Assignment 1

COMP SCI 2ME3 and SFWR ENG 2AA4

1 Dates and Deadlines

Assigned: January 11, 2021

Part 1: January 21, 2021

Receive Partner Files: January 26, 2021

Part 2: January 28, 2021

Last Revised: January 15, 2021

All submissions are made through git, using your own repo located at:

`https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2021/[macid].git`

where `[macid]` should be replaced with your actual macid. The time for all deadlines is 11:59 pm. If you notice problems in your Part 1 `*.py` files after the deadline, you should fix the problems and discuss them in your Part 2 report. However, the code files submitted for the Part 1 deadline will be the ones graded.

2 Introduction

The purpose of Part 1 of this software design exercise is to write Python modules that matches the natural language specification provided. In Part 2 you will critique the design, based partially on your experience working with a portion of one of your classmate's implementations. Part 2 will also include answering questions related to the course content.

All of your code, except for your test driver (Step 3), should be documented using doxygen. All of your reports should be written using LaTeX. Your code should follow the given specification. In particular, you should not add public methods that are not specified and you should not change the number or order of parameters for methods or change the types of the returned values.

Please remember to commit to GitLab frequently. **A portion of your grade will be based on you demonstrating a reasonable commit history.** Frequent small commits are a good habit to form, and this practice might save you if something goes wrong with your local machine. As mentioned at the end of the assignment, you will build additional confidence in your implementation if you test it on mills.

Part 1

In this part of the assignment you will implement two modules in Python, each in a separate file. You will also write a Python file for your test driver.

Step 1

Write a module in Python that implements an Abstract Data Type (ADT) for complex numbers (`ComplexT`). The code should be in a file named `complex_adt.py`. The module will define several methods (access programs) as defined below. To find the necessary mathematics, please refer to the Wikipedia page on complex numbers.

- A constructor (`ComplexT`) that takes two floats x and y as input and creates a `ComplexT` object. x is the real part of the complex number and y is the imaginary part. See the Notes at the end of the assignment specification about implementing constructors in Python.
- A getter named `real` that returns the real part (x) of the complex number $z = x+yi$.
- A getter named `imag` that returns the imaginary part (y) of the complex number $z = x + yi$.
- A getter named `get_r` that returns the absolute value (or modulus or magnitude) of the complex number $z = x + yi$.
- A getter named `get_phi` that returns the argument (or phase) of the complex number $z = x + yi$. The phase should be returned in radians. See the Notes at the end of the assignment specification about assumptions and exceptions.
- A method named `equal` that takes an argument of `ComplexT` and returns a Boolean. The result is True if the argument and the current object are equal and False otherwise. See the Notes at the end of the assignment specification for details on the optional step of including a “magic” method for equality.

- A method named `conj` that takes no argument and returns a `ComplexT`. The result is the complex conjugate of the current object.
- A method named `add` that takes an argument of `ComplexT` and returns a `ComplexT` object that adds the argument to the current object.
- A method named `sub` that takes an argument of `ComplexT` and returns a `ComplexT` object that subtracts the argument from the current object.
- A method named `mult` that takes an argument of `ComplexT` and returns a `ComplexT` object that multiplies the argument and the current object.
- A method named `recip` that takes no argument and returns a `ComplexT`. The result is the reciprocal of the current object.
- A method named `div` that takes an argument of `ComplexT` and returns a `ComplexT` object that divides the current object by the argument.
- A method named `sqrt` that takes no argument and returns a `ComplexT`. The result is the positive square root of the current object.

Step 2

Write a second Python module, named `triangle_adt.py`, that implements an ADT for triangles (`TriangleT`). The module will define several methods (access programs) as defined below:

- A constructor (`TriangleT`) that takes three arguments, each of type integer, and constructs an object of type `TriangleT`. The arguments are the lengths of the sides of the triangle.
- A getter named `get_sides` that return a tuple of three integers, where each integer is the length of one side of the triangle.
- A method named `equal` that takes one argument of type `TriangleT` and returns True if the current `TriangleT` is equal to the argument. Otherwise False is returned.
- A method named `perim` that takes no arguments and returns an integer representing the perimeter of the current `TriangleT`.
- A method named `area` that takes no arguments and returns a float representing the area of the current `TriangleT`.

- A method named `is_valid` that takes no arguments and returns a Boolean. The returned value is True if the three sides in the current `TriangleT` form a valid triangle. Otherwise False is returned.
- A method named `tri_type` that takes no arguments and returns a `TriType`. `TriType` is an element of the set {`equilat, isosceles, scalene, right`}. The elements of `TriType` correspond to the triangle types equilateral, isosceles, scalene and right angle. The class `TriType` should be defined in the `triangle_adt.py` file. Details on how to define an enumerated type are given in the Notes section at the end of the assignment specification.

Step 3

Write a third module that tests the first and second modules. It should be a Python file named `test_driver.py`. Your initial git repo contains a `Makefile` (provided for you) with a rule `test` that runs your `test_driver` source with the Python interpreter. Each function should be tested as completely as you are able. A specific number of tests is not prescribed, nor is an approach for devising test cases. At this point you should use your intuition and best judgement of the tests that will build confidence in the correctness of your implementation. (Throughout the course we will return the question of testing and potentially be adding to and refining your intuition.) Please note for yourself the rationale for test case selection and the results of testing. You will need this information when writing your report in Step 7. The requirements for testing are deliberately vague; at this time, we are most interested in your ideas and intuition for how to build and execute your test suite.

Your test driver should be as automated. A test case consists of an expected answer and a calculated answer. You should write code to compare your calculated results to the expected results. A test case passes when they match. Otherwise the test fails. Please avoid manual tests where the output is simply printed to the screen, with the expectation that a human user will check it. A manual approach like this is simply not maintainable over time. To get you started, some sample tests are given in `test_expt`, along with a make rule (`make expt`) for running the sample tests.

You are not required to use a unit testing framework, but you are welcome to use `pytest`.

Step 4

Test the supplied `Makefile` rule for `doc`. This rule should compile your documentation into an html and LaTeX version. Your documentation should be generated to the `A1` folder. Along with the supplied `Makefile`, a doxygen configuration file (`docConfig`) is also given in your initial repo.

Step 5

Submit (add, commit and push) all code files using git. (Of course, you will be doing this throughout the development process. This step is to explicitly remind you that the version that will be graded is the one we see in the repo.) Please **do not change** the names and locations for the files already given in your git project repo. For Part 1, the only files that you should modify are the Python files. Changing other files could result in a serious grading penalty, since the TAs might not be able to run your code and documentation generation. The only exception to this would be modifying the `Makefile` should you choose to use `pytest`. You should NOT submit your generated documentation (html and latex folders). In general, files that can be regenerated are not put under version control.

After the deadline for submitting your solution has passed, your partner files, `complex_adt.py` and `triangle_adt.py`, will be pushed to your repo.

Part 2

Step 6

After you have received your partner's files temporarily replace your corresponding files with your partner's. (Do not push your partner's code to your repo; your code should still be what appears in the repo. The replacement of the code is simply for testing purposes.) Do not initially make any modifications to any of the code. Run your test driver and record the results. Your evaluation for this step does not depend on the quality of your partner's code, but only on your discussion of the testing results. If the tests fail, for the purposes of understanding what happened, you are allowed to modify your partner's code.

Step 7

Write a report using LaTeX (`report.tex`) following the template given in your repo. The elements that you need to fill in include the following:

1. Your name and macid.
2. Your code files.
3. Your partner's code file.
4. List assumptions you made and exceptions you added. Assumptions are conditions on the input that you assume will always apply. Exceptions are Python exceptions you have added for when an unexpected situation arises.
5. Summary of your test cases and the rationale for their selection.
6. The results of testing your files combined with your partner's files.
7. A critique of the design. What did you like? What areas need improvement? How would you propose changing the design?
8. Answers to the following questions [to be completed]:
   (a) Which of the methods for the classes ComplexT and TriangleT are mutators (setters)? Which methods are selectors (getters)?
   (b) The specification did not specify state variables for either ADT. As long as you achieve the specified behaviour, you can determine the state variables. What are at least two options for the state variables for `ComplexT` and for `TriangleT`? You do not need to worry whether the options you list are good from an implementation point of view, just that they are possible.
   (c) The class `ComplexT` has an equal method. Would it make sense to also add a methods for greater than and less than? Why or why not?
   (d) Is it possible that the three integers input to the constructor for `TriangleT` will not form a geometrically valid triangle? What should the class `TriangleT` do in the case where the constructor is given an invalid triangle? Please justify your answer.
   (e) the `TriangleT` class could have a state variable for the type of triangle. Is this a good or a bad idea? Why?
   (f) (Course Content Question) What is the relationship between the software qualities of performance and usability?

(g) (Course Content Question) Are there situations where it is not really necessary to “fake” a rational design process?

(h) (Course Content Question) How might reusability affect the reliability of products?

(i) (Course Content Question) What are some examples of how programming languages are abstractions built on top of hardware?

The writing style for the report should be professional, but writing in the first person is fine. Some of your ideas can be summarized in lists, but most of the report should be written in full sentences. Spelling and grammar is important and will be graded.

Commit and push `report.tex` and `report.pdf`. Although the pdf file is a generated file, for the purpose of helping the TAs, we'll make an exception to the general rule of avoiding version control for generated files. If you have made any changes to your Python files since pushing Part 1, you should also push the new changes.

Including code in your report is made easier by the `listings` package: https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings.

Linking to the original code in the repo is also helpful via the hyperref package: https://www.sharelatex.com/learn/Hyperlinks.

Notes

1. Your git repo will be organized with the following directories at the top level: `A1`, `A2`, `A3`, and `A4`. Inside the `A1` folder you will start with initial stubs of the files and folders that you need to use. Code files will be in the `src` folder, while the report will be in the `report` folder. Please do not change the names or locations of any of these files or folders.

2. Please use the following doxygen components at the start of all Python files `@file`, `@author`, `@brief` and `@date`.

3. Your program must work in the ITB labs on mills when compiled with its versions of Python (version 3), LaTeX, doxygen and make. Python is called via python3 or python on mills.

4. You are free, **in fact you are encouraged**, to use any existing Python libraries for your implementation. However, only use libraries that are already installed on mills.

5. The specification is for the external interface for the classes, not for their internal implementation. Externally the constructor is the class name, even though internally you will use (`__init__`). Externally a method's arguments do not include `self`, but internally this will be your first argument.

6. If you feel that you need private methods, please use the Python convention of "dunder" names with double underscores (`__methodName__`).

7. When importing modules in Python, you should not include the path to the library in the import statement. All you will need is the filename.

8. If completing the assignment requires making any assumptions, or adding exceptions, please document this. Exceptions are documented with `@throws`.

9. The specification is for the external interface of the objects. That is, the specification is how other programs would access the services of these modules, not how the modules will be implemented.

10. For types that are defined as a set of potential values (like `TriType`), you should use an enumerated type (https://docs.python.org/3/library/enum.html).

11. The equal method is required in the assignment. This method should be implemented as named. Optionally, you can also include a the private method `__eq__`. (The equal method is required regardless of whether you define the "magic" method.)

12. Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes. The diffs for the commits of the changes provide a convenient way to see the specific modifications to the assignment specification.