CSE 331: Software Design & Implementation Winter 2026

Homework 1

Due: Wednesday, January 14th, 11:59pm

Task 1 – Every Log Has Its Day [8 pts]

We plan to provide the following method:

```
/** Calculates the integer, base-3 logarithm of n.
 * ...
 * @return the integer k such that 3^k <= n < 3^(k+1)
 */
public static int log3(int n);
```

a) When n is a power of 3, the integer k from the spec will satisfy $3^k = n$. Taking logarithms on both sides, we see that $k = \log_3(n)$. Now consider the case where n is not a power of 3. Explain what value is returned by the method in this case by relating it to $\log_3(n)$ (which is not an integer).

b) Keeping in mind your answer to (a), what is an another, equally reasonable but incomparable, way to specify this function? Your answer should leave the precondition unchanged and should only change the postcondition when n is *not* a power of 3. Write your answer in two ways: once in a similar format to how you answered part (a), and once as a drop-in replacement for the `@return` section of the original specification (using similar formulas).

c) This specification precisely defines the return value for positive `n`, but it does not make sense when `n` is not positive. Give **two** distinct ways of turning this into a specification that fully defines behavior for all integer inputs, now including non-positive values (i.e. each spec must reasonably handle all possible integers).

Your two specifications must still return the value described in the **original** specification when `n` is positive but they should be incomparable with each other. Format your specifications using 331-style Javadoc tags.

d) Your two specifications from part (c) are incomparable. Give **another** distinct specification that is **weaker** than **both** of your specifications from part (c).

Again, it must still return the value described in the original specification when `n` is positive.

Task 2 – The New Formal [9 pts]

In this problem, we will formalize the following English description: "the function `mult` multiplies together the values in a given list".

a) Calculate the value of mult on $3 :: 5 :: 2 ::$ nil and each shorter list: $5 :: 2 ::$ nil, and $2 ::$ nil.

b) Looking at these examples, describe how we can calculate mult$(x :: L)$ in terms of x and mult(L)? Show how your description works on $3 :: 5 :: 2 ::$ nil.

c) It is not clear from the English that mult(nil) makes any sense. Suppose that we want to support empty lists (as opposed to disallowing empty lists, which is another reasonable option).

How do we define mult(nil) so that the recursive pattern you identified in part (b) applies to lists containing one element also?

d) Write a formal definition of mult, both its type and definition, using our mathematical notation.

Task 3 – New Year, New Me [6 pts]

We plan to provide the following method. A `Resolution` is an object containing an attribute of type `Theme` (and other attributes).

```
/**
* Returns the count of the number of resolutions of a specified theme.
* ...
*/
public static int countResolutions(Resolution[] myResos, Theme wantedTheme);
```

To do so, we need to fill in the rest of the specification.

We are considering the following alternatives:

```
@requires myResos is not null                                          // Spec A
@throws NullPointerException if wantedTheme is null
@return the count of items in myResos matching the wantedTheme

@requires myResos is not null                                          // Spec B
@return the count of items in myResos matching the wantedTheme
        or -1 if wantedTheme is null

@requires myResos is not null and wantedTheme is not null              // Spec C
@return the count of items in myResos matching the wantedTheme
```

In the following questions, decide how the two specs compare, and explain your reasoning briefly in 1-2 sentences.

a) Specs **A** and **B**

b) Specs **A** and **C**

c) Specs **B** and **C**

Task 4 – Extra Credit: The Log Ate My Homework [0 pts]

The original specification in Task 1 is *declarative*. Write an imperative specification for positive n using our mathematical notation (inside the Javadoc `@return`). Do not worry about efficiency.

Assume that integer addition, multiplication, and comparison operators are already defined.

Hint: Define a function with two parameters, where the first parameter is n and the second parameter is used to count upward until we find the correct value of k. You can then specify the return value, in the `@return`, as a call to your function with appropriate choices for its two parameter values. For this problem, you can assume that our math notation includes exponentiation when the base is 3 (e.g., 3^n).

Task 5 – Optional: A-I, Captain [0 pts]

In this problem, we will use AI to implement the `log3` method from Task 1 with different specifications.

a) Which AI are you using?

b) Prompt the AI to fill in the body of `log3` for your incomparable specifications from Task 1(c). Show the code that it produces.

c) Now, prompt it using your weaker specification from Task 1(d). Which code does it give you? Is that guaranteed to be the one that you wanted?

d) Change back to one of your specifications from Task 1(c), but now, change the postcondition for positive values of `n` to your alternative, incomparable `@return` from Task 1(b).

Prompt the AI to fill in the body. Show the code that it produces.

e) Now, change the `@return` to this less formal, English specification:

```
@return the base-3 logarithm of n
```

Prompt the AI to fill in this method body. Which code does it give you? Is that guaranteed to be the one that you wanted?