ASSIGNMENT II
Refactoring

Check Canvas Assignment Posting for Due Date

1. DESIGN ISSUES

Some change requests for the "Eval Sorts" program are described below. Answer the accompanying questions, applying what you have learned about design issues, such as separation of concerns, information hiding, and design for change.

1.1 Suppose that two programmers want to rewrite the "eval sort". One programmer will implement the sort algorithms and write the code that runs the sorts and produces the report. The other programmer will deal with inputting data from the source and preparing it for the code that the first programmer writes. Unfortunately the source code control system available to them prevents them from both working in the same source file simultaneously.

Identify aspects of the current 'design' which will make their work difficult. Are the issues primarily related to information hiding or separation of concerns? How would you recommend they change the class structure of the program?

1.2 We need to change the "eval sort" in a way that it can be used as a part of other programs. (For example, create a library function that, based on some parameters, prepares the data, runs the sorts, and returns the results to the calling program. Please describe what design issue arises in this situation.

2. REFACTORING

This is a real programming task; you are required to turn in refactored code as well as narrative answers.

2.1 Identify at least three different "code smalls" in Eval Sorts". Name each smell and identify the lines that exemplify the smell.
For each smell identified, make an argument for whether fixing it is primarily a matter of abstraction, information hiding, or separation of concerns.

2.2 Our customer wants to have the reported results sorted in order from fastest to slowest.

a. Identify the refactoring steps you would do to make this change. Hint: there must be at least two: one to hide certain information and a one related to separation of concerns.

b. Refactor the code. Make your changes stepwise. Change one design decision per step. Make only the changes needed to meet the requirement. Capture and submit a labelled snapshot of the code at the end of each refactoring step.

Recommended reading:

Martin Fowler, Kent Beck (1999) Refactoring: Improving the Design of Existing Code, Chapter 3: Bad Smells in Code
http://proquest.safaribooksonline.com.proxy.lib.iastate.edu/book/software-engineering-and-development/refactoring/0201485672/chapter-3dot-bad-smells-in-code/ch03 (Available on-line via the libraries Safari subscription.)