EECE 437 - Software Architecture and Design
Code Review

March 2, 2017

Code reviewed: Malak Mazeh
Reviewed by: Dana Abou Ali

1 Basic contents

1.1 Basic numeric information:

Lines of code	164 (including white space and comments)
Ratio comments to code	$\frac{14}{164}$ or 8.5%
Number of classes	17 classes
Number of files	1 file
Maximum level of inheritance	2 (Classes inherit from the Base class)
Readability	8.75/10
Consistency	6.5/10

1.2 Comments

- **Readability:** The code reading went relatively smoothly. In general, I was able to understand her code quickly and semi-effortlessly
- **Consistency:** The code is consistent in names whether its classes, functions or variables. But there seems to be some inconsistencies in the use of braces. I also found the classes hierarchy to be somewhat inconsistent (Boolean operators inherit from operator classes while arithmetic operators inherit from the Base class), but this could simply be part of the design.

1.3 Strengths of the repo

- Readability: As mentioned in the **Comments** section, the code was readable. This was mostly due to the following:
    1. The use of explicit function names that define the role of each function (evaluate(), set()...)
    2. The use of spacing to make the code less crowded hence easier to look at and read. (The code is not crowded)
    3. The use of comments to define what each class does
    4. The simplicity of the code (use of 'if' instead of ?: for example)
- Simplicity: The simplicity of the code and the logic behind it can be considered as a strength of its own. In addition to making the code clearer, the simplicity makes the code very easy to understand
- Consistency: The consistency in the names of:
    - Classes (First letter capital, the rest lower case. All classes names start by TA the the type)
    - Functions (all lower case)
    - Variables (one lower case letter)

1.4 Deviations from good practice

The code deviated from good practice in the following ways:

- Lack of consistency in the use of braces (Sometimes, there are braces to enclose single statements while other times there aren't)
- In my opinion, the number of comments is relatively low. This is probably because, as stated in the **Strengths of the Repo** section, the code is very easy to understand hence there was no necessity for comments.
- The different units of code (classes) are not separated into different files. But, in this case, because the classes are small (around 3-4 lines each), it's easier for both the reader and the writer to have them all in the same file. This is probably why she didn't separate her classes into different .cpp and .h files.

1.5 Advice

The style sheet available on Moodle contains some advice that can help the author have a better repository:

1. After opening a brace, directly close it and write code in between (This can help with forgotten braces like in Base class or in the TAPlus class)

2. Give meaningful variables names (for example intvalue instead of i)
3. Use incremental compilation (compile (and possibly run) after writing a new function/class/block...). This can help to avoid a huge code that doesn't compile
4. Be more consistent when using braces; always use braces to include code, even if its just one statement (Sometimes there's braces to include one line, sometimes there isn't)
5. Include a ReadMe file to explain how to read the code, important decisions taken ect..

2 Code Review

2.1 On form

- The name of the project (EECE-437-Ass-1) does not reflect the functionality of the project
- The checked in items conform to the requirement: Only a cpp file was checked in; no generated items
- As stated in the **Deviations from good practice** subsection, the different units of code were not separated
- Class names, variable names and function names were chosen consistently. Both class names and function names reflect the functionality or structure. For variables, the names don't really reflect the role of the variables (i instead of integer for example). But, because she uses only three variables (integer i, double d and boolean b) this isn't a major issue.

2.2 On operation

The code did not compile.

2.3 On content

- There is no explicitly stated separation of classes, or English rewriting of the requirements. But I noticed that boolean operators were separated into unary and binary operators (Operators inherit either from operation1 class (unary) or operation2 class (binary)) even thought I wasn't able to find these classes.
- The functionality is reflected in the evaluate method which was implemented in all operator classes. The constructor's headers are defined, but most of them don't have a body

- The submission does not implement constants. When it comes to arithmetic operators, the author implemented two versions one for doubles and one for integers, probably to resolve the problem of return types. To forbid mixing types in operations, she created an enum called type, assigned a type for each primitive data (integer, boolean, doub) and raised runtime errors when types were mixed. Finally, the use of a Base class from which all other classes derive was probably meant to be used to implement pairs and arrays, but I can't know for sure as these parts weren't implemented

2.4 Questions to answer

- The submission does not put the problem into context. It takes into account the provided use case (as seen in the class main), but doesn't propose additional usages or services.

- The submission does not state the principle design decisions, but some of them can be deduced from the code:

  - All classes seem to inherit from a Base class. This could help in the implementation of arrays and pairs, as specified in the **On content** section.
  - All boolean operators seem to either inherit from operation1 class (unary operator) or operation2 class (binary operator) as seen in the class definitions. I was not able to find these classes
  - To support having different arithmetic operators (addition of integers returns an integer while addition of doubles returns a double for example), she seems to have implemented two versions of each class-one for doubles and one for integers.
  - To forbid mixing types in operations, each class has a type value which are compared and output an error message when different (similar to runtime errors)