Post-Mortem

Assignment 02

February 4, 2019

We normally publish the post-mortem for an assignment after it has been marked and released. Here is a list of common errors provided by the graders for assignment 2.

Style and Spacing

- Constants (and helper functions) should be defined above the design recipe for the function they are used in. A few students defined their constants and helper functions between their examples and function definition, or after the main functions specified in the assignment.
- Helper functions require their own purpose, contract, and examples.
- Lines should be less than 80 characters long: excessively long lines should be broken up into multiple shorter lines, and make use of DrRacket's auto-indenting features.
- Some students did not lay out their `cond` question/answer pairs consistently, and had a varying amount of question/answer pairs on each line.
- If the `cond` question/answer pair does not fit in one line and causes the code to look too vertical, students are advised to put the answer component on a new line.
- Some students forgot to leave a blank line before and after function definitions, or included both the function header and part of the function body on the same line.

General

- Remember to check basic test results after submitting to catch simple (but impactful) errors.
- Some students code did not run at all, hence they lost all correctness marks. You are advised to make sure your code runs.
- Some students used a different function name in their `check-expect` tests. It is highly recommended to ensure you are testing the correct code.
- Many students still had black highlighting present in their code, which is a clear indictaion of a lack of thorough testing. Students are advised to - as an absolute minimum - ensure there is no black highlighting in their code once it has been run.
- Many students used parameter names that were not meaningful (e.g., `x`, `y`, `z`) or used ambiguous/unclear names (e.g., `stand`, `prem`). It is better to have names that are long and clear than names that are short and cryptic.
- The names of helper functions should also be clear and descriptive. For example, `helper`, `test`, `function`, and `compute` are very poor choices for helper function names.

- For now, `equal?` should only be used to compare two values of unknown types, or values that can take on more than one type. When the types of the arguments are known, use the most appropriate comparison function (e.g., `symbol=?`).
- To determine if a boolean value is true or false, use that value directly. Expressions like `(boolean=? param true)` or `(equal? param false)` are not recommended; it is advised to use that value directly.
- Starting a new `cond` expression in an `else` clause is unneeded. Instead, directly check for the next condition in the original `cond` expression.
- Symbol values should not be defined as constants. As stated in the course notes, they are self-documenting, and defining them as constants does not provide any additional meaning.
- As specified in the assignment preamble, any symbol values should exactly match the description in the questions. Some students used symbols that differed slightly from those written in the assignment (such as using `'fallingquickly` instead of `'falling-quickly`, and thus lost some correctness marks.

Design Recipe

- Purposes, contracts, and function definitions should not be explicitly labelled.
- Purposes should begin with a function header (e.g., `(func-name param1 param2 param3)`). These headers should include each of the parameter names used in the function.
- Purposes should meaningfully use each parameter name in the description of the function, and these references to the parameter names should be written **exactly** as they appear in the function header.
- Contracts should begin with `func-name:` .
- If restrictions are already implied in a data type, they do not need to be included in the `requires:` section. For example, if a function consumes a `Nat`, there is no need to specify that the consumed `Nat` should be greater than or equal to 0.
- Many students forgot to include the requirements depicted in the question in their `requires` section.
- Requirements are not needed for the values and types that functions produce.

Question 1

- Many students did not define constants for the points gained for each line.
- Many students did not ensure that each conditional clause has been tested.

Question 2

- A few students enumerated every possible input for 2a, which defeated the point of the question.
- Some students did not correctly specify the allowed symbols in the `requires` section for both parts a and b. Remember that contracts should be as specific as possible.
- Many students used `(boolean=? raining? true)` in the question part of their `cond`, which is redundant.
- Purposes should only describe what the function does and not how it does it (i.e. they should not specify that the function was implemented using only cond/boolean expressions).

Question 3

- Many students did not define a sufficient amount of constants and there were many magic numbers in the function.
- It is highly recommended to use helper functions to avoid duplicate code.