



- On paper, write a C++ function that takes a single int argument (n) and returns the product of all the integers between 1 and n.
 - Could use a for or while loop.
- (This is actually a useful function in science and mathematics, called the factorial function.)

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

- Let's look at this problem a different way:
- $\text{fact}(1) = 1$
- $\text{fact}(2) = 1 * 2$
- $\text{fact}(3) = 1 * 2 * 3$
- $\text{fact}(4) = 1 * 2 * 3 * 4$
- $\text{fact}(5) = 1 * 2 * 3 * 4 * 5$
- Notice that each product involves computing the entire product on the row above.

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

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

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

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

- Let's look at this problem a different way:
- $\text{fact}(1) = 1$
- $\text{fact}(2) = 1 * 2$
- $\text{fact}(3) = 1 * 2 * 3$
- $\text{fact}(4) = 1 * 2 * 3 * 4$
- $\text{fact}(5) = 1 * 2 * 3 * 4 * 5$
- Let's reformulate the definition of a factorial to take advantage of this.

Recursion

- A ***recursive function*** is a function that calls itself.
- Recursive functions are used to solve problems where the ***solution to the problem involves solving a smaller version of the same problem.***

- A recursive function has two parts:
- **Base case:** How to solve the smallest version(s) of the problem that we care about.
- **Recursive case:** How to reduce a bigger version of the problem to a smaller version.
 - In order to work, the recursive case (when applied over and over) must eventually reduce every size of the problem down to the base case.
- What are these for factorial?

How does this work in C++

- Recursion works (in all modern programming languages) because:
 - All variables are local.
 - The stack makes new memory for local variables every time a function is called.
- Lets look at the stack when we call `factrec(3)`.

Why is this useful?

- Any loop (for/while) can be replaced with a recursive function that does the same thing.
 - Some languages don't include loops!
- Because we started with Python and C++, we naturally see things in terms of loops.
- Some problems have a "naturally" recursive solution that is hard to solve with a loop.
- Other problems have solutions that work equally well recursively or with loops (iteratively).

How to "get" recursion

An "instance" of a problem is a single example or occurrence of that problem.

- Forget all loops.
- To find the base case:
 - "What is the smallest version of this problem I would ever care about solving?"
- To find the recursive case:
 - "If I have a *instance* of the problem, how can I phrase how to solve the problem in terms of a smaller instance?"

Trust the recursion

- Base case is usually easy ("When do I stop?")
- In recursive case:
 - Break the problem into two parts (not necessarily the same size):
 - A part I can solve "now."
 - The answer from a smaller instance of the problem.
 - ***Assume the recursive call does the right thing.***
 - Figure out how to combine the two parts.

Try this

- I want to write a function that returns an uppercase version of an entire string
 - `uc("hello")` would return "HELLO"
- All C++ gives me is a function that returns the uppercase of a single character (`toupper`).
- To solve this recursively, find the recursive case and the base case.