1 Computing Factorial Using Iteration and Recursion 2

2 Computing Fibonacci Using Iteration and Recursion 5

3 Writing a Recursive Function 9
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:

\[
\begin{array}{l}
0! = 1 \\
1! = 1 \\
2! = 1 \times 2 = 2 \\
3! = 1 \times 2 \times 3 = 6 \\
4! = 1 \times 2 \times 3 \times 4 = 24 \\
5! = 1 \times 2 \times 3 \times 4 \times 5 = 120 \\
\end{array}
\]

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.

```java
int factorial( int n )
{
    // base case
    if ( n == 0 ) {
        return 1;
    }

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

int main()
{
    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. 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:

<table>
<thead>
<tr>
<th>fib(n)</th>
<th>0 if n = 0</th>
<th>1 if n = 1</th>
<th>fib(n-1) + fib(n-2) if n &gt; 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib(0) = 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(1) = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(2) = 0 + 1 = 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(3) = 1 + 1 = 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(4) = 1 + 2 = 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(5) = 2 + 3 = 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(6) = 3 + 5 = 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(7) = 5 + 8 = 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fib(8) = 8 + 13 = 21</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We can write a function to calculate the \(n\)th Fibonacci number using a for loop:

```cpp
int fib( int n ) {
    // base cases
    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;
}
```
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.

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

- 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