RECURSION

Recursion

- A recursive definition is when something is defined *partly* in terms of itself
- Here's the mathematical definition of factorial:

$$\text{factorial}(n) = \begin{cases} 1, \text{ if } n <= 1 \\ n * \text{factorial}(n - 1) \text{ otherwise} \end{cases}$$

- Here's the programming definition of factorial:

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

Supporting recursion

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

- If you call x = factorial(3), this enters the factorial method with n=3 on the stack
- | factorial calls itself, putting n=2 on the stack
- | | factorial calls itself, putting n=1 on the stack
- | | factorial returns 1
- | factorial has n=2, computes and returns 2*1 = 2
- factorial has n=3, computes and returns 3*2 = 6

Factorial

- x = factorial(3)

3 is put on stack as n

- static int factorial(int n) {
  //n=3
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

r is put on stack with value 1

All references to r use this r

All references to n use this n

Now we recur with 2...

Factorial

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

2 is put on stack as n

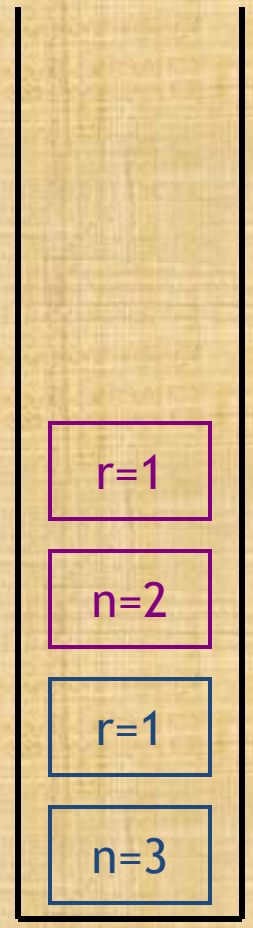
- static int factorial(int n) {//n=2
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

r is put on stack with value 1

Now using this r

And this n

Now we recur with 1...



Factorial

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

1 is put on stack as n

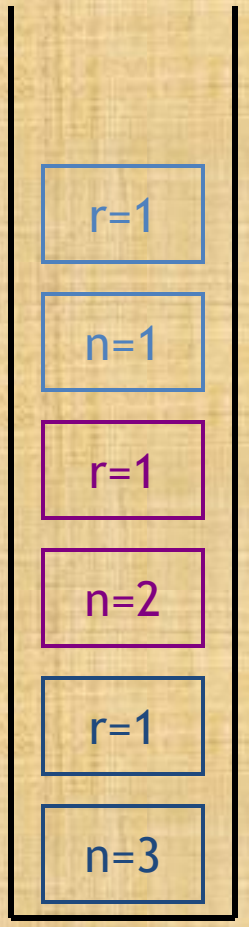
- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

r is put on stack with value 1

Now using this r

And this n

Now we pop r and n off the stack and return 1 as factorial(1)



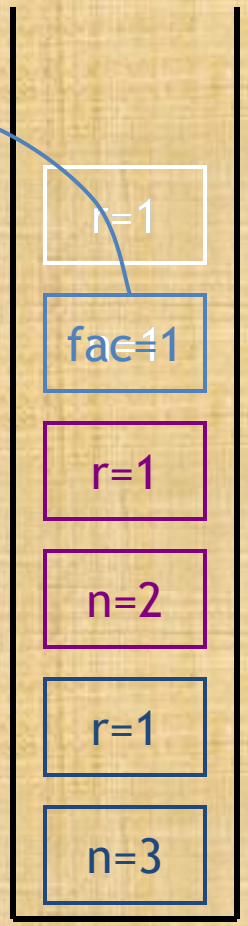
Factorial

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

- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

Now using this r

And this n



Now we pop r and n off the stack and return 1 as factorial(1)

Factorial

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

- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

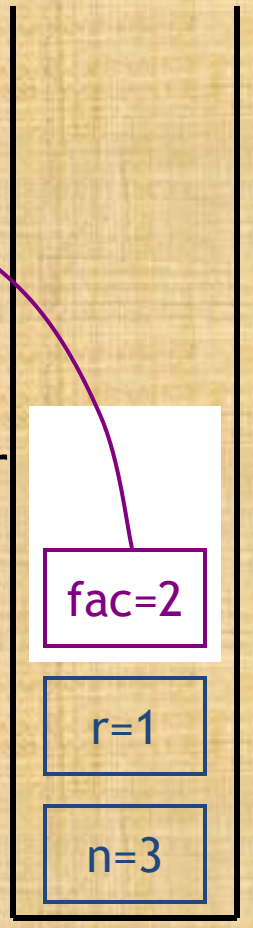
1

2 * 1 is 2;
Pop r and n;

Return 2

Now using this r

And this n



Factorial

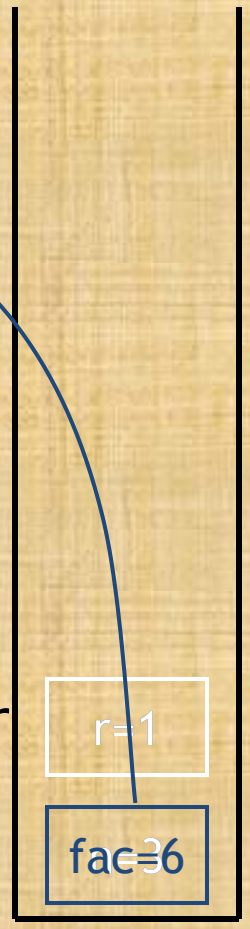
- x = factorial(3)

- static int factorial(int n) {
  int r = 1;
  if (n <= 1) return r;
  else {
  r = n * factorial(n - 1);
  return r;
  }
  }

2

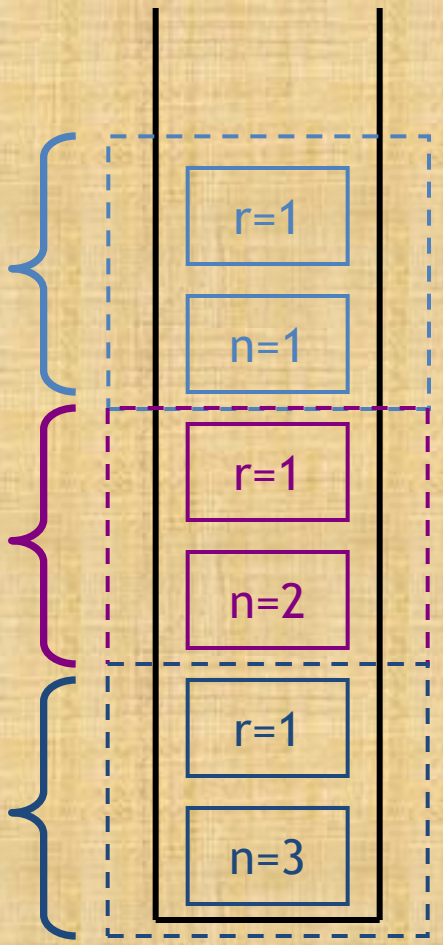
3 * 2 is 6;
Pop r and n;

Return 6



Stack frames

- Rather than pop variables off the stack one at a time, they are usually organized into stack frames
- Each frame provides a set of variables and their values
- This allows variables to be popped off all at once
- There are several different ways stack frames can be implemented



Summary

- Stacks are useful for working with any nested structure, such as:
  - Arithmetic expressions
  - Nested statements in a programming language
  - Any sort of nested data