CS 161
Intro to CS I

Recursion

Oregon State University

Example: Factorial

- Definition

  0!=1;

  n!=n * (n-1) * … * (n-(n-1)) * 1 = n * (n-1)! ; n > 0

  iterative — recursive

Iterative Factorial

factorial(0) = 1;

factorial(n) = n*n-1*n-2*…*n-(n-1)*1;

```
long factorial(int n) {
        long fact;
        if(n==0)
            fact=1;
        else
          for(fact=n; n > 1; n--)
             fact=fact*(n-1);
       return fact;
}
```

Recursive Factorial

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

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

Computing Factorial Iteratively

factorial(4)

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;

Computing Factorial Iteratively

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;

factorial(4) = 4 * 3

Computing Factorial Iteratively

factorial(0) = 1;

factorial(n) = n*(n-1)*…*2*1;

factorial(4) = 4 * 3

= 12 * 2

Computing Factorial Iteratively

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;

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

Computing Factorial Iteratively

factorial(0) = 1;

factorial(n) = n*(n-1)*…*2*1;

factorial(4) = 4 * 3

= 12 * 2

= 24 * 1

= 24

Computing Factorial Recursively

factorial(4)

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

Computing Factorial Recursively

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

factorial(4) = 4 * factorial(3)

Computing Factorial Recursively

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

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))

Computing Factorial Recursively

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

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))

Computing Factorial Recursively

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

factorial(4) = 4 * factorial(3)

= 4 * (3 * factorial(2))

= 4 * (3 * (2 * factorial(1)))

= 4 * (3 * (2 * (1 * factorial(0))))

Computing Factorial Recursively

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

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)))

Oregon State University

Computing Factorial Recursively

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

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))

Computing Factorial Recursively

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

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)

Oregon State University

Computing Factorial Recursively

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

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

Computing Factorial Recursively

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

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

Differences

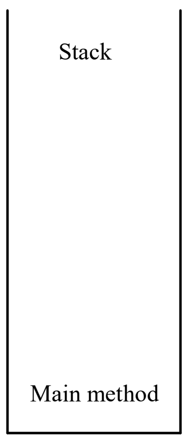
- Pros
  - Readability
- Cons
  - Efficiency
  - Memory

} not a reason to avoid recursion

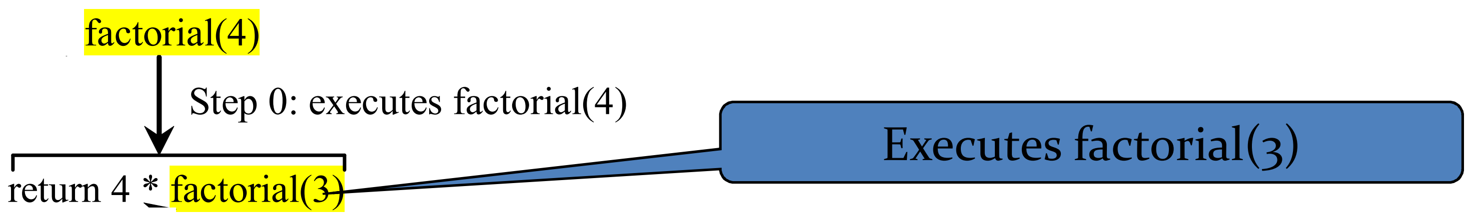
Recursive Factorial

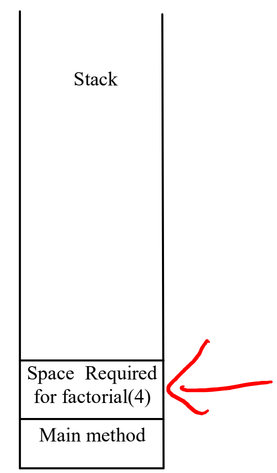
Executes factorial(4)

factorial(4)

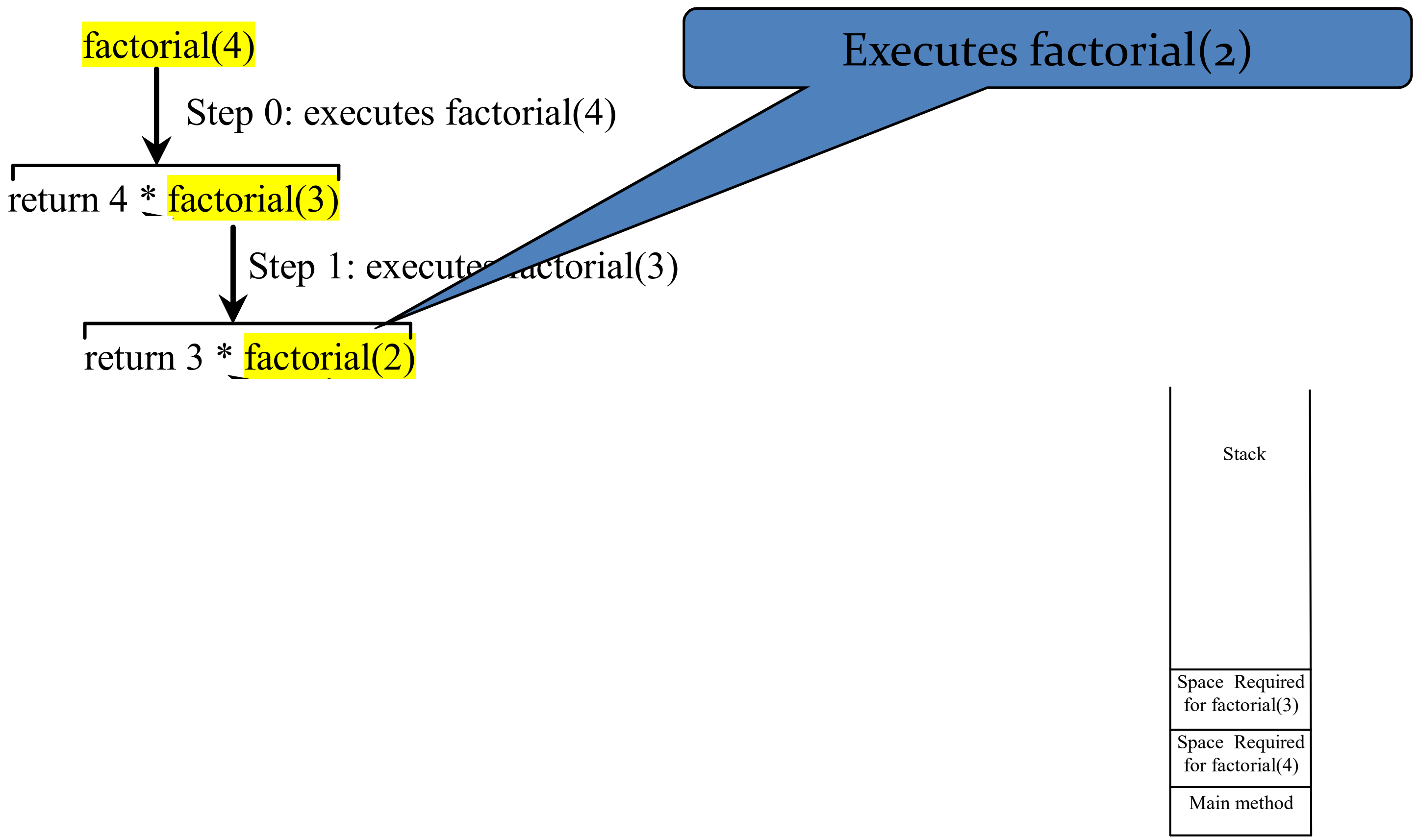


Recursive Factorial

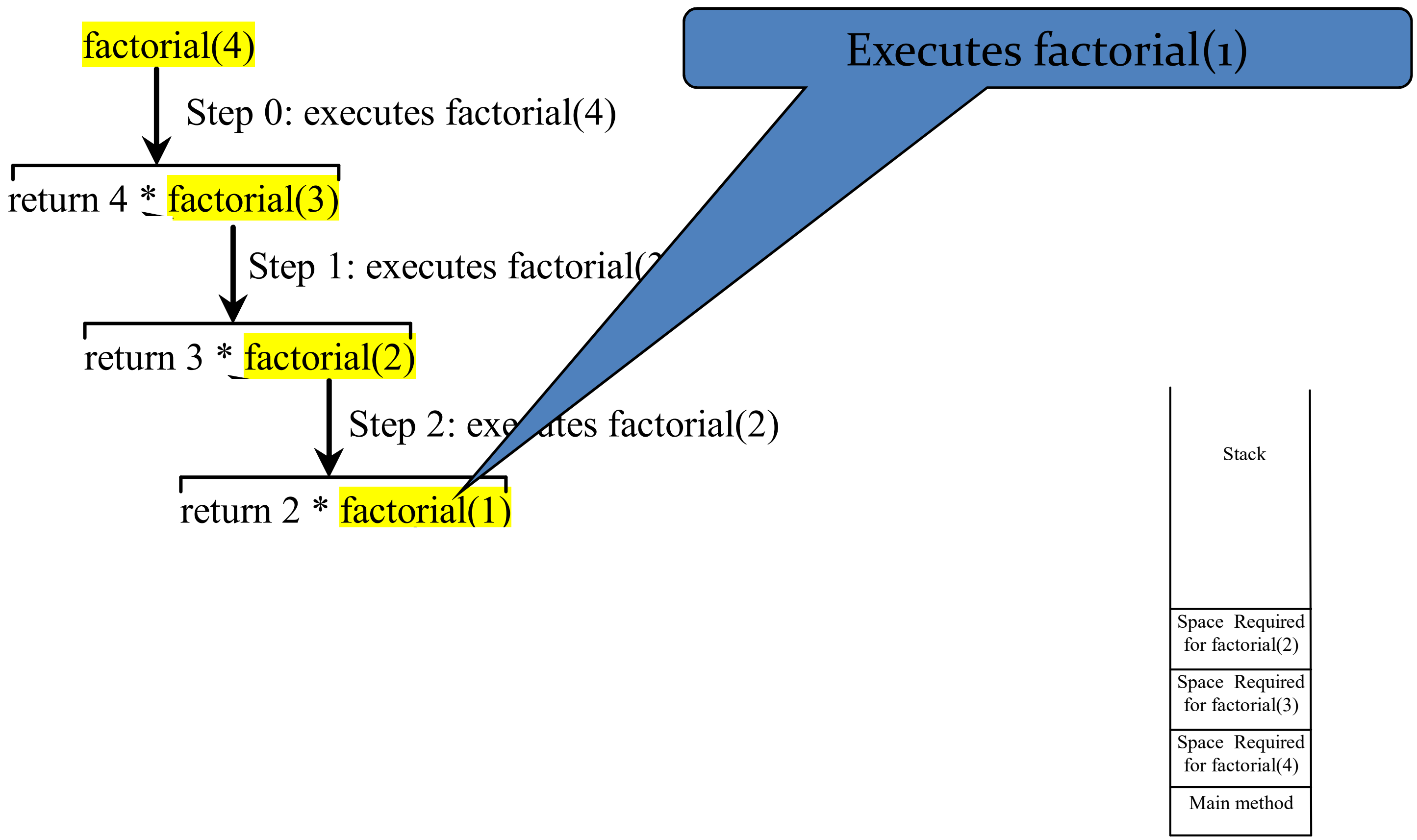




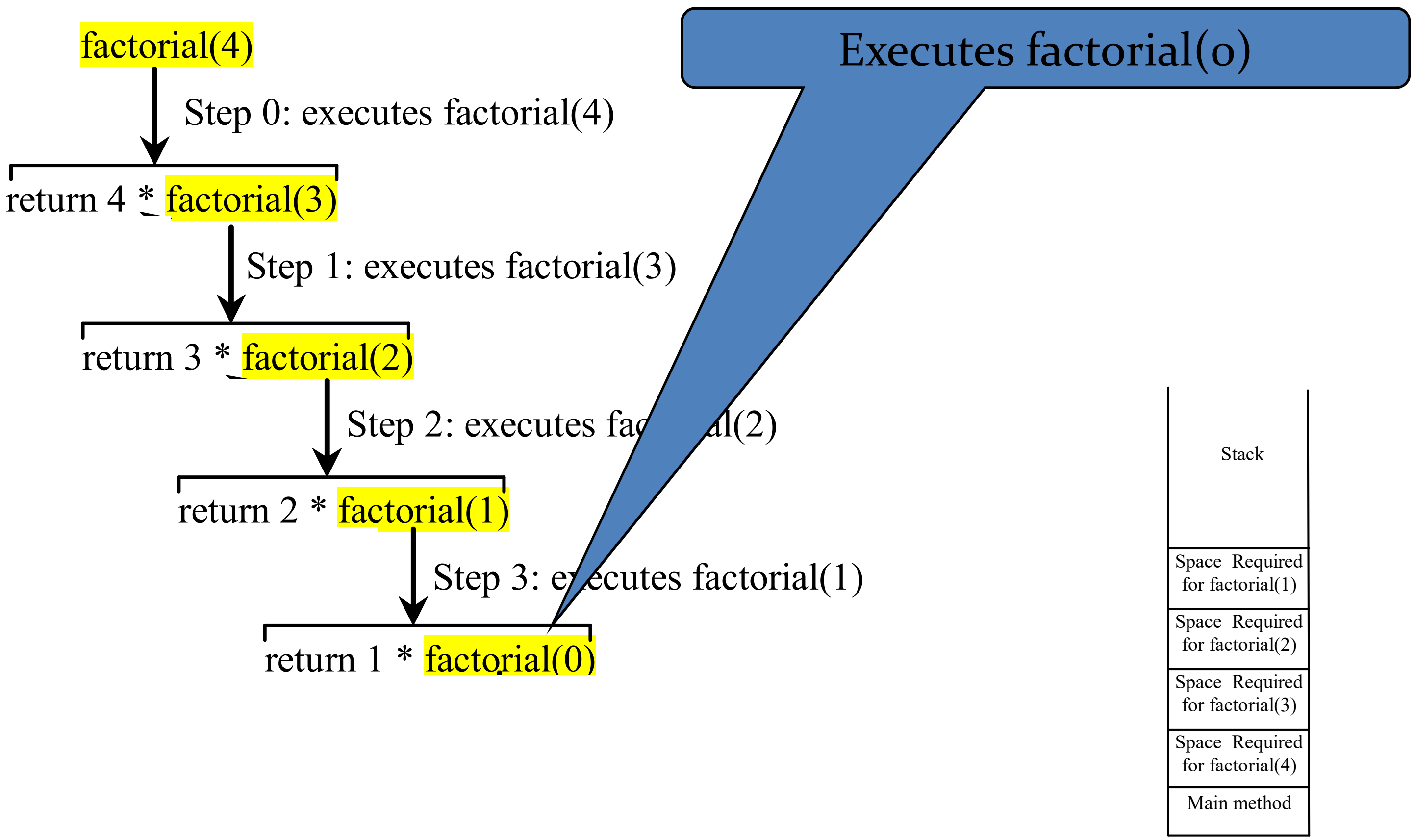
Recursive Factorial



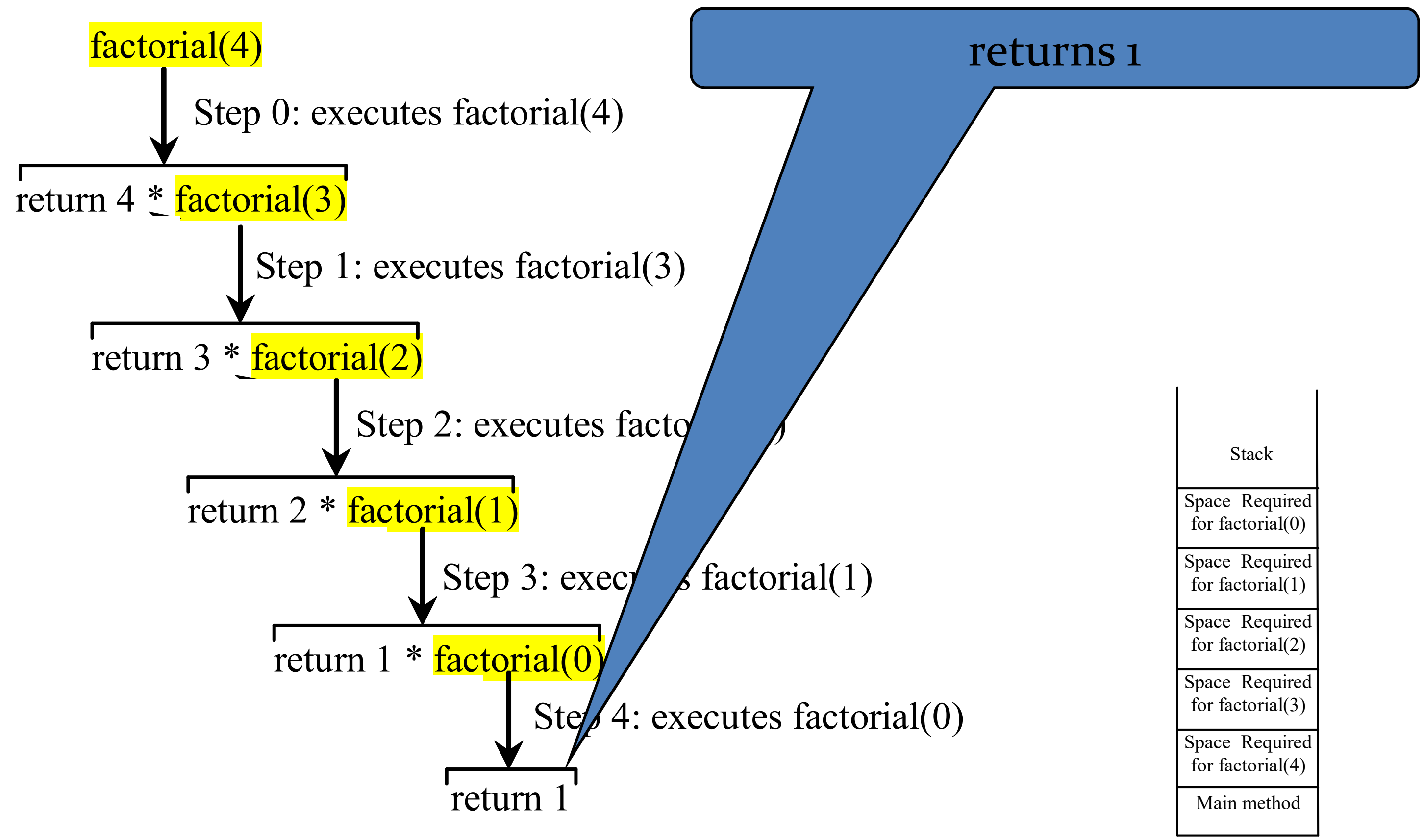
Recursive Factorial



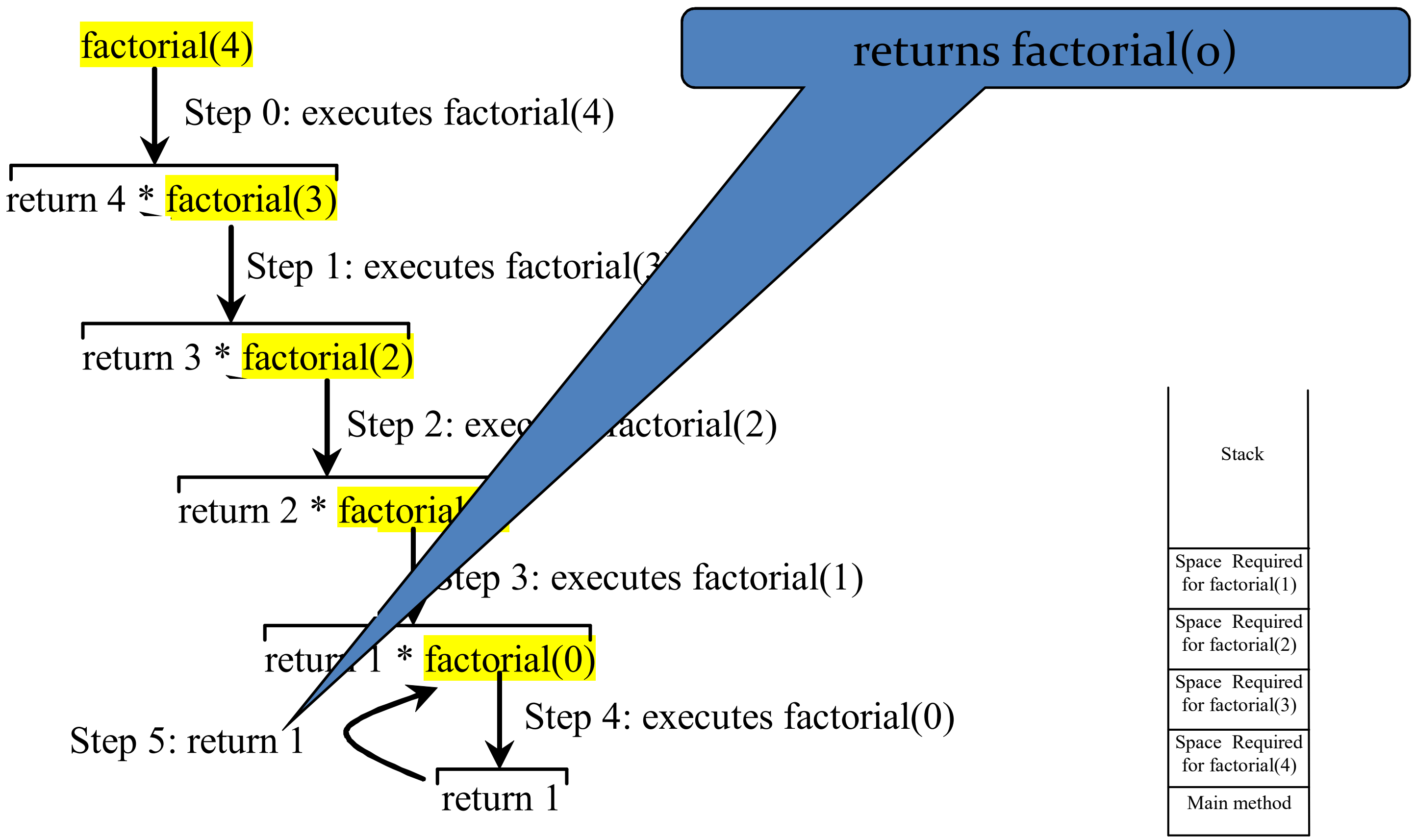
Recursive Factorial



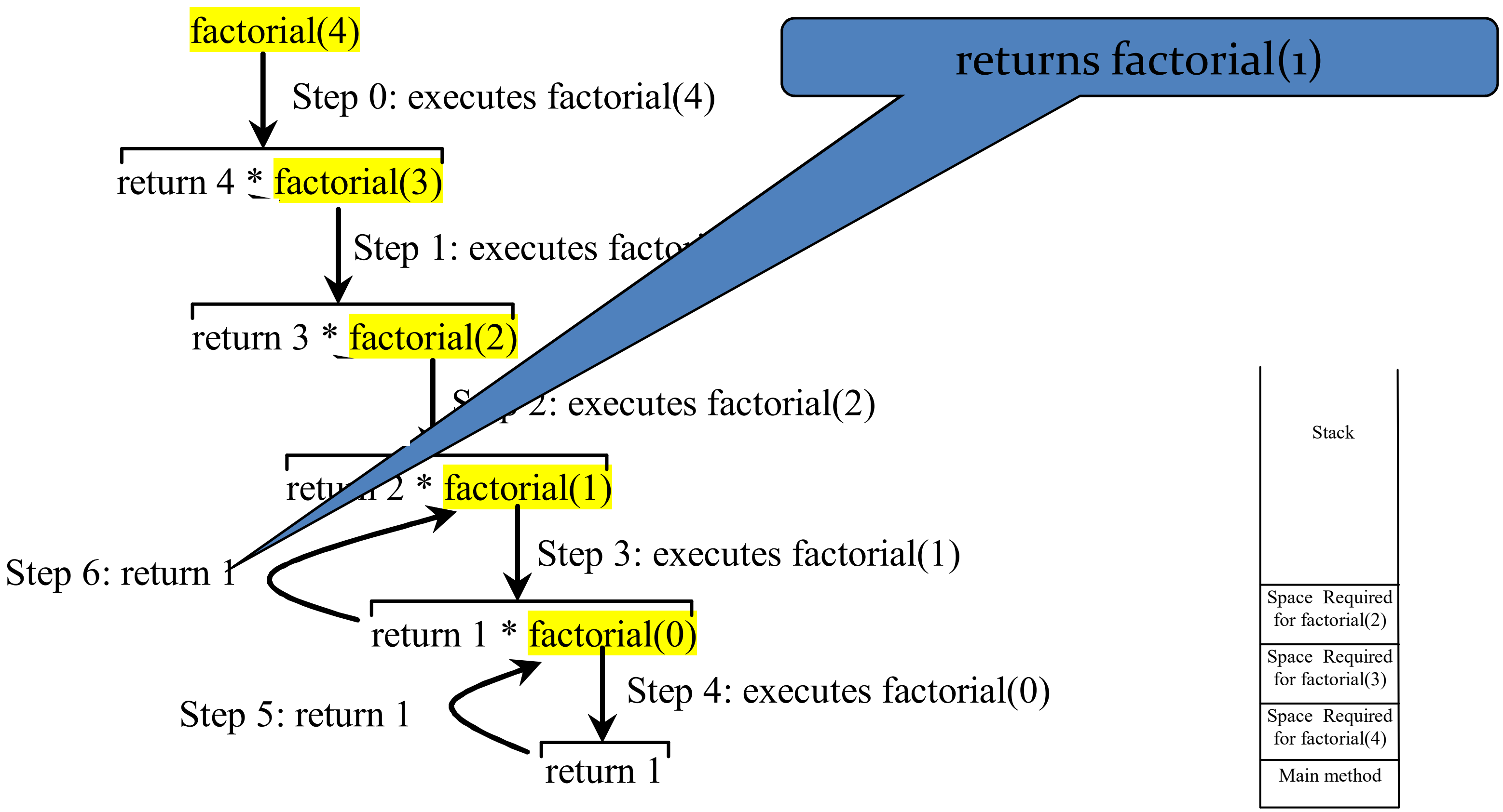
Recursive Factorial



Recursive Factorial

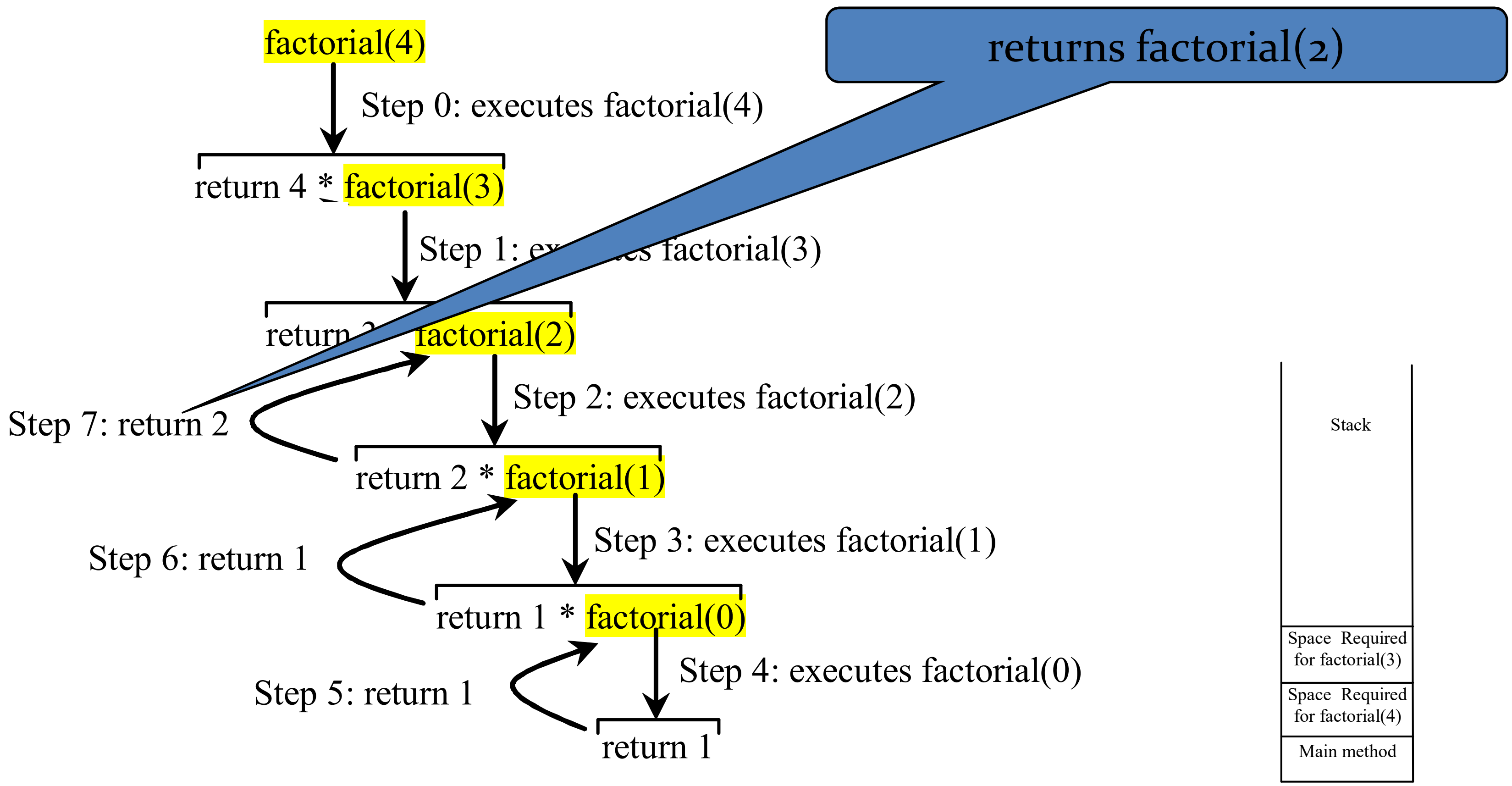


Recursive Factorial

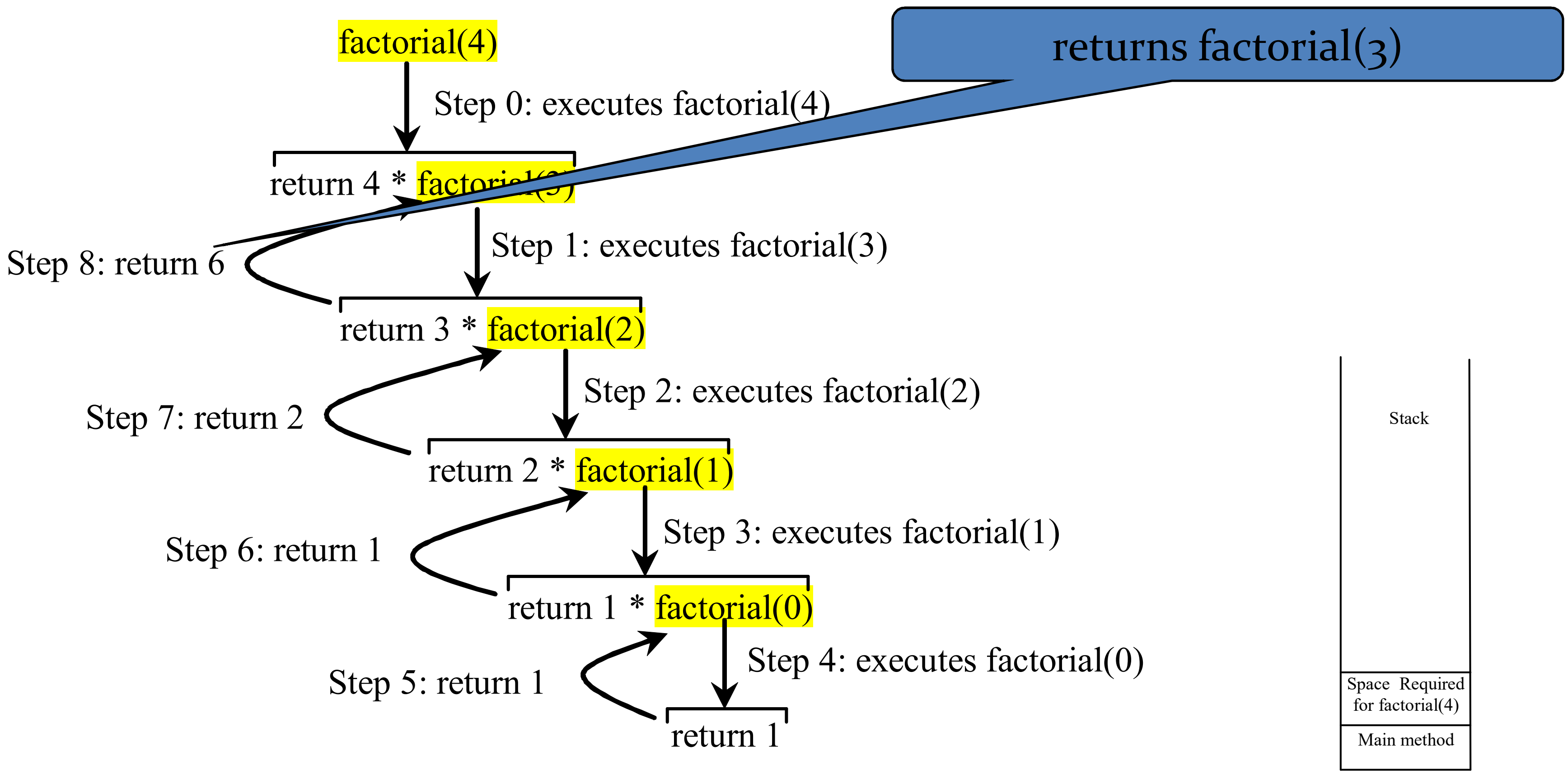


Oregon State University

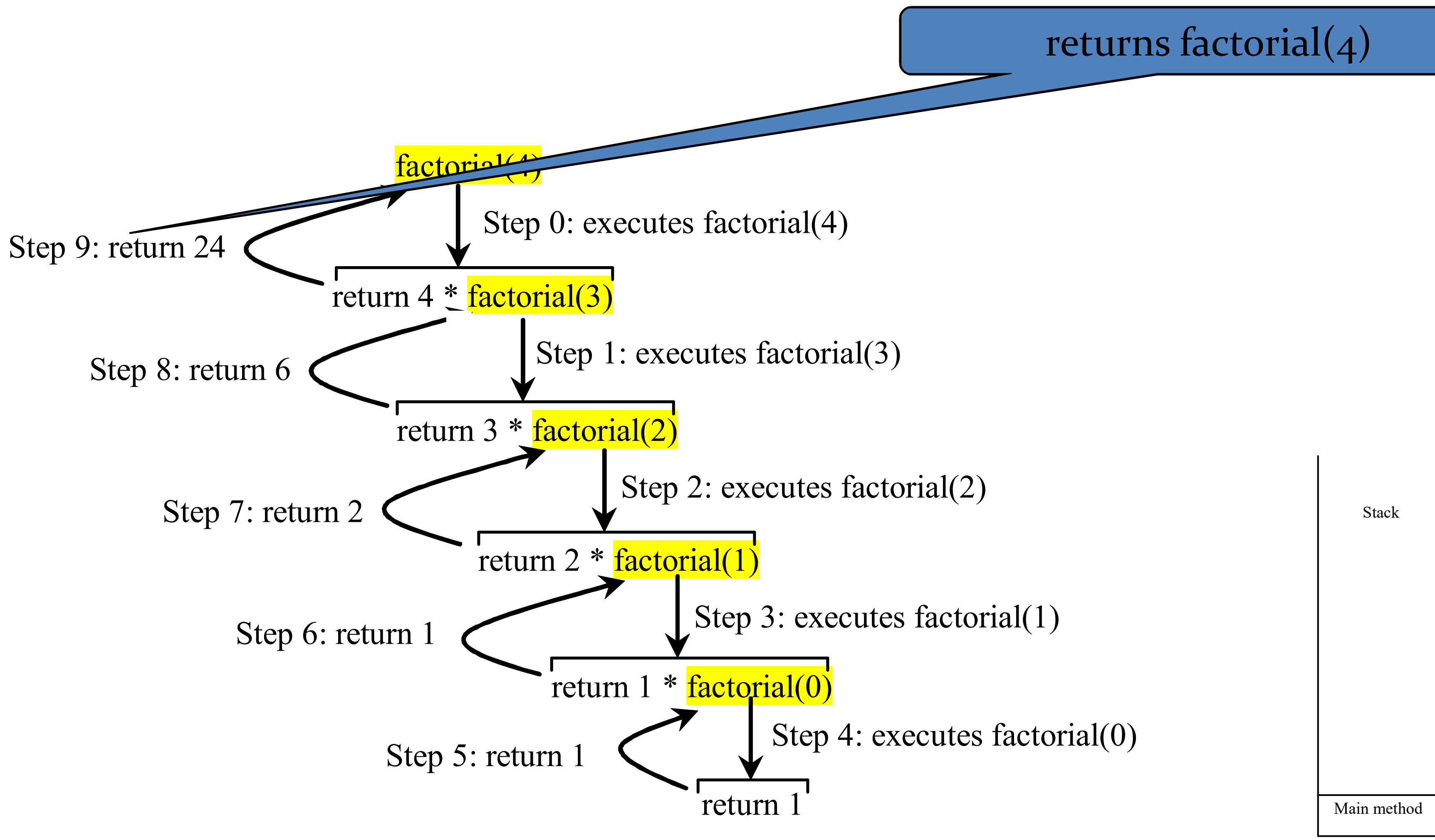
Recursive Factorial



Recursive Factorial



Recursive Factorial



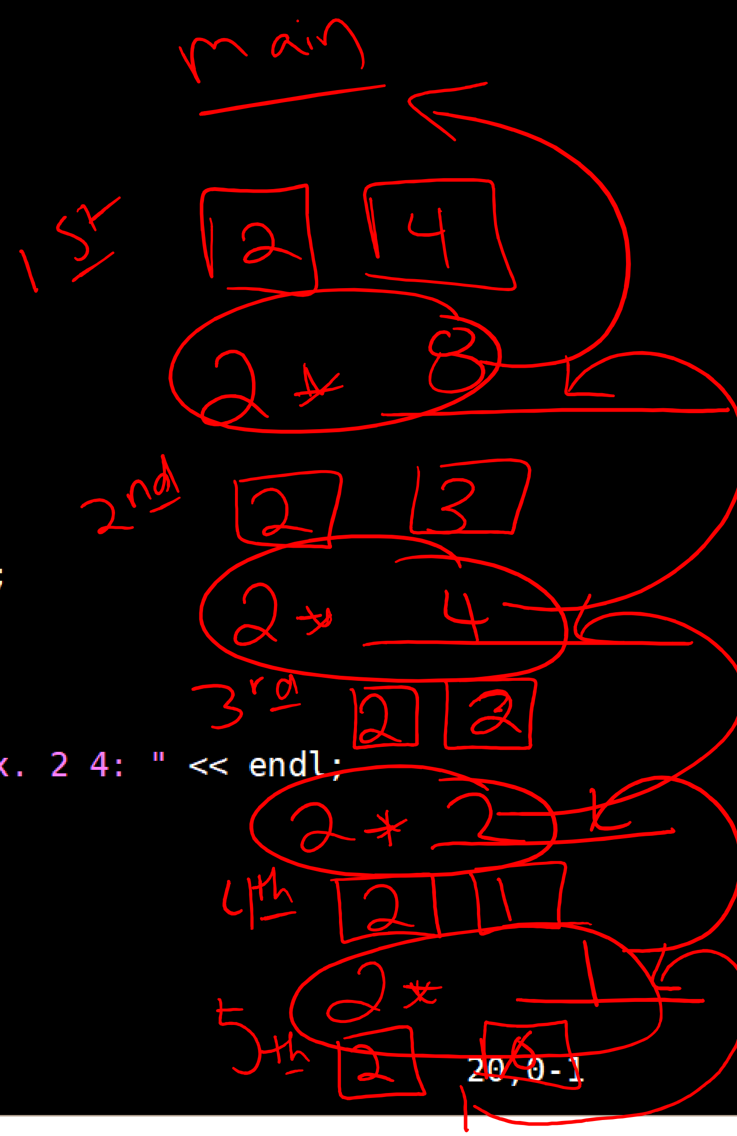
In-class Exercise #4

- Get into groups of 4 – 5.
- Write your own recursive *int pwr()* function that takes two integers as arguments and returns the integer result.
  - What does the function prototype look like?
  - Now, write the function definition…

2. ENGR

Re-attach Fullscreen Stay on top Duplicate Close

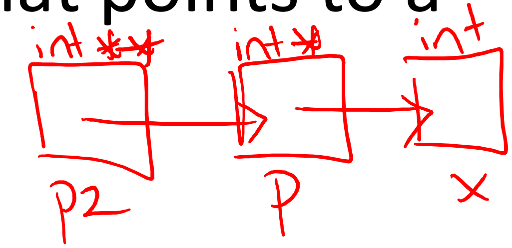
```
#include <iostream>

using namespace std;

int power(int base, int exponent) {
   int p=1;
   for(int i=0; i<exponent; i++)
      p*=base;

   return p;
}

int pwr(int base, int exponent) {
   //base
   if(exponent==0)
      return 1;
   else
      return base*pwr(base,exponent-1);
}

int main() {
   int base, exp;
   cout << "Enter base and exponent, ex. 2 4: " << endl;
   cin >> base;
   cin >> exp;
   cout << power(base, exp) << endl;
   cout << pwr(base, exp) << endl;

   return 0;
}
```



20,0-1 All

In-class Exercise
Pointers vs. References

- What if you made a pointer (p2) that points to a pointer (p) to an int (x)?
  - What would the picture look like?
  - Write the code for this picture.



int x, *p, **p2;
p = &x;
p2 = &p;

*p gives contents of x
*p2 gives contents of p, addr. of x
**p2 gives contents of x

- Can you make this same picture for references? no!
  - What if you had two references, r and r2?