Recursion II

Factorial (iterative)

```
long long fact(int n)
{
  long long answer = 1;
  for (int x = 1; x <= n; x++)
    answer *= n;
  return answer;
}
```

Factorial

- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = 1 * 2 * 3
- fact(4) = 1 * 2 * 3 * 4
- fact(5) = 1 * 2 * 3 * 4 * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = 1 * 2 * 3
- fact(4) = 1 * 2 * 3 * 4
- fact(5) = 1 * 2 * 3 * 4 * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = 1 * 2 * 3
- fact(4) = 1 * 2 * 3 * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = 1 * 2 * 3
- fact(4) = 1 * 2 * 3 * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = 1 * 2 * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = 1 * 2 * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = fact(2) * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = 1 * 2
- fact(3) = fact(2) * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = fact(1) * 2
- fact(3) = fact(2) * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = fact(1) * 2
- fact(3) = fact(2) * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5

- Let's look at this problem a different way:
- fact(1) = 1
- fact(2) = fact(1) * 2
- fact(3) = fact(2) * 3
- fact(4) = fact(3) * 4
- fact(5) = fact(4) * 5
- General formula:
  - **fact(n) = 1 [for n = 1]**
  - **fact(n) = fact(n-1) * n [for n > 1]**

Factorial (recursive)

```
long long fact(n)
{
  if (n == 1)
    return 1;
  else
    return fact(n-1) * n;
}
```

Uppercase (iterative)

```
string uc(string s)
{
  string answer = “”;
  for (int x = 0; x < s.size(); x++)
    answer += toupper(s);
  return answer;
}
```

```
long long fact(int n)
{
  long long answer = 1;
  for (int x = 1; x <= n; x++)
    answer *= n;
  return answer;
}
```

Uppercase

uc("d") = "D"
uc("cd") = "CD"
uc("bcd") = "BCD"
uc("abcd") = "ABCD"

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + toup('d')
uc("bcd") = toup('b') + toup('c') + toup('d')
uc("abcd") = toup('a') + toup('b') + toup('c') + toup('d')

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + toup('d')
uc("bcd") = toup('b') + toup('c') + toup('d')
uc("abcd") = toup('a') + toup('b') + toup('c') + toup('d')

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + toup('d')
uc("bcd") = toup('b') + toup('c') + toup('d')
uc("abcd") = toup('a') + uc("bcd")

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + toup('d')
uc("bcd") = toup('b') + toup('c') + toup('d')
uc("abcd") = toup('a') + uc("bcd")

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + toup('d')
uc("bcd") = toup('b') + uc("cd")
uc("abcd") = toup('a') + uc("bcd")

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + toup('d')
uc("bcd") = toup('b') + uc("cd")
uc("abcd") = toup('a') + uc("bcd")

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + uc("d")
uc("bcd") = toup('b') + uc("cd")
uc("abcd") = toup('a') + uc("bcd")

Uppercase

(toup stands for toupper)

uc("d") = toup('d')
uc("cd") = toup('c') + uc("d")
uc("bcd") = toup('b') + uc("cd")
uc("abcd") = toup('a') + uc("bcd")

General rule:

uc(s) = toup(s) [if s is one letter long]
uc(s) = toup(s[0]) + uc(rest of s) [otherwise]

Let's look at the C++ solution

What does this do?

```
void weird(int n)
{
    if (n == 0)
        return;
    else
    {
        cout << n << endl;
        weird(n - 1);
    }
}
```

Tower of Hanoi