1 Single Recursion 3

2 Multiple Recursion 6

3 Writing a Recursive Function 8
Our goal is to understand what the word “recursion” means, so we 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:

\[
\begin{align*}
  n! &= \begin{cases} 
    1 & \text{if } n = 0 \\
    n \times (n-1)! & \text{if } n > 0 
  \end{cases} 
\end{align*}
\]

So in other words:

<table>
<thead>
<tr>
<th>n! =</th>
<th>1 \times (n-1)!</th>
<th>n = 0 or n &gt; 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0!</td>
<td>= 1</td>
<td>= 1</td>
</tr>
<tr>
<td>1!</td>
<td>= 1</td>
<td>= 1</td>
</tr>
<tr>
<td>2!</td>
<td>= 1 \times 2</td>
<td>= 2</td>
</tr>
<tr>
<td>3!</td>
<td>= 1 \times 2 \times 3</td>
<td>= 6</td>
</tr>
<tr>
<td>4!</td>
<td>= 1 \times 2 \times 3 \times 4</td>
<td>= 24</td>
</tr>
<tr>
<td>5!</td>
<td>= 1 \times 2 \times 3 \times 4 \times 5</td>
<td>= 120</td>
</tr>
</tbody>
</table>

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;
}
```
1. Single Recursion

- 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?

\[
n! = \begin{cases} 
1 & \text{if } n = 0 \\
n \times (n - 1)! & \text{if } n > 0 
\end{cases}
\]
1. Single Recursion

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

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

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

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 pattern 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. Use printf to display the tick marks.

```c
int ruler( int height ) {
    // Pseudo-code for drawing tick marks
}
```
### 3. Writing a Recursive Function

<table>
<thead>
<tr>
<th>ruler(1)</th>
<th>ruler(2)</th>
<th>ruler(3)</th>
<th>ruler(4)</th>
<th>ruler(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- **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
3. Writing a Recursive Function

Think about the recursive call tree?

Manually work through example ruler