Assignment 1

COMP SCI 2ME3 and SFWR ENG 2AA4

1 Dates and Deadlines

Assigned: January 7, 2019

Part 1: January 18, 2019

Receive Partner Files: January 23, 2019

Part 2: January 25, 2019

Last Revised: January 18, 2019

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

```
https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2019/[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 a Python program 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.

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 as closely as possible. In particular, you should not change the number

or order of parameters for functions or change the types of the returned values.If you feel a need to add additional functions in your implementation, you can do so.

Please remember to commit to GitLab frequently. This is a good habit to form, and it 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

The specified modules represent a portion of a program written to allocate engineering students from first year into their second year programs. One module will read the relevant data from files and the other will perform operations on the data. Each of the modules will be written as a separate Python file. You will also write a Python file for your test driver.

Step 1

Write a first module that reads the necessary data. It should consist of Python code in a file named `ReadAllocationData.py`. The module will define several functions as defined below:

- A function named `readStdnts(s)` that takes a string `s` corresponding to a filename and returns a list of dictionaries of student information. Each dictionary in the list corresponds to one student. The dictionary for each student will have the following form: `{'macid': string, 'fname': string, 'lname': string, 'gender': string, 'gpa': float, 'choices': [string, string, string] }`. You are free to determine the format of the text file that stores this information. The fields in the dictionary correspond respectively to the following concepts: the student's McMaster id (not the student number, but the short (unique) string based on their name), their first name, their last name, their gender (either `'male'` or `'female'`) their grade point average (on a scale from 1 to 12), and their preferred choices for second year program. The choices are represented by a three entry list of strings, where each string corresponds to the type of engineering they would prefer to study. The seven options are (`'civil'`, `'chemical'`, `'electrical'`, `'mechanical'`, `'software'`, `'materials'`, `'engphys'`). The three entries for each student are listed in order of the student's preference.

- A function named `readFreeChoice(s)` that takes a string `s` corresponding to a filename and returns a list of strings, where each entry in the list corresponds to

the macid (string) of a student that has been granted free choice. (Free choice is a status that is granted to incoming Engineering students that have a sufficiently high highschool average.) The students on this list are automatically allocated to their first choice of program, as long as they are eligible. (The allocation algorithm is responsible for determining eligibility; the current function does not filter the incoming names in any way.) The information needed is in the test file that corresponds to the name given in the argument `s`. You are free to determine the format of this file.

- A function named `readDeptCapacity(s)` that takes a string `s` corresponding to a filename and returns a dictionary where the key value pairs are `{'dept': integer}`. The key string is from the list of seven departments listed (given above under function `readStdnts`) and the value is the capacity of the department. That is, the number of slots available in each department for student positions. The information needed is in the text file that corresponds to the name given by the argument `s`. Again, you are free to determine the format of this text file, as long as it includes each department and its capacity.

Step 2

Write a second module, named `CalcModule.py`, that completes various calculations on the list of students created using the functions in the previous module. You are free to use any existing Python libraries for your implementation.

- A function named `sort(S)` that takes one argument: `S` a list of the dictionaries of students (created by function `readStdnts` from above) and returns a list of student dictionaries in sorted order. The sort should be in descending order based on gpa. You can assume that once created the dictionary for each student will not be changed, so you do not need to worry about issues with aliasing.

- A function named `average(L, g)` that takes two arguments: `L` a list of the dictionaries created by function `readStdnts(s)` and `g` a string representing the gender (male or female). The function returns the average gpa of male or female students, with the choice depending on the value of `g`.

- A function named `allocate(S, F, C)` that takes three arguments: `S` a list of the dictionaries of students (created by function `readStdnts` from above), `F` a list of students with free choice (created by function `readFreeChoice`), and `C` a dictionary of department capacities (created by function `readDeptCapacity(s)`). This function returns a dictionary with the following format: `{'civil':[student, student,`

`...], 'chemical':[student, student, ...], 'electrical':[student, student, ...], 'mechanical':[student, student, ...], 'software':[student, student, ...], 'materials':[student, student, ...], 'engphys':[student, student, ...]}`. The "type" `student` corresponds to a student dictionary (following the format given in function `readStdnts`). The output corresponds to which students have been allocated to which departments. The algorithm for the allocation will allocate all students with a gpa greater than 4.0. Those with less than 4.0 will not be allocated. Students with free choice will be allocated to their first choice. All other students will be allocated in the order of gpa. Starting with the highest gpa, students are put into departments by their first choice, until the bin for a given department is full. From that point onward students that select that department are instead allocated to their second choice. If their second choice is full, then they are allocated to their third choice.

Step 3

Write a third module that tests the second module (`CalcModule.py`). It should be a Python file named `testCalc.py`. Your initial git repo contains a `Makefile` (provided for you) with a rule `test` that runs your `testCalc` 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 as possible. A test case consists of an expected answer and a calculated answer. You should compare your calculated results to the expected results and report that the 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 maintainble over time.

You are not required to use a unit testing framework (like pytest), but you are welcome to do so.

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) the files `ReadAllocationData.py`, `CalcModule.py`, and `testCalc.py` 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. You should also push any input data files you created for testing purposes. For Part 1, the only files that you should modify are the Python files and the only "new" files you should create are the input data 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. 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 file, `CalcModule.py`, will be pushed to your repo. **Including your name in your partner code files is optional.**

Part 2

Step 6

After you have received your partner's files, replace your corresponding files with your partner's. Do not initially make any modifications to any of the code. Run your test module 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. The results of testing your files (along with the rational for test case selection).
5. The results of testing your files combined with your partner's files.
6. A discussion of the test results and what you learned doing the exercise. Under the test results, you should list any assumptions you needed to make about the program's inputs or expected behaviour. List any problems you found with: (a) your program, (b) your partner's module
7. The specification of this assignment imposed design decisions on you, like the data structure for students, freechoice, and department capacities. Please provide a critiques of the design. What did you like? What areas need improvement? How would you propose changing the design?
8. Answers to the following questions:
   (a) How could you make function `average(L, g)` more general? That is, can you specify a similar function, but one that is more versatile/flexible than the given function? The new function should be capable of the identical behaviour as `average(L, g)`, but also have other capabilities. Along a similar line of thinking, how could you make the `sort(S)` more general?
   (b) The assignment states that you can assume that aliasing will not occur with the dictionaries in your lists. What does aliasing mean in this context? In general, could aliasing be a concern with dicts? How might you guard against potential problems?
   (c) The assignment did not require you to test the `ReadAllocationData.py` module. Describe some potential test cases you could have used to build confidence in this module. Of the two modules, why do you think `CalcModule.py` was selected over `ReadAllocationData.py` as the one you should test?

(d) The assignment uses strings as the keys in several dictionaries. It also uses strings to represent members of finite sets. For instance the strings `'male'` and `'female'` are used to represent elements of the set $\{\text{male}, \text{female}\}$. What are the problems with using strings in this way? What would be a better approach?

(e) A dictionary isn't the only option to implement records and structs in Python, where records and structs correspond to the mathematical notion of a tuple. What are other options for implementing the mathematical notion of tuples in Python? You may find the discussion at https://dbader.org/blog/records-structs-and-data-transfer-objects-in-python helpful in answering this question. Would you recommend changing the data structure used in the code modules? How would you change it?

(f) In the specification the student's preferred choices are listed as `'choices': [string, string, string]`. If the list of strings was changed to a different data structure, like a tuple, would the `CalcModule.py` module need to be modified? Consider instead if a custom class (Abstract Data Type (ADT)) was written for students, and the `CalcModule.py` module was modified accordingly. This custom class provides a method that returns the next choice and another method that returns `True` when there are no more choices. In this new case, if the data structure inside the custom class changes, say from lists to tuples, will the `CalcModule.py` module need to be modified? Please justify your answer.

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, you should also push those 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. If completing the assignment requires making any assumptions, or adding exceptions, please document this. Exceptions are documented with `@throws`.

5. 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.

6. As mentioned previously you do not need to worry about aliasing issues. The contents of dictionaries that are created are assumed to not be altered. Therefore, it is not necessary to make copies. References to the existing dictionaries is fine.

7. Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes.