ECE 2400 Computer Systems Programming
Fall 2017

T02 C Recursion

School of Electrical and Computer Engineering
Cornell University

revision: 2017-09-10-22-52

- Recursion is when the algorithm is defined in terms of itself
- No new syntax or semantics
- Understanding recursion simply involves applying what we have already learned with respect to functions, conditionals, iteration

1. Computing Factorial Using Iteration and Recursion

Recall from mathematics, the factorial of a number (n!) is:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

So in other words:

$0!$	$=$		$=$	1
$1!$	$=$		$=$	1
$2!$	$=$	1×2	$=$	2
$3!$	$=$	$1 \times 2 \times 3$	$=$	6
$4!$	$=$	$1 \times 2 \times 3 \times 4$	$=$	24
$5!$	$=$	$1 \times 2 \times 3 \times 4 \times 5$	$=$	120

We can write a function to calculate factorial using a `for` loop:

```
1  int factorial( int n ) {
2    int result = 1;
3    for ( int i = 1; i <= n; i++ )
4      result = result * i;
5    return result;
6  }
```

- The loop implementation does not really resemble the original mathematical formulation
- The mathematical formulation is inherently recursive
- Can we implement factorial more directly using recursion?

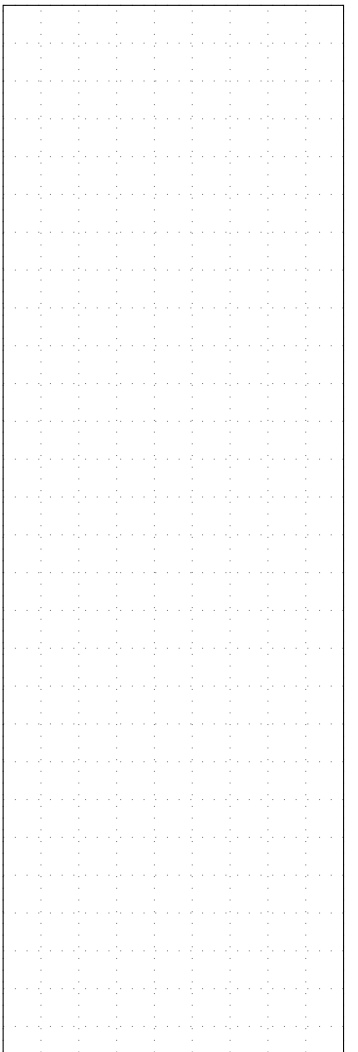
$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

We can use the exact same "by-hand" execution approach we learned in the previous topic to understand recursion.

```
1  int factorial( int n )
2  {
3    // base case
4    if ( n == 0 ) {
5      return 1;
6    }
7    // recursive case
8    if ( n > 0 ) {
9      return n *
10       factorial(n-1);
11   }
12 }
13
14 int main()
15 {
16   factorial(3);
17   return 0;
18 }
```

Questions:

- What if n is negative?
- What if the execution arrow reaches end of a non-void function without encountering a return statement?



2. Computing Fibonacci Using Iteration and Recursion

Recall from mathematics, the Fibonacci number is:

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$$

So in other words:

fib(0)	=		=	0
fib(1)	=		=	1
fib(2)	=	0 + 1	=	1
fib(3)	=	1 + 1	=	2
fib(4)	=	1 + 2	=	3
fib(5)	=	2 + 3	=	5
fib(6)	=	3 + 5	=	8
fib(7)	=	5 + 8	=	13
fib(8)	=	8 + 13	=	21

We can write a function to calculate the n^{th} Fibonacci number using a `for` loop:

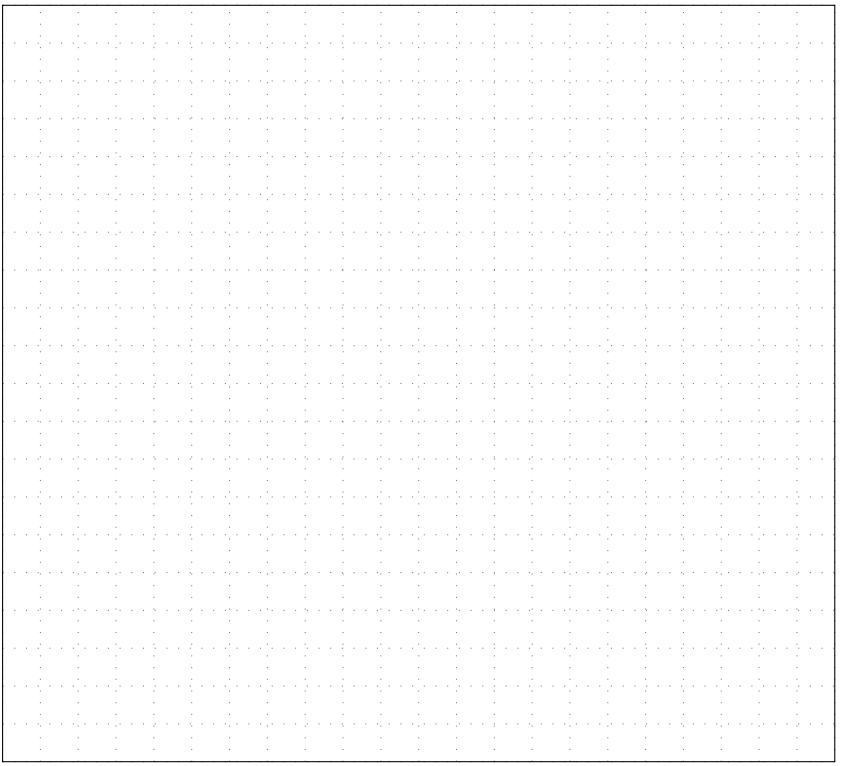
```
1  int fib( int n ) {
2
3    // base cases
4    if (n == 0) return 0;
5    if (n == 1) return 1;
6
7    int fib_minus2 = 0;
8    int fib_minus1 = 1;
9    int result     = 0;
10
11   for ( int i=2; i<=n; i++ ) {
12
13     result = fib_minus1
14            + fib_minus2;
15
16     fib_minus2 = fib_minus1;
17     fib_minus1 = result;
18
19   }
20   return result;
21 }
```

This page intentionally left blank.

- The loop implementation does not really resemble the original mathematical formulation
- The mathematical formulation is inherently recursive
- Can we implement factorial more directly using recursion?

$$\text{fib}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$$

Illustrating call tree for `fib`



3. Writing a Recursive Function

Write pseudo-code for a recursive function which draws the tick marks on a vertical ruler. The middle tick mark should be the longest and mark the 1/2 way point, slightly shorter tick marks should mark the 1/4 way points, even slightly shorter tick marks should mark the 1/8 way points and so on. The recursive function should take three arguments: the index of the top tick mark, the index of the bottom tick mark, and the height of the middle tick mark. Assume the number of tick marks is always a power of two (e.g., 2, 4, 8, 16, etc). Use `printf` to display the tick marks.

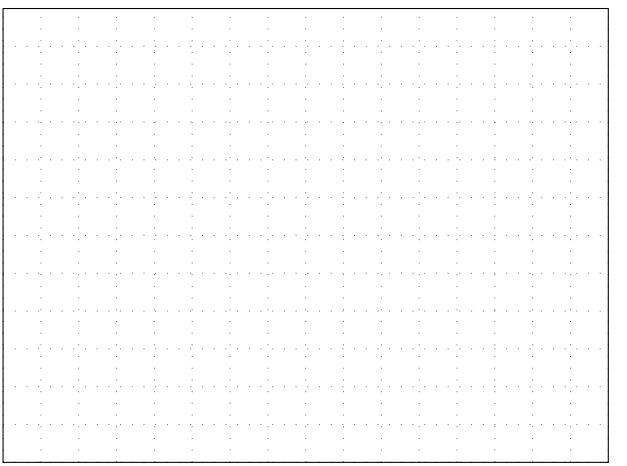
`ruler(0,16,4)`

```
0
1  -
2  --
3  -
4  ---
5  -
6  --
7  -
8  ----
9  -
10 --
11 -
12 ---
13 -
14 --
15 -
16
```

```
void ruler( int top, int bottom, int height ) {
```

- Step 1: Work an example yourself
- Step 2: Write down what you just did
  - What is the base case?
  - What is the recursive case?
- Step 3: Generalize your steps
- Step 4: Test your algorithm
- Step 5: Translate to code

Think about the recursive call tree?



Manually work through example ruler