Solution to practice problem #2 explained with a stack:

```c++
int sumAllByOne(int n) {
    // base case - the simplest case possible is when n = 1
    // then we just return 1
    if (n == 1) return n;

    // recursive call is for smaller input, that is in this case
    // is n - 1 (each step)
    else return n + sumAllByOne(n-1);
}
```

When function can’t return right away some value and calls itself (recursion), stack of calls for this function starts growing, because machine needs to “remember” every call that was made, so the very first call is saved, every subsequent call is placed on top of the previous one. Every layer of a stack (a call to a function) waits for the value returned from the following call. After it receives a value and returns to the call below – the layer is deleted from stack. At the end of the execution of this function, there should be just one layer with a final value on it that is returned to whoever called it from main() function. See below the stack for the example of a function sumAllByOne:

1) Lets assume that user gave us 4, in main() the call is made:
```c++
cout << sumAllByOne(4) << endl;
```

2) sumAllByOne(4) is called, base case is not met yet, the stack is created for the call:

```
1) return 4 + sumAllByOne(3);
    call from main() to sumAllByOne for n = 4 is made
    second line of code in a function was executed

2) return 3 + sumAllByOne(2);
3) return 2 + sumAllByOne(1);
```

3) sumAllByOne(3) can’t return a value right away since the base case is not met yet, so it calls itself as a well-behaved recursion and the stack starts growing:
```
1) return 4 + sumAllByOne(3);
2) return 3 + sumAllByOne(2);
3) return 2 + sumAllByOne(1);
```

4) sumAllByOne(2) can’t return a value right away as well, so it calls itself as a well-behaved recursion and the call is placed on top of the one who called it:
```
2) return 3 + sumAllByOne(2);
3) return 2 + sumAllByOne(1);
```

5) Finally when we call sumAllByOne(1), the base case is met (n == 1) and the topmost layer can return to the one below and be deleted from stack (its mission is accomplished):
```c++
4) return 1;
5) return 2 + sumAllByOne(1);
6) return 3 + sumAllByOne(2);
7) return 4 + sumAllByOne(3);
```
6) 

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>return 2 + 1;</td>
<td>call #4 returned 1, so this one can return 3 to #2</td>
</tr>
<tr>
<td>2</td>
<td>return 3 + sumAllByOne(2);</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>return 4 + sumAllByOne(3);</td>
<td></td>
</tr>
</tbody>
</table>

7) 

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>return 3 + 3;</td>
<td>call #3 returned 3, so this one can return 6 to #1</td>
</tr>
<tr>
<td>1</td>
<td>return 4 + sumAllByOne(4);</td>
<td></td>
</tr>
</tbody>
</table>

8) Finally, there is just one layer left and function can return 1 back to the caller from main():

<table>
<thead>
<tr>
<th></th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>return 4 + 6;</td>
<td>call #2 returned 6, so can return 10 to main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>finally (4 + 3 + 2 + 1 = 10), exactly what we wanted)</td>
</tr>
</tbody>
</table>