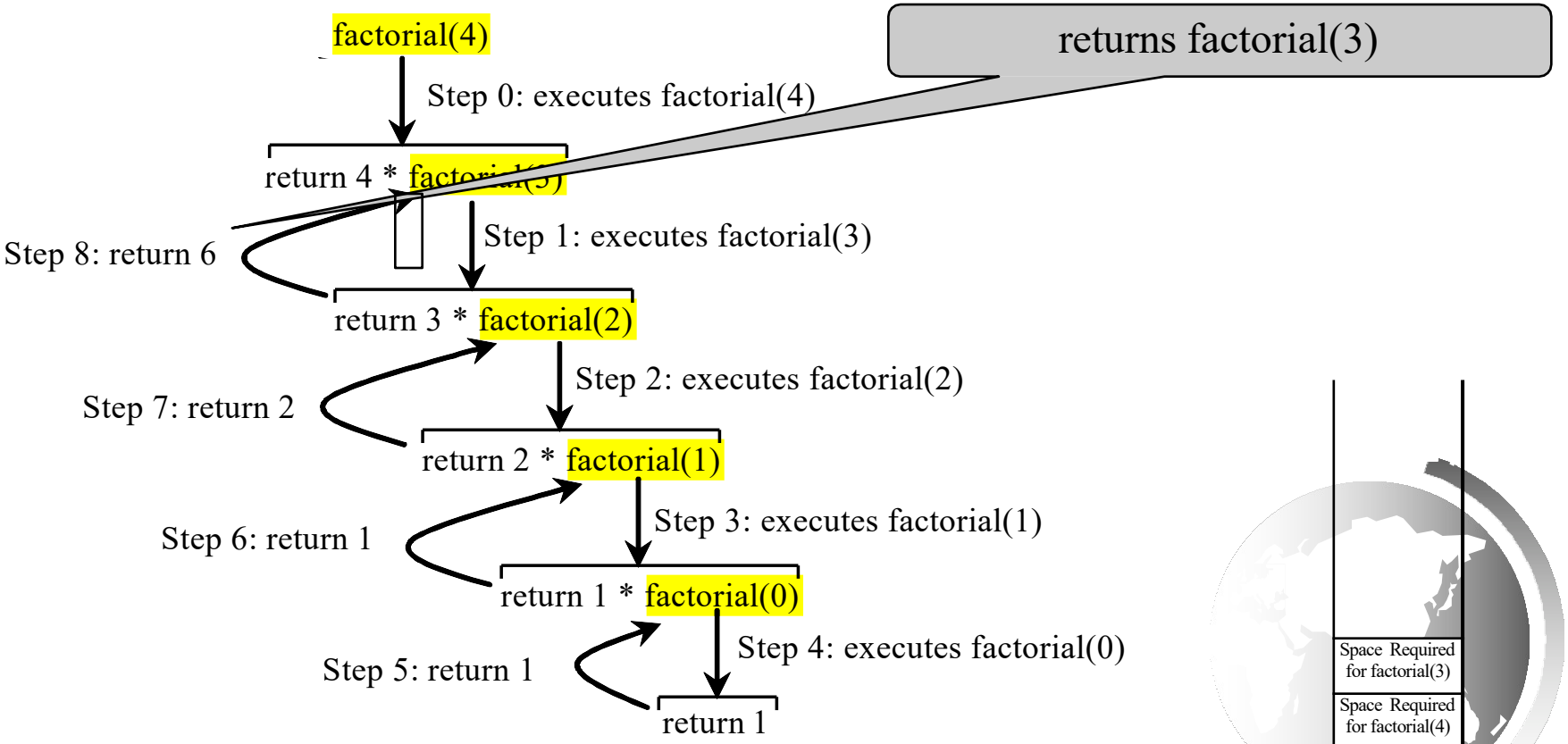
animation

Trace Recursive factorial



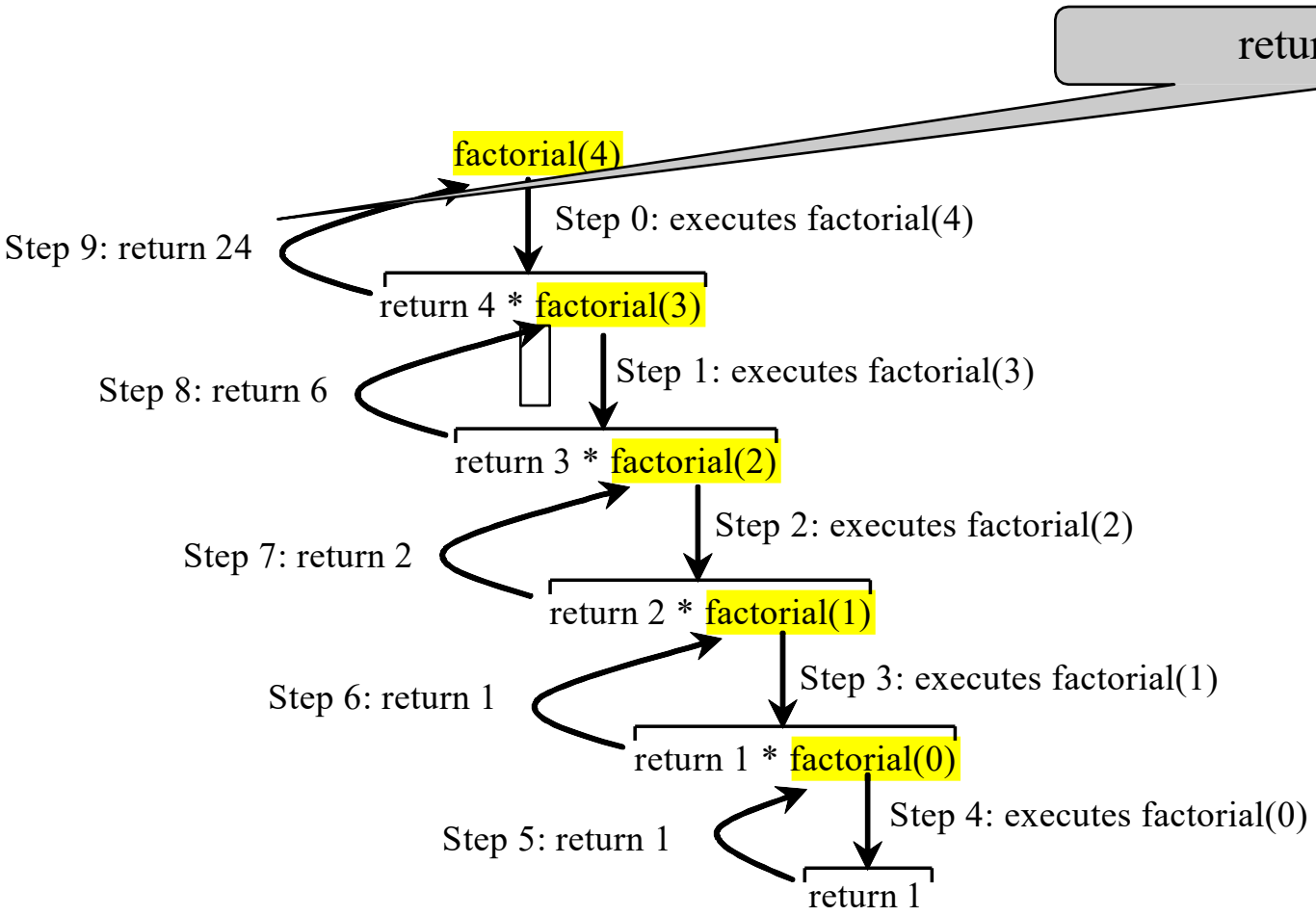
animation

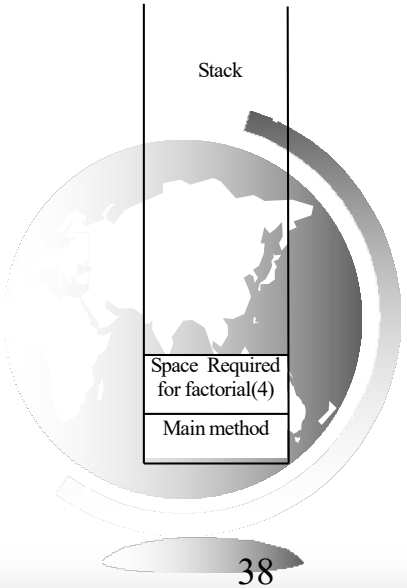
Trace Recursive factorial



animation

Trace Recursive factorial





Other Recursive definitions

f(0) = 0;

f(n) = n + f(n-1);

g(0)=1;

g(n)=g(n-1)+2

h(0)=1;

h(n)=3*h(n-1);

Characteristics of Recursion

All recursive methods have the following characteristics:

- One or more non-recursive **base cases** are used to stop the recursion.
- Recursive calls that reduce the original problem, bringing it increasingly closer to a base case until it becomes a base case.

To solve a problem using recursion, you break it into smaller subproblems, similar to the original problem.

```
DoSomething(list){
    Do(head); DoSomething(head.next);
}
```

Reaching the base case

- You must convince yourself that the non-recursive base case is eventually reached. What about:

```
public void doIt(int n){
 if(n != 0){
   bla;
   doIt(n-2);
 }
}
```

Fibonacci's Rabbits

- Suppose a newly-born pair of rabbits, one male, one female, are put on an island.
  - A pair of rabbits doesn't breed until 2 months old.
  - Thereafter each pair produces another pair each month
  - Rabbits never die.
- How many pairs will there be after n months?

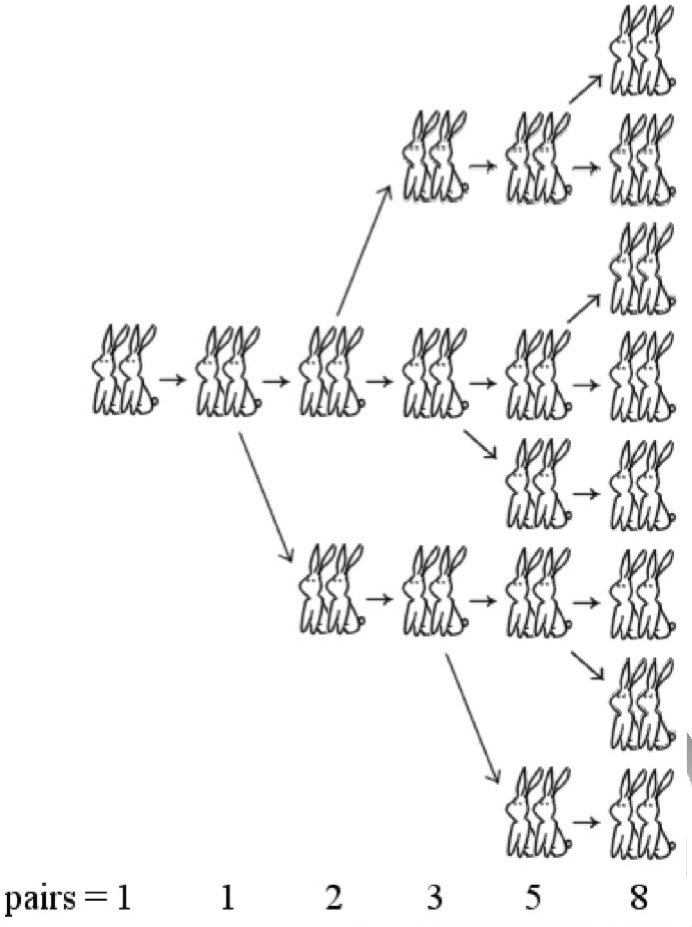


image from: http://www.jimloy.com/algebra/fibo.htm

Fibonacci Numbers

```
Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55 89…

         indices: 0 1 2 3 4 5 6 7  8  9  10 11
```

fib(0) = 0;

fib(1) = 1;

fib(index) = fib(index -1) + fib(index -2); index >=2

fib(3) = fib(2) + fib(1)

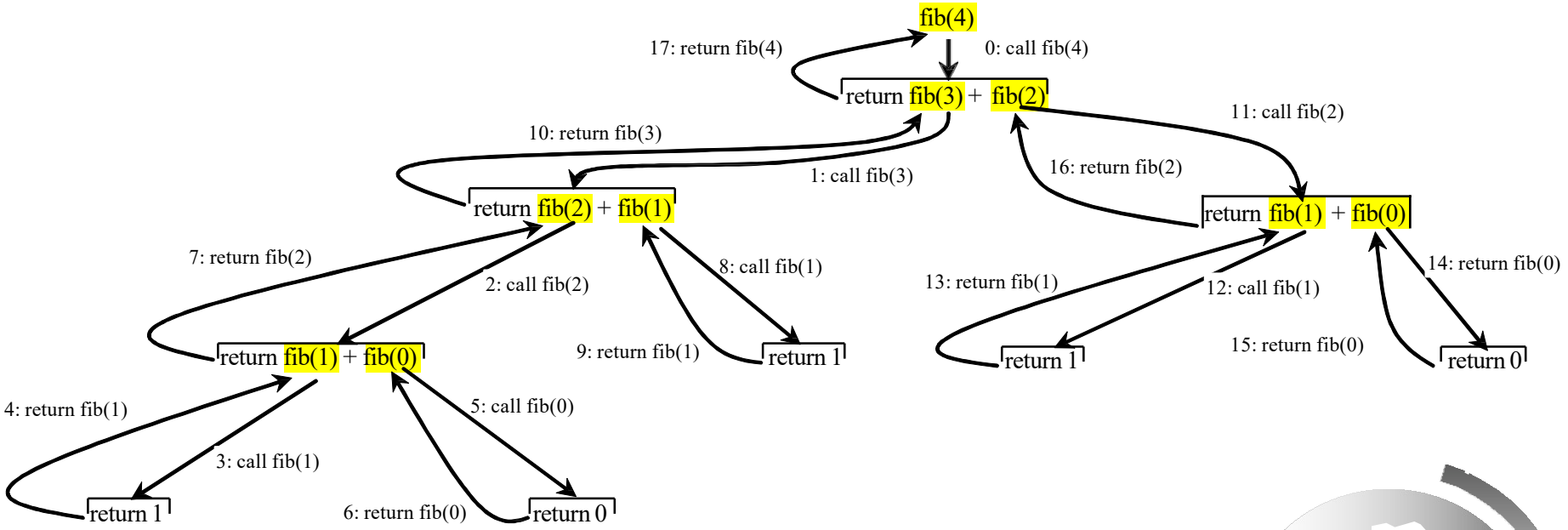
= (fib(1) + fib(0)) + fib(1)

= (1 + 0) + 1 + 1 = 2

ComputeFibonacci

Run

Fibonnaci Numbers, cont.



Characteristics of Recursion

All recursive methods have the following characteristics:

- One or more base cases (the simplest case) are used to stop recursion.
- Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

Break a problem into subproblems.
If a subproblem is the same as the original problem, but with a smaller size, solve the subproblem recursively

Exercise

- Let's write a method `reverseLines(Scanner scan)` that reads lines using the scanner and prints them in reverse order.
  - Use recursion without using loops.
  - Example input:

```
this
is
fun
no?
```


Expected output:

```
no?
fun
is
this
```

  - What are the cases to consider?
    - How can we solve a small part of the problem at a time?
    - What is a file that is very easy to reverse?

Reversal pseudocode

- Reversing the lines of a file:
  - Read a line L from the file.
  - Print the rest of the lines in reverse order.
  - Print the line L.

- If only we had a way to reverse the rest of the lines of the file....

Reversal solution

```
public void reverseLines(Scanner input) {
    if (input.hasNextLine()) {
        // recursive case
        String line = input.nextLine();
        reverseLines(input);
        System.out.println(line);
    }
}
```

– Where is the base case?

Memoization

- Problems like Fibonacci and parkingLot create "bushy" trees.
- These trees are full of repeated calls
- We can achieve tremendous speedup by saving intermediate results.

Look back at the Fibonacci call tree:

fib(n) calls fib(n-1) and fib(n-2)

fib(n) calls fib(n-2) and fib(n-3)

So fib(n) calls fib(n-2) twice (1 direct, 1 indirect)

Fast Fib

```
private long[] memo = new long[100];
public long fastFib(int n){
    if(n<2) return n;
    if (memo[n]==0)  // not computed yet
      // so compute and memoize it
      memo[n] = fastFib(n-1) + fastFib(n-2);
    return memo[n];
}
```

Exercise 18.3 GCD

gcd(2, 3) = 1

gcd(2, 10) = 2

gcd(25, 35) = 5

gcd(205, 301) = 5

gcd(m, n)

Approach 1: Brute-force, start from min(n, m) down to 1, to check if the number is common divisor for both m and n, if so, it is the greatest common divisor.

Approach 2: **`Euclid's`** algorithm

Approach 3: Recursive Euclid

Euclid's algorithm

E.g., gcd(287, 91)

- 287 = (287/91)*91 + 287%91= 91*3 + 14

  any divisor of 287 and 91 is a divisor of 14:

  287-91*3 = 14

  also

  any divisor of 91 and 14 must be a divisor of 287:

  287 = 91*3 + 14

- Hence gcd(287,91) = gcd(91,14)

Now compute gcd(287,91) using this method.

Euclid's algorithm

```
// Get absolute value of m and n;
t1 = Math.abs(m); t2 = Math.abs(n);
// r is the remainder of t1 divided by t2;
r = t1 % t2;
while (r != 0) {
  t1 = t2;
  t2 = r;
  r = t1 % t2;
}

// When r is 0, t2 is the greatest common
// divisor between t1 and t2
return t2;
```

Recursive Euclid

gcd(m, n) = n if m % n = 0;
gcd(m, n) = gcd(n, m % n); otherwise;

Exercise: write this as a java method.

Using Recursion

Recursion is good for solving problems that are inherently recursive, and not easily solved iteratively

- Spock, Parkinglot, Hanoi

This usually means: more than linear recursive

- Multiple recursive calls
- All the above have two recursive calls

Linear recursion can be easily replaced by iteration

- palindrome, reverse, factorial, binary search, gcd