Post-Mortem

Assignment 02

February 5, 2020

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. Many students defined their constants and helper functions between their examples and function definition, or after the main functions specified in the assignment. Students are advised to refer to pages 18-19 of the style guide for a sample submission that includes constant and helper function definitions.
- 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 horizontal, students are advised to put the answer component on a new line.
- Many 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.
- 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.
- Some students used parameter names that were not meaningful (e.g., `x`, `y`, `z`) or used ambiguous/unclear names (e.g., `number`, `str`). 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., `string=?`).
- To determine if a boolean value is true or false, use that value directly. Expressions like `(boolean=? inside-pocket? true)` or `(equal? inside-pocket? 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.

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:` (including the colon).
- 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.
- Some students forgot to capitalize the type names in their contract (e.g. using `nat` instead of `Nat`).

Question 1

- Many students have incorrect purpose statements. Please refer to style guide for details.

Question 2

- Many students defined three different helper functions to convert three parameters to numbers. Duplicated helper functions, which are doing exactly the same work, are not encouraged.

Question 3

- Purposes should only describe what the function does and not how it does it.
- Many students did not define any constants at all. While the use of helper functions may offset some of the need for constants, the nature of the question warrants at least 4-5 constant definitions in addition to having helper functions.
- Quite a few students didn't use any helper functions in their code, resulting in a long and unneccesarily complex function.

- Quite a few students had very unclear (or incorrect purposes). Students should ensure their purposes clearly state what the function does.
- Many students had an incorrect contract for Q3b - either there were one too many paramters or the return type was incorrectly stated as `Bool` instead of `Sym`. Please look over your contracts one more time before submitting.

Question 4

- Many students had an insufficient number of examples, students are advised to have 2-3 examples for each function.
- Many students had an insufficient number of tests. Every possible corner and boundary case should be tested.
- Many students used nested conds, which is discouraged. No mark was deducted for that.
- Although a lot of the design recipe errors from Q2 carried forward here, this question was relatively well done overall.