ECE 2400 Computer Systems Programming
Fall 2021

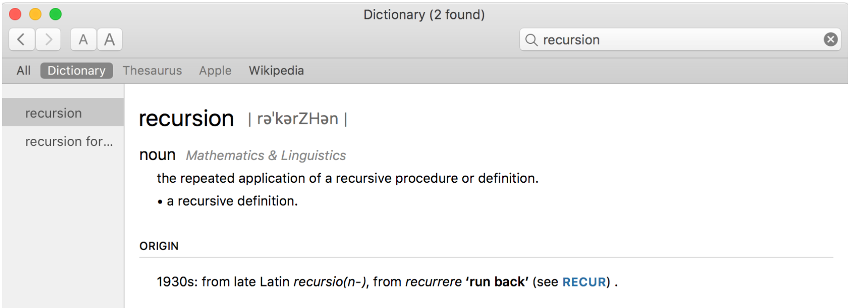
Topic 2: C Recursion

School of Electrical and Computer Engineering
Cornell University

revision: 2021-08-29-22-42

zyBooks The zyBooks logo is used to indicate additional material included in the course zyBook which will not be discussed in detail in lecture. Students are responsible for all material covered in lecture and in the course zyBook.

Our goal is to understand what the word “recursion” means, so let’s look up “recursion” in the dictionary ...



- 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. Single 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:

```c
int factorial( int n ) {
  int result = 1;
  for ( int i = 1; i <= n; i++ )
    result = result * i;
  return result;
}
```

- 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.

```
int factorial( int n )
{
  // base case
  if ( n == 0 ) {
    return 1;
  }
  // recursive case
  if ( n > 0 ) {
    return n *
      factorial(n-1);
  }
}

int main()
{
  int a = factorial(3);
  return 0;
}
```

stack

Questions:

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

2. Multiple Recursion

Recall from mathematics, the Fibonacci sequence is a sequence of integers such that every number after the first two is the sum of the two preceding ones:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

The numbers in the Fibonacci sequence are called "Fibonacci numbers". By definition, the first two numbers in the Fibonacci sequence are 0 and 1. Ancient scholars realized the importance of this sequence in both mathematics and nature. Fibonacci sequences can be found in the arrangement of leaves on a stem or patterns in a pine cone.

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

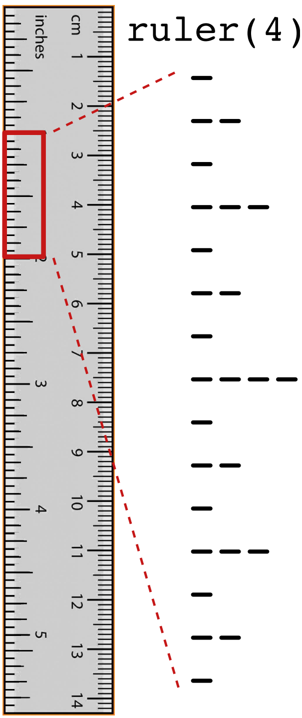
```c
int fib( int n ) {

  // by definition
  if (n == 0) return 0;
  if (n == 1) return 1;

  int fib_minus2 = 0;
  int fib_minus1 = 1;
  int result     = 0;

  for ( int i=2; i<=n; i++ ) {

    result = fib_minus1
           + fib_minus2;

    fib_minus2 = fib_minus1;
    fib_minus1 = result;

  }
  return result;
}
```

Can we implement factorial more elegantly using recursion?

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 function should take one argument: the height of the middle tick mark (i.e., the number of dashes). The function should always return 0.




```
int ruler( int height ) {
```

```
ruler(1)
-
```

```
ruler(2)
-
--
-
```

```
ruler(3)
-
--
-
---
-
--
-
```

```
ruler(4)
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
```

```
ruler(5)
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
-----
-
--
-
---
-
--
-
----
-
--
-
---
-
--
-
```

- Step 1: Work an example yourself
  - height = 2, height = 3
- Step 2: Write down what you just did
  - What is the base case?
  - What is the recursive case?
- Step 3: Generalize your steps
  - for any height
- Step 4: Test your algorithm
  - does it work for height = 4?
- Step 5: Translate to pseudocode

Think about the recursive call tree?

Manually work through example ruler