Understanding the problem before coding it

Marcus Birkenkrahe

April 13, 2024

Some of you had trouble with the loop exercise to compute the factorial n! for any integer n > 0, `n!  = n * (n-1) * ...  * 1`, and `0!=1` by definition.

How should you approach this problem as a coding problem? The pseudocode is already the result of your understanding - you can put it straight into code it it's correct. What if you don't know where to start?

Follow these steps:

1. What are the input and the associated variables and constants?
2. What are the output and the associated variables?
3. What is the output for a few sample values?
4. If loops are involved, how many?
5. What type of loop? Write the loop and print the loop variable.
6. What will the loop body do?

For the factorial problem, the answers are:

1. Input: integer `n >= 0`, constant `Null` = 0! = 1
2. Output: factorial integer `fac` = n * (n-1) * ... * 1
3. Examples:

```
0! = 1 (definition)
1! = 1
2! = 2 * 1 = 2
3! = 3 * 2 * 1 = 6
4! = 4 * 3 * 2 * 1 = 24
etc.
```

4. From the example, we learn that one loop is sufficient: for `n=4`, the loop variable `i` runs through the values 1,2,3,4.

5. We have three loops to choose from. In this case, `for` was required, but it is also the best choice because we're counting up (or down) in whole numbers:

```
for ( int i = 1; i <= 4; i++) printf("%d ",i); // for n = 4
puts("");
for ( int i = 4; i >  0; i--) printf("%d ",i); // for n = 4
```

6. In the loop body, we compute the factorial for each value of `i`. This is best illustrated with a table. Let's take the case `n=4` and count down from 4. The table shows:

   - Value of the loop variable
   - Test and (Boolean) test result
   - Factorial computation, where we assign the value to `fac`

Loop variable `i`	Test	Factorial `fac`	Lesson
4	`4 > 0 == TRUE`	`fac * 4 => fac`	Initialize `fac` as `1`
3	`3 > 0 == TRUE`	`fac * 3 => fac`	
2	`2 > 0 == TRUE`	`fac * 2 => fac`	
1	`1 > 0 == TRUE`	`fac * 1 => fac`	
0	`0 > 0 == FALSE`		Exit loop

   So at the end, `i=0` and `fac = 4 * 3 * 2 * 1`. The pattern for the loop body is clear, too: there is only one computation, `fac *= i` (short for `fac = fac * i`).

You do not always need to write things out, especially if you already coded something in another language, but it really helps, especially the table that shows what happens in every iteration.