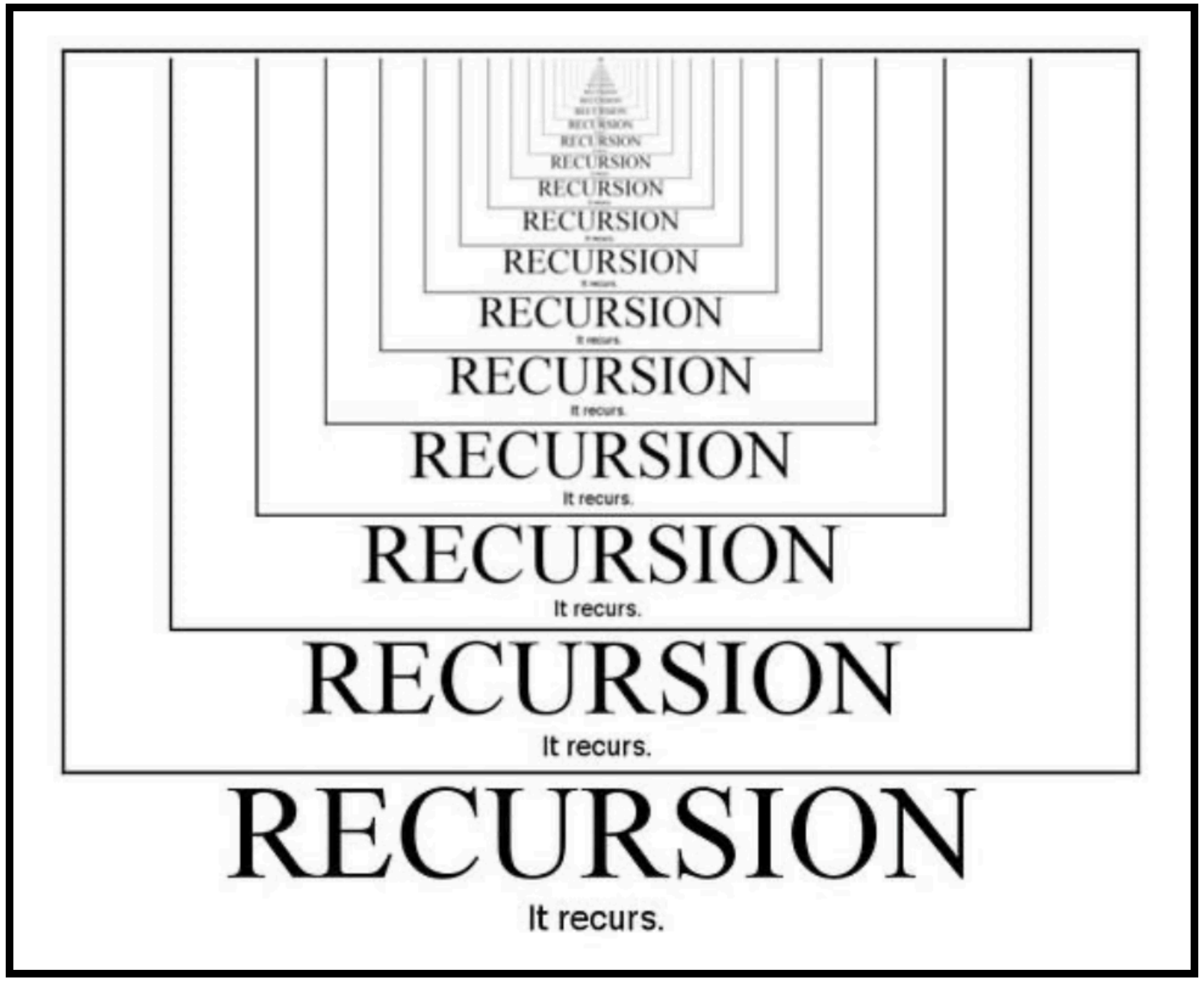
COSC 2436: Recursion

J. Eoin Donovan - 2023

What is recursion?

Recursion is a technique where a function calls itself to solve a problem.



How does recursion work?

Recursion consists of two parts which make it work.

- **Base Case - this is the simplest version of the problem that can be solved directly**
- **Recursive Step - this breaks down the problem into a smaller version of itself (getting one step closer to reaching the base case)**

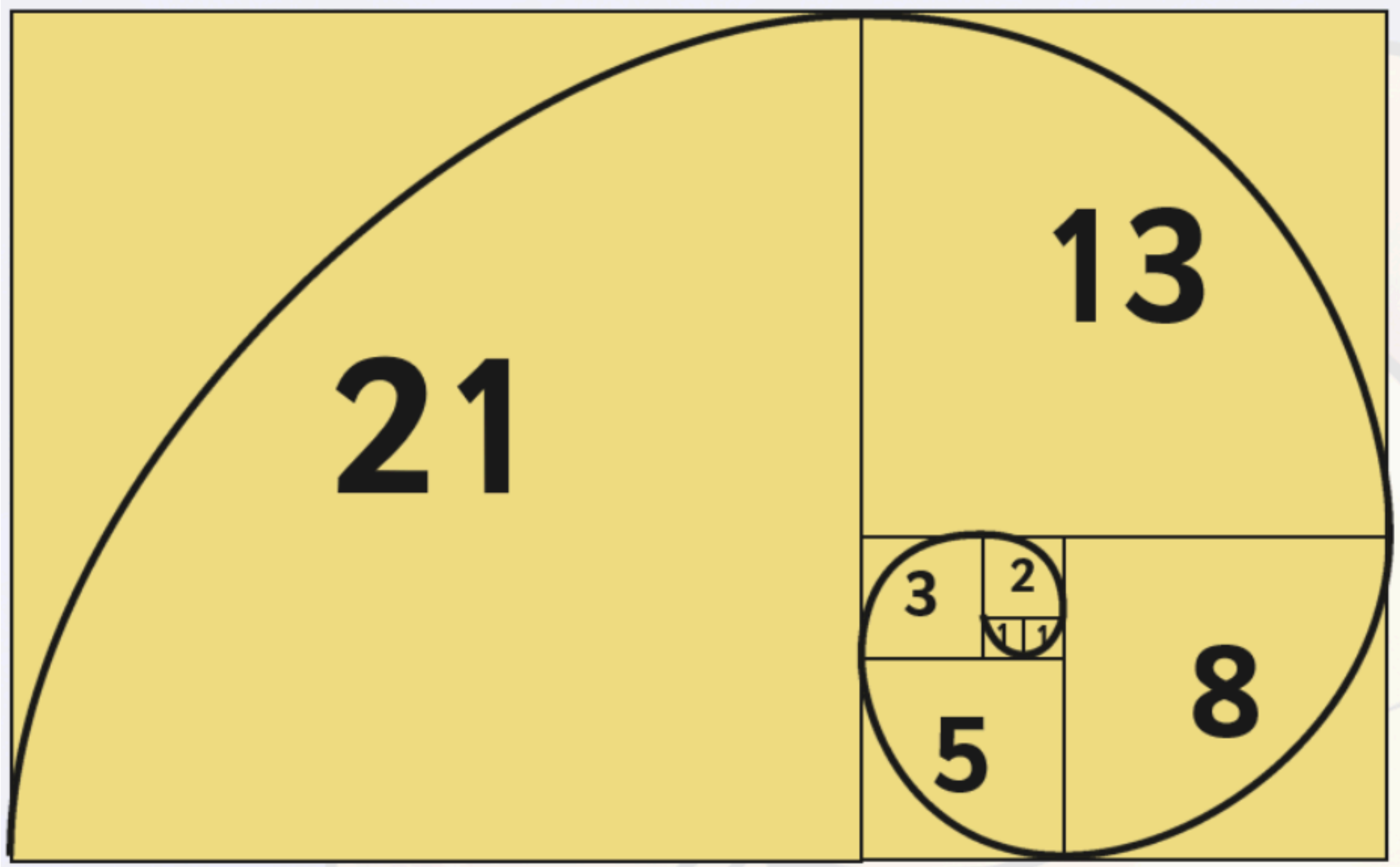
Recursion Example - Russian Dolls

How many dolls do we have?

- **Base Case - when we reach a doll that can't be opened**
- **Recursive Step - opening the doll to unveil the next doll.**

Recursion Advantages

- Simplicity - solution is often intuitive and concise
- Divide & Conquer - break down problem into smaller parts
- Applicability - can be well-suited for some problems

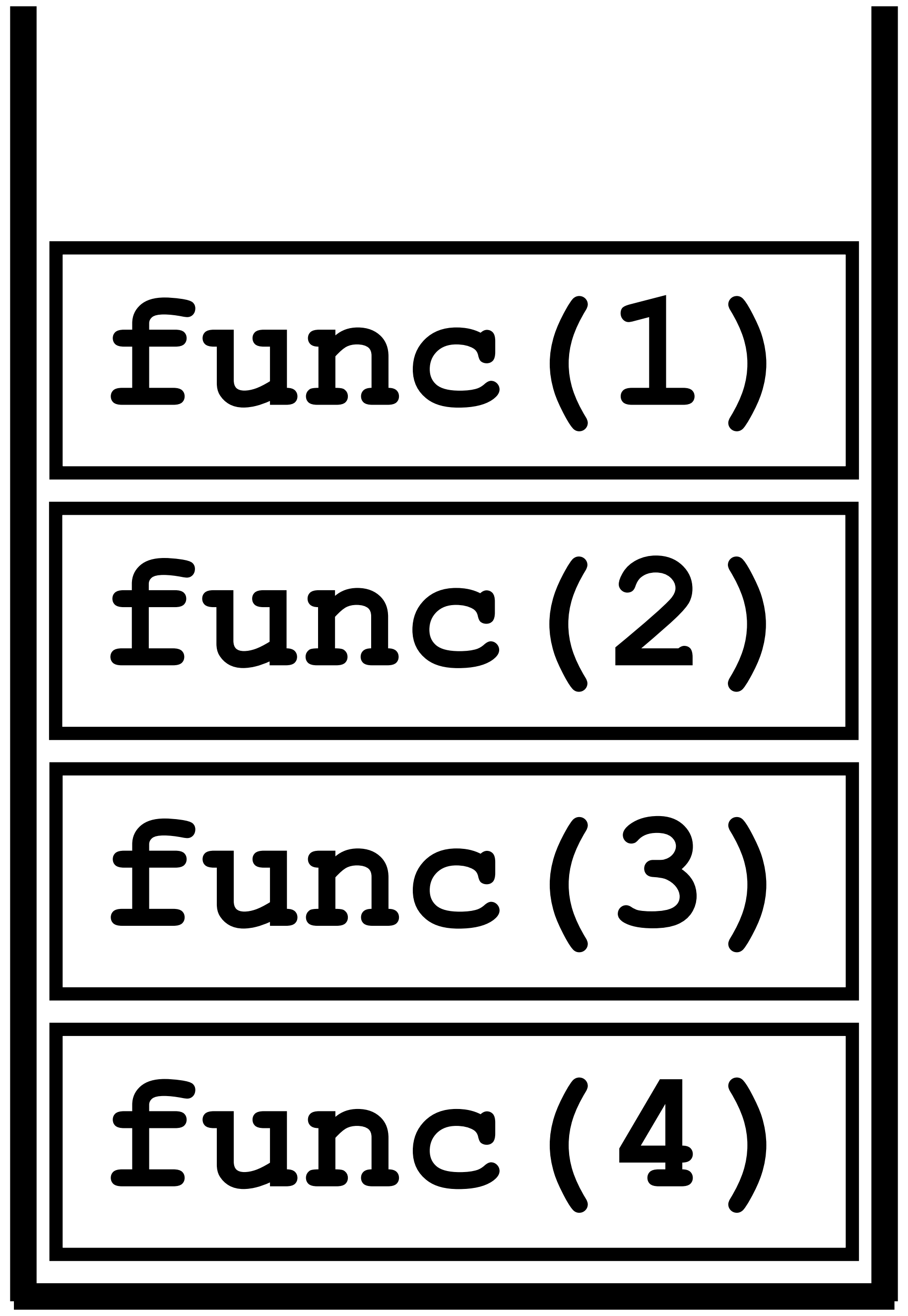


Recursion Disadvantages

- Memory Overhead - uses additional memory on the call stack
- Performance - sometimes less efficient
- Complexity - harder to understand for more advance recursive solutions

The Call Stack

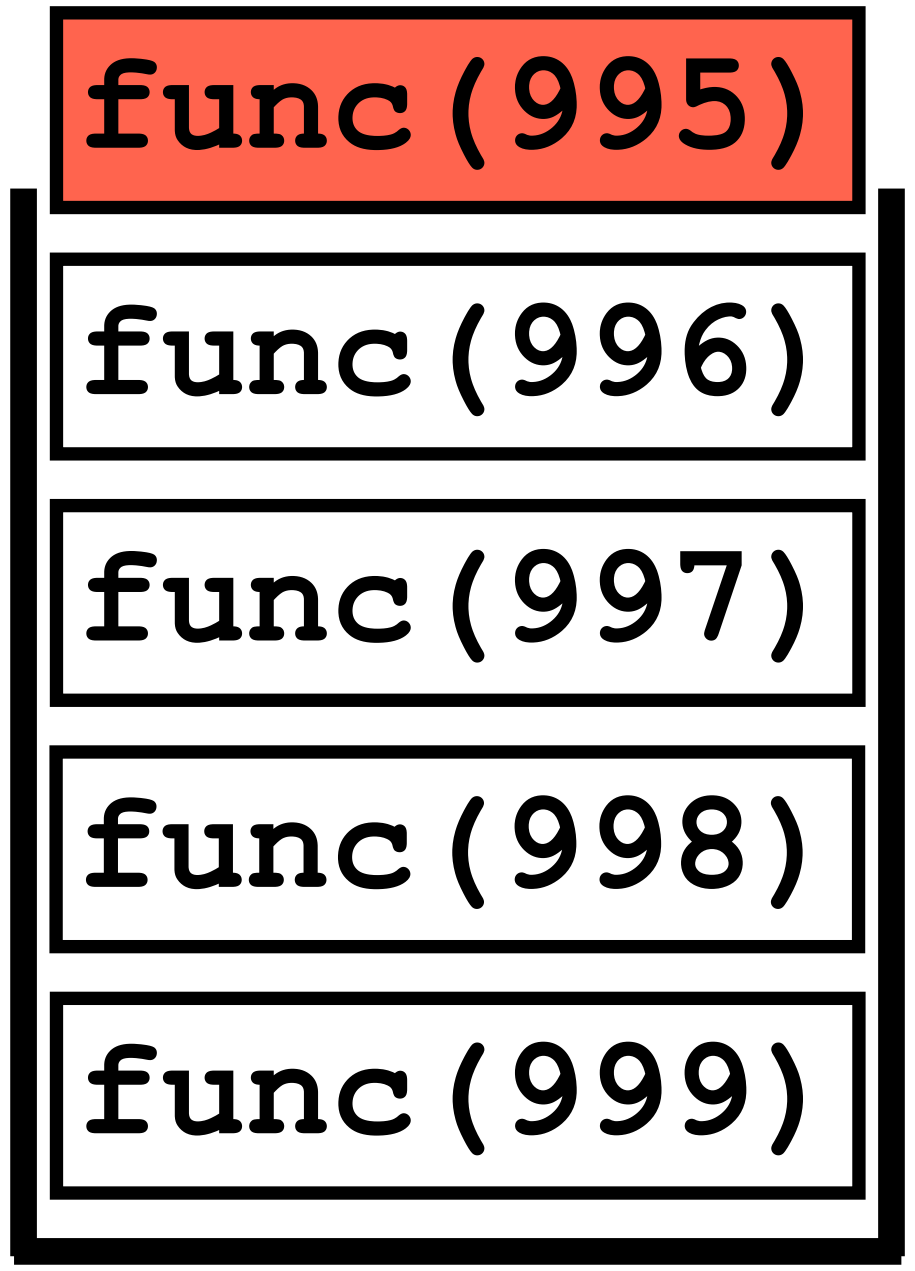
The call stack is a region of memory that tracks active function calls and their data in a program.



Recursion & the Call Stack

Recursion employs the call stack to create and store function instances and then pop off functions after the base case is reached.

Stack overflow happens when too many recursive calls are made to the call stack.



Recursion - Practice

Write a function to find the factorial for a given number *x* using recursion.

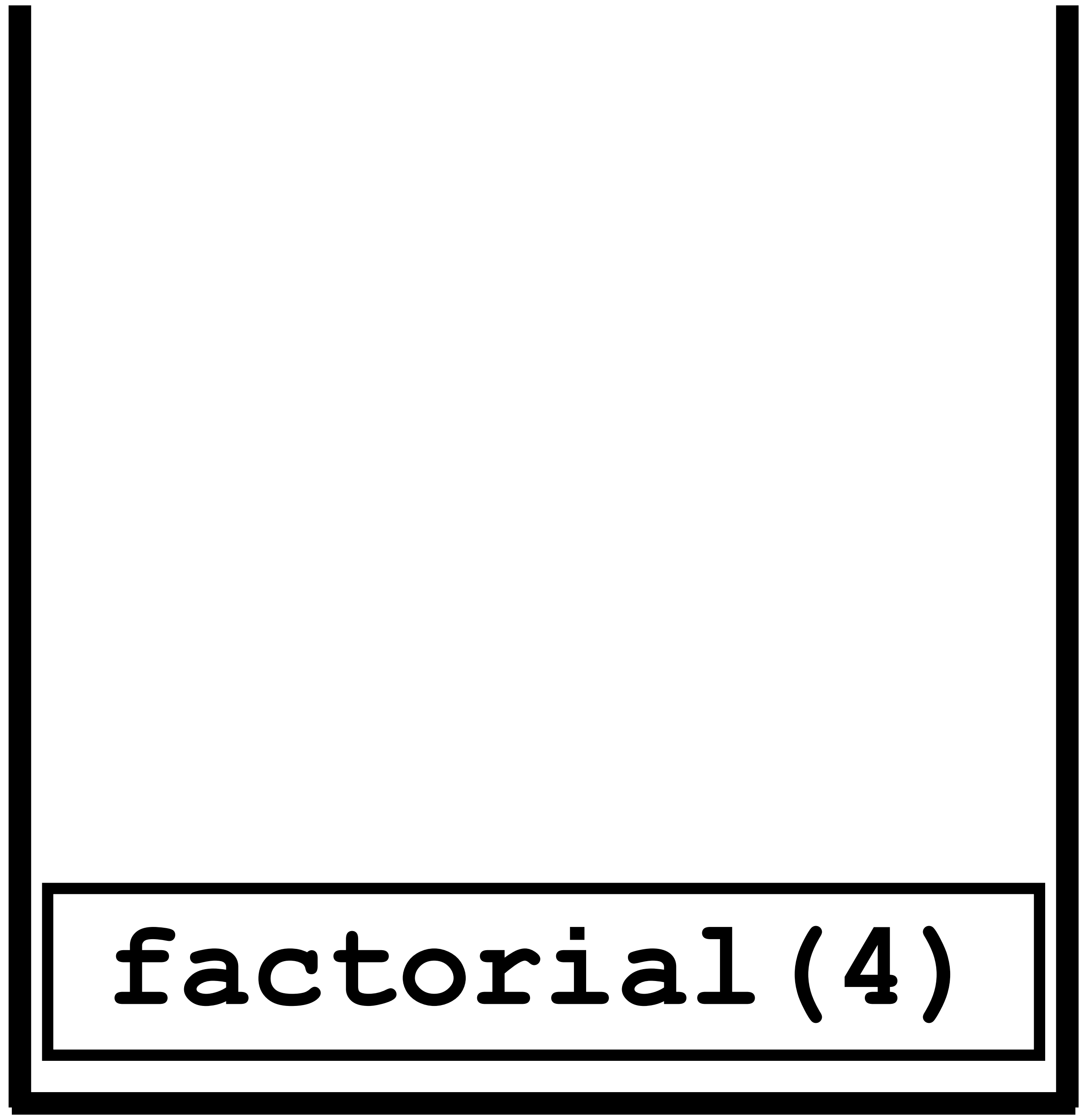
```
factorial(4) => 4 * 3 * 2 * 1 => 24
```

```
int factorial(int x){



}
```

Recursion - Practice

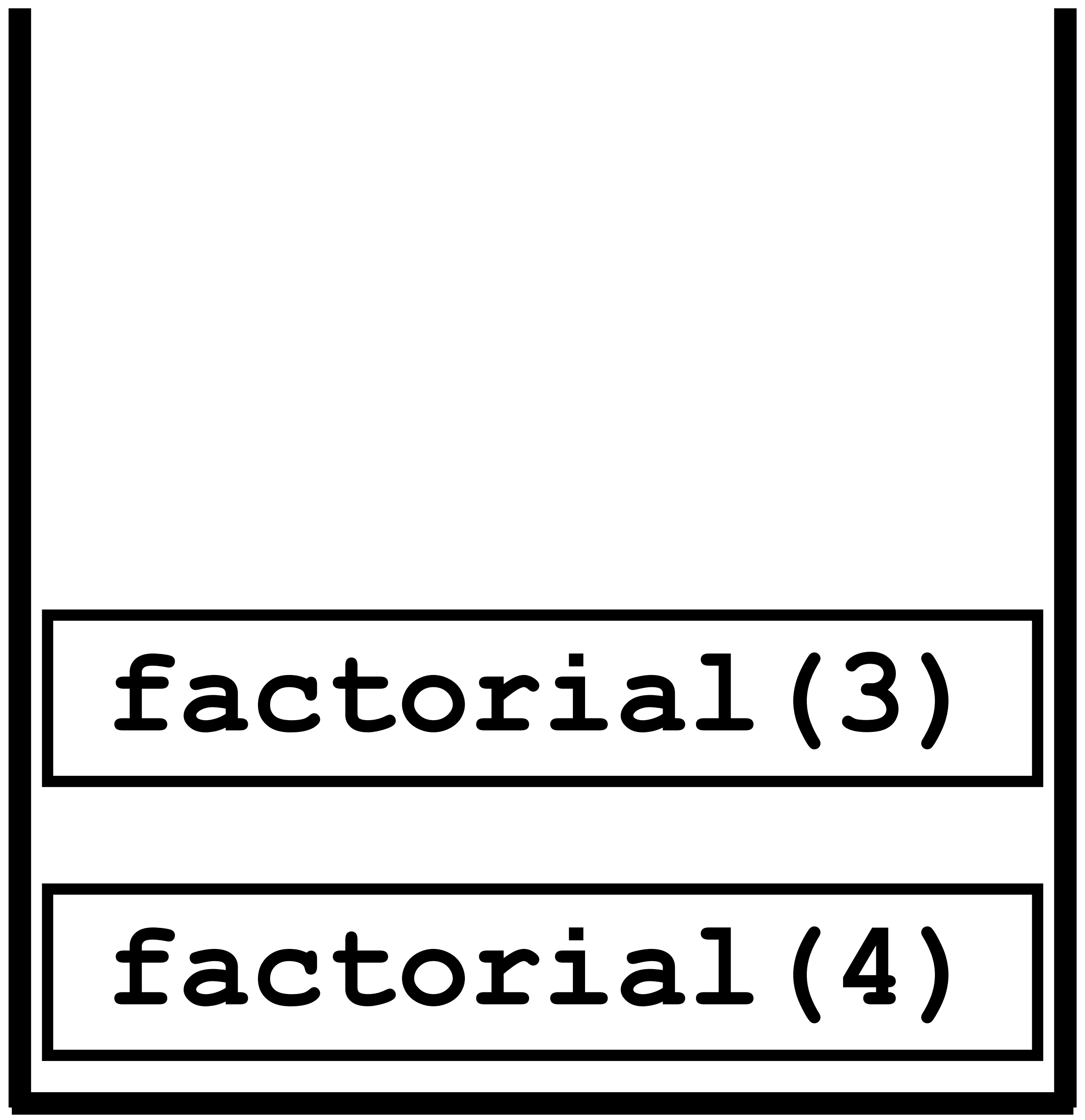
```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```

```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```



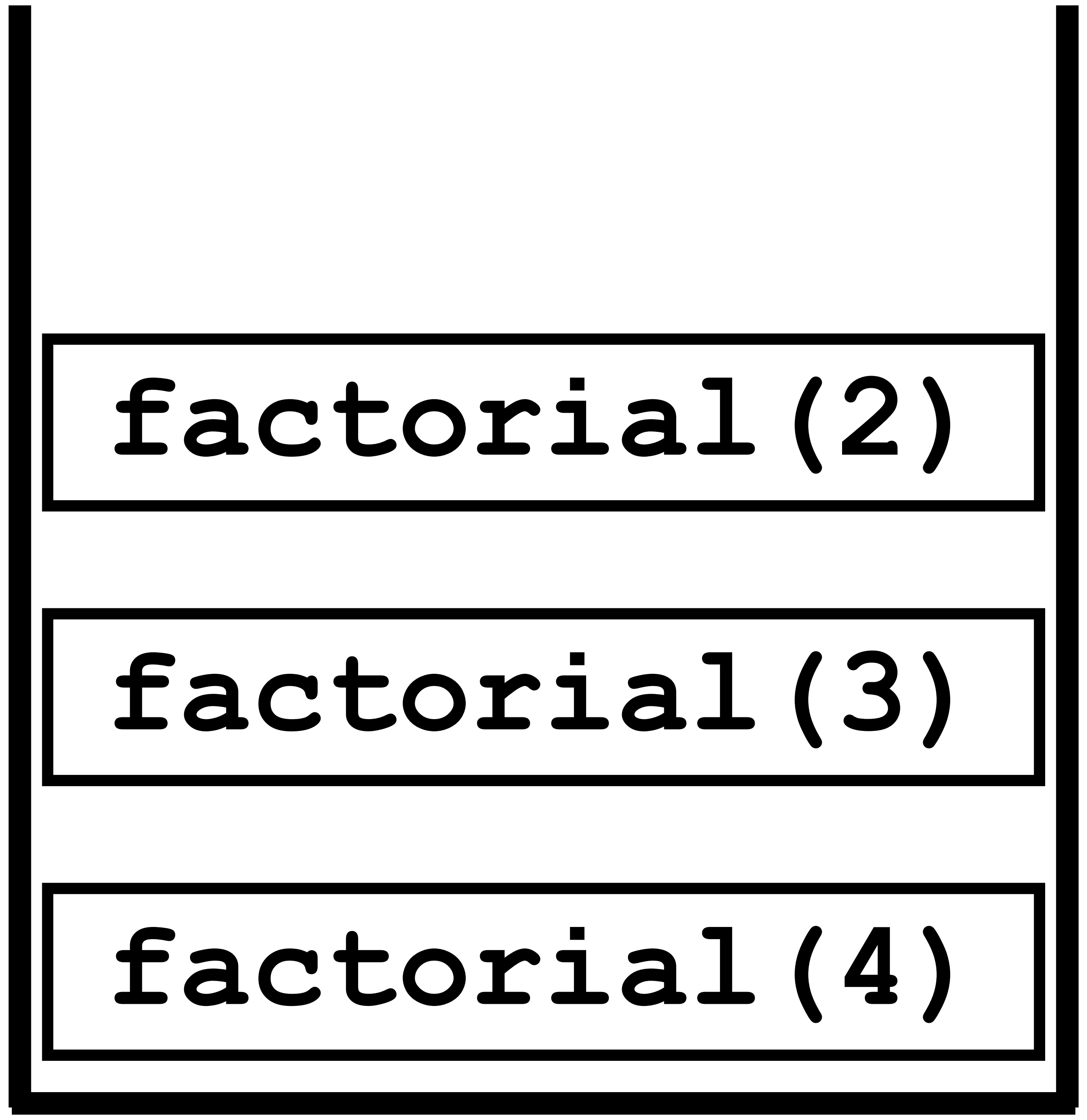

```
return 4 * factorial(3)
```

```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```



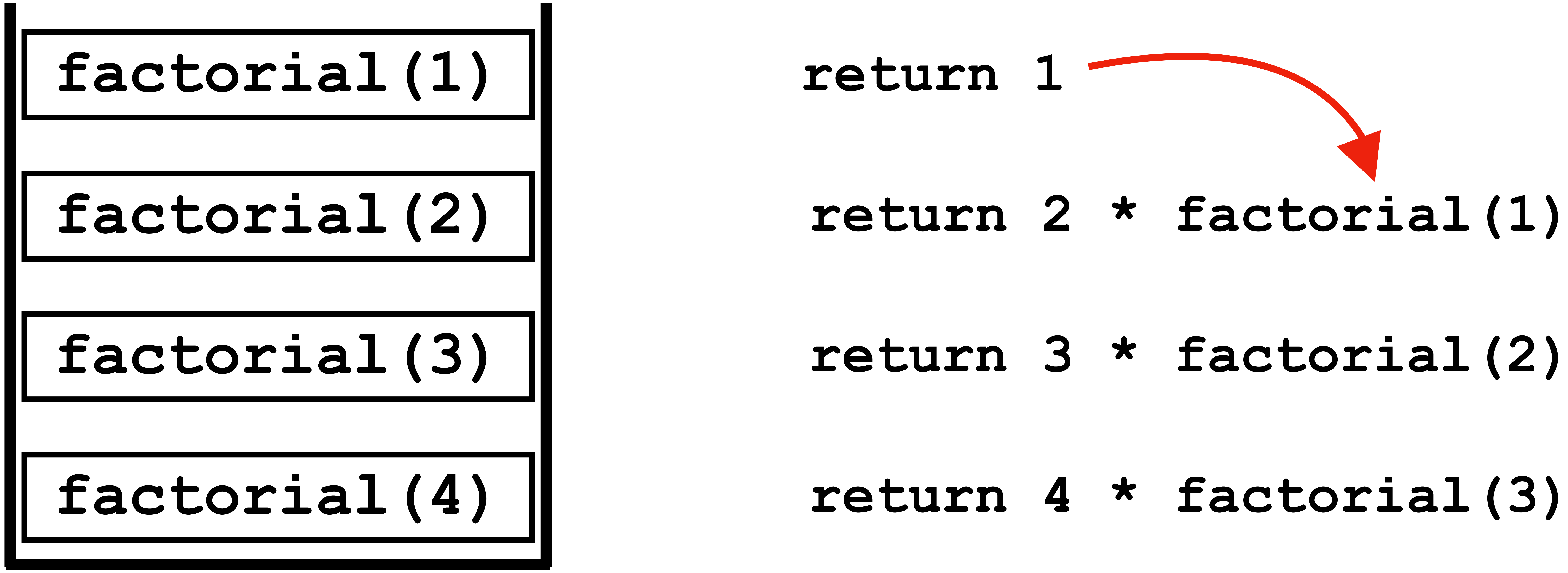

```
return 3 * factorial(2)

return 4 * factorial(3)
```

```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```

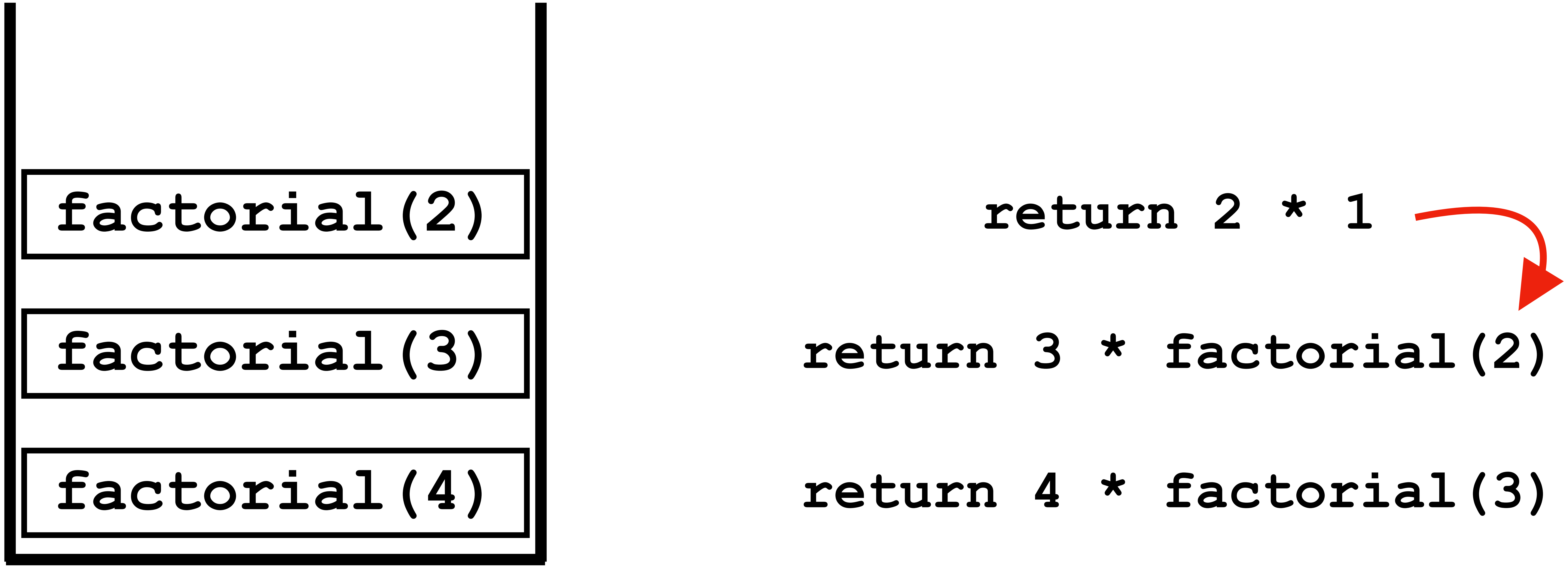



```
return 2 * factorial(1)
return 3 * factorial(2)
return 4 * factorial(3)
```

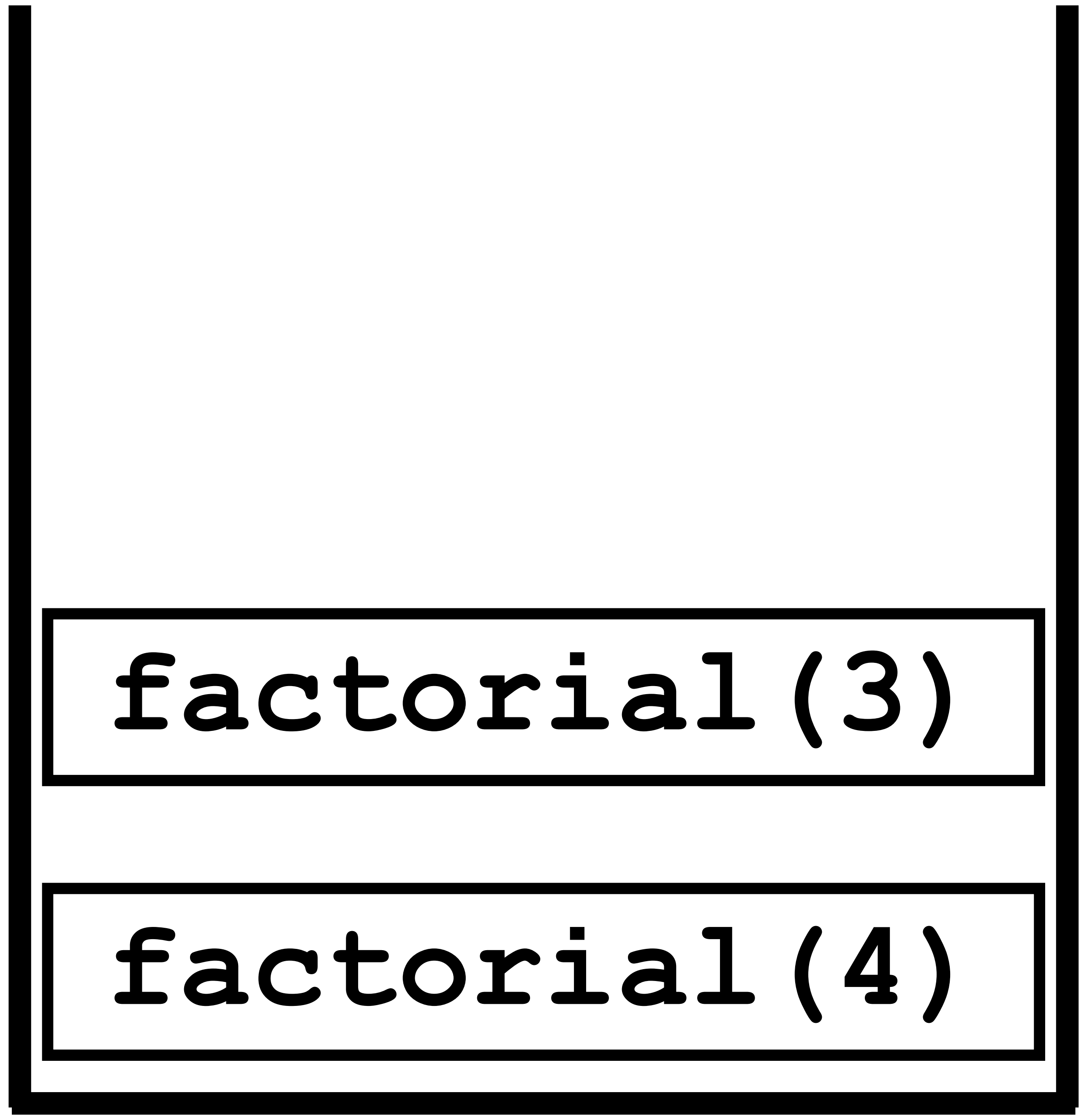
```
int factorial(int x){
   if(x == 0 || x == 1){
      return 1;
   }
   return x * factorial(x-1);
}
```



```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```

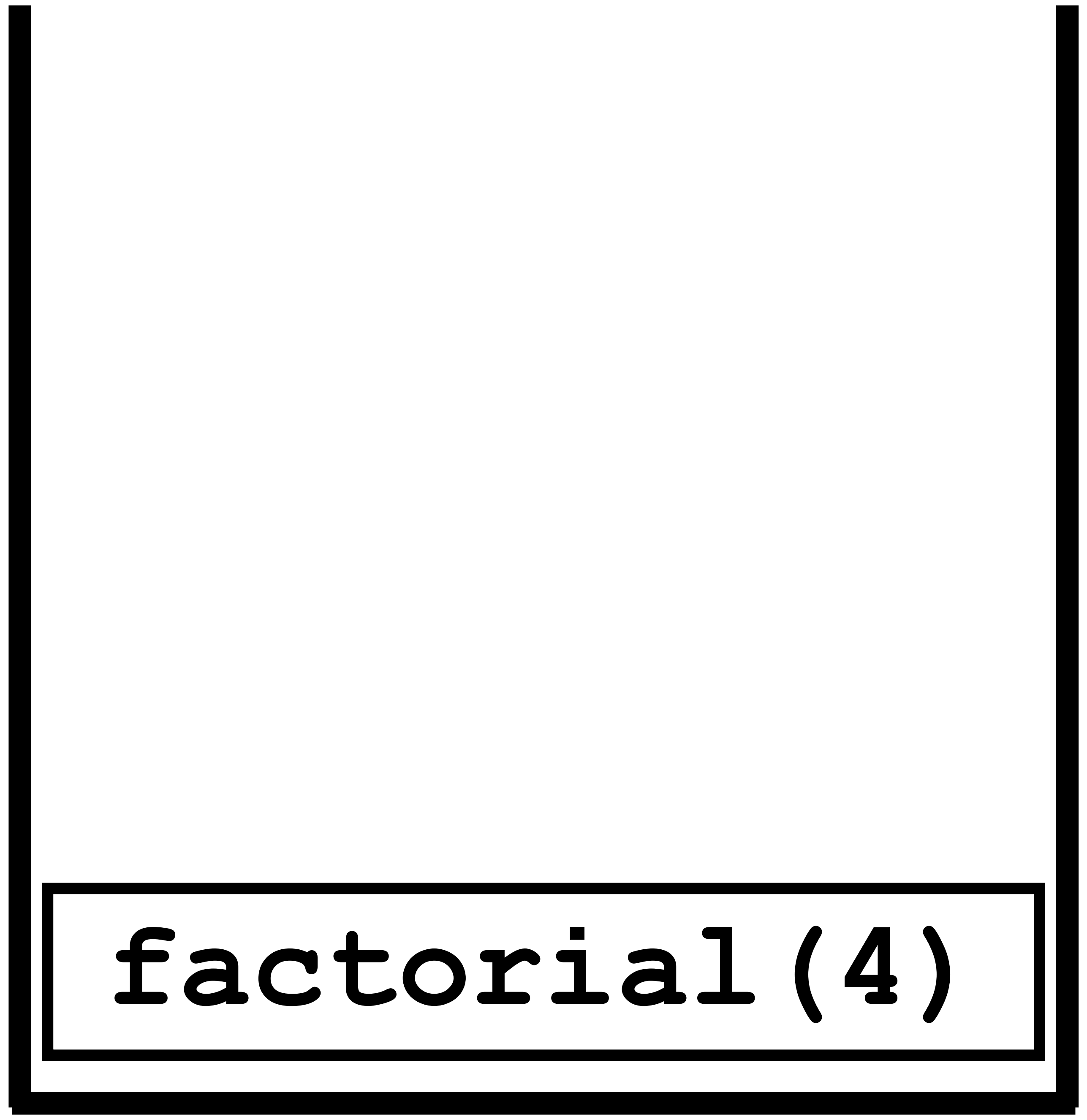


```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```



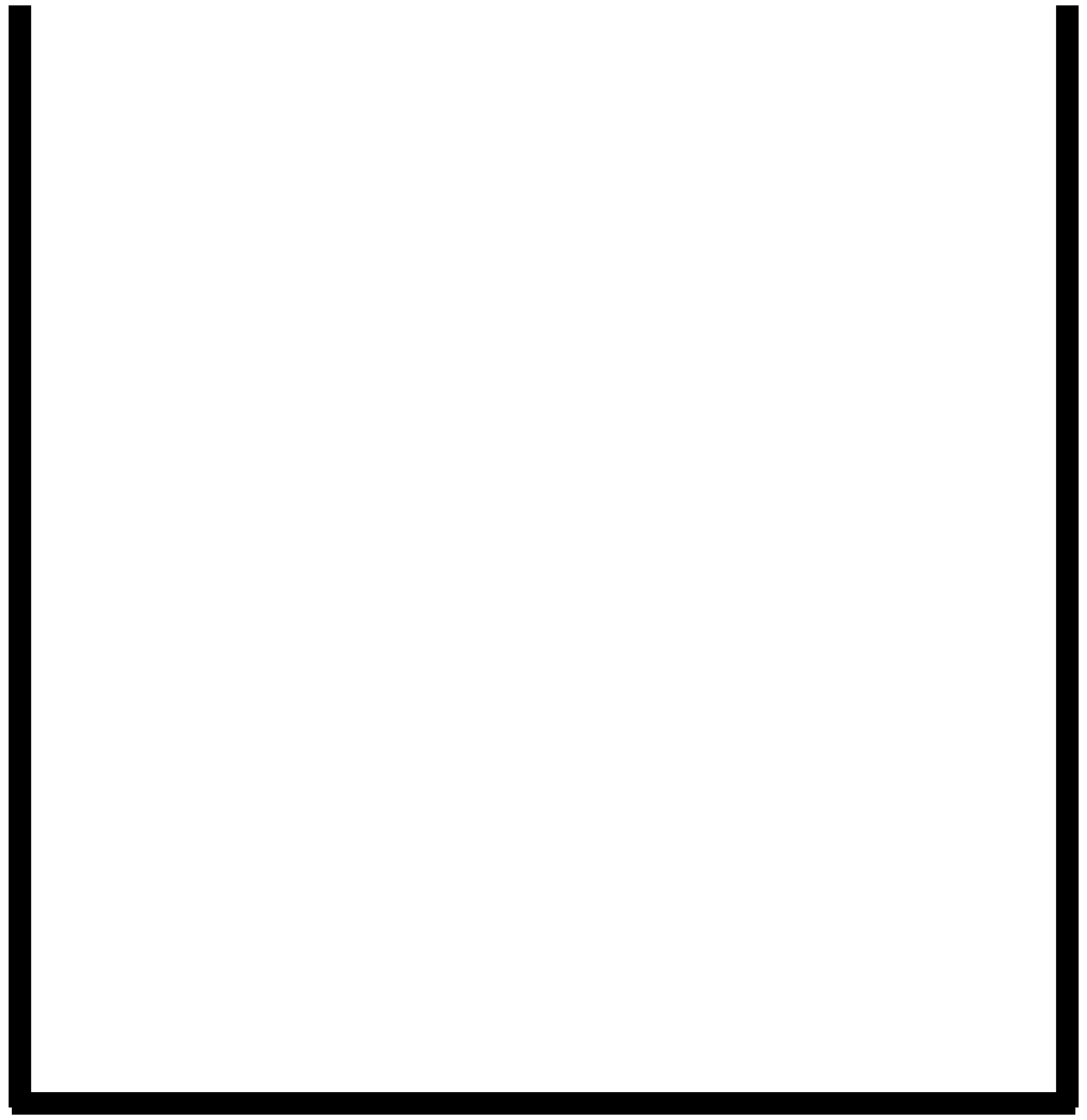

```
return 3 * 2

return 4 * factorial(3)
```

```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```



return 4 * 6

```
int factorial(int x){
  if(x == 0 || x == 1){
    return 1;
  }
  return x * factorial(x-1);
}
```



Result: 24

Recursion - Practice

The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,
F(0) = 0, F(1) = 1
F(n) = F(n-1) + F(n - 2), for n > 1.
Write a recursive function to calculate F(n) given n.

```
F(4) => F(3) + F(2) = 2 + 1 = 3
```

```
int fibonacci(int n){



}
```

Recursion - Practice

```
int fib(int n){
  if(n == 0 || n == 1){
    return n;
  }
  return fib(n-1) + fib(n-2);
}
```