Post-Mortem

Assignment 04

February 18, 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 4.

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.
- 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.
- 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.

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 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.
- The names of helper functions should also be clear and descriptive. For example, `helper-loc`, `function`, and `p` are very poor choices for helper function names.

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.

- 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

- Some students did not provide a correct contract for sum-even. Either `(listof Int)` → `Int` or `(listof Num)` → `Int` is fine. `(listof Nat)` → `Nat` is not enough for this question, since an even number does not have to be a natural number. If you choose `(listof Num)` → `Int` as your contract, you should remember to use `integer?` to check if your input is an integer before you call `even?`. As illustrated in tutorial 5, `even?` errors if the input is not an integer.

Question 2

- This question is very well-done.

Question 3

- Many students did not use `divisor` as a helper function in part b.

Question 4

- Some students directly used `string-length` for part a, which is technically not allowed (mentioned on Piazza). We will give marks for that this time, but starting from A05, students are responsible to know all additional requirements and information posted on Piazza.