ENSC 251 Project

Project Overview:

In the ENSC 251 project: you will design and implement a simplified graduate student admission system using the skills learned throughout this course, i.e., Object-Oriented Programming with C++. Hopefully you can learn more about the programming skills, as well as how to get into a graduate program. The project weighs 50 marks, which includes 20 marks for the interim report and 30 marks for the final presentation. All students need to form a 4-person team to work on the project and will be evaluated per team (including both individual marks and team marks). Some general grading logistics have been discussed in Lecture 0 and posted on our course website: https://canvas.sfu.ca/courses/70216/pages/project-logistics.

You can divide the work roughly based on the listed tasks; the overall marks for major tasks are also listed for your guidance, but not the detailed marks for every single item. Please include a simple document in the project report indicating which parts each member finished. There is some negative marking if you don't complete the listed tasks; details are not listed. Basically, the detailed grading scheme (except the overall marks for major tasks) for the project will NOT be released before your project grading is done. Think about you are in a real interview, nobody will tell you what the detailed grading schemes are.

In this simplified graduate student admission system, we will have a number of domestic and international students applying to SFU graduate school, each student will be evaluated based on his/her CGPA, research score, and optional English language testing scores (if the applicant is an international student). Only the top applicants who satisfy certain criteria will be admitted to SFU graduate school.

We make some simplifications here. First, for CGPA, we assume all student applicants will use SFU 4.3 CGPA scale (round to single digit after the decimal point, e.g., 4.33 to 4.3, 3.67 to 3.7). Second, we simplify the research score to an integer number from 0 to 100 (full score 100). Note in reality, research score is based on a number of factors, including the applicant's research experience, publication, recommendation letters and etc. Third, we assume all international students need to provide their TOEFL scores for the English language test.

In the project interim part, you have already built the basic components. Now let's finish implementing this system.

Your Tasks for Final Report (30 marks, Due Jul 28th, 2022):

For the final project report, you will have to implement the following tasks to further enhance this graduate admission system.

Part 1: Update to singly linked list based system (12 marks)

To support the feature of ***quickly inserting a new applicant or deleting an existing applicant***, we will use ***singly linked lists*** to store the DomesticStudent and InternationalStudent objects ***in a sorted order***. You will have to achieve the following goals:

1. [3 marks] Use one singly linked list to store all the DomesticStudent objects in a sorted order and another singly linked list to store all the InternationalStudent objects in a sorted order, which are read and initialized from the input files.
    a. Whenever you read one line of data from the input file, initialize a DomesticStudent (and InternationalStudent) object, and insert this object into the DomesticStudent (and InternationalStudent) singly linked list. You have to modify your DomesticStudent (and InternationalStudent) class to make it a linked list node type. Note the linked list node type should be a class instead of a struct in this project.
    b. Each time you insert a DomesticStudent (and InternationalStudent) object into the DomesticStudent (and InternationalStudent) singly linked list, you have to make sure that all objects in the singly linked list are sorted based on the overall sorting scheme (by all fields) that you implemented in the interim project portion. That is, your singly linked list is always sorted.
    c. You should maintain both a ***head*** pointer and a ***tail*** pointer that point to the head node and tail node of your DomesticStudent (and InternationalStudent) singly linked list.

2. [6.5 marks] Based on the user input, your program should be able to
    a. [1.5 marks] Search existing DomesticStudent (and InternationalStudent) objects in the DomesticStudent (and InternationalStudent) linked list based on the user input information "application id", or "cgpa", or "researchScore". Print out all objects which have the same application id, or cgpa, or researchScore, as the user input. If there is no match, print out information indicating there is no match found. Note each search here should just take one input (e.g., cgpa) and basically you should have three separate search functions for application id, cgpa, and researchScore.
    b. [1 mark] Search existing DomesticStudent (and InternationalStudent) objects in the DomesticStudent (and InternationalStudent) linked list based on the user input information "firstName and lastName". Print out all objects which have the same firstName and lastName (both matched) as the user input. If there is no match, print out information indicating there is no match found.
    c. [1 mark] Create a new DomesticStudent (and InternationalStudent) node based on the user input information, and insert this new node into the DomesticStudent (and InternationalStudent) singly linked list in order (using the overall sorting scheme in the interim project portion).
    d. [2 marks] Delete existing DomesticStudent (and InternationalStudent) objects in the DomesticStudent (and InternationalStudent) linked list based on the user input information "firstName and lastName". Delete all objects which have the same firstName and lastName (both matched) as the user input.
    e. [1 mark] Delete both the head node and tail node from the DomesticStudent (and InternationalStudent) linked list in a single delete function.

3. [2.5 marks] Based on the user input, merge the two sorted DomesticStudent and InternationalStudent linked lists into a single Student linked list, which is also sorted based on a modified overall sorting. And then print all the Student objects information. That is, you print out the information of all Student objects (except those InternationalStudent objects who didn't meet the TOEFL score requirement), which are sorted based on their CGPA and researchScore.

a. When you merge the two lists, the modified overall sorting is similar to the one used in the interim project portion, but slightly different. First, two students (can be either DomesticStudent or InternationalStudent) are sorted based on their research score. If two students have the same research score, then they are further sorted based on their CGPA. If two students further have the same CGPA, then they are merged in their original order if both are DomesticStudent, or both are InternationalStudent, and you always put a DomesticStudent ahead of an InternationalStudent.
b. Search existing Student objects in the merged linked list based on the user input information "cgpa_threshold and researchScore_threshold". Print out all DomesticStudent and InternationalStudent object information who have both a cgpa >= cgpa_threshold and a researhScore >= researchScore_threshold; basically, these are the students who will be admitted to SFU. **Here you should use polymorphism and virtual function for the print.** If there is no match, print out information indicating there is no match found.
c. To make your life easier, in your main program, assume you test item 2 (i.e., operations on each individual DomesticStudent or InternationalStudent list) first, and then test item 3 (i.e., operations on merged list). After the merge, you no longer have to maintain the original DomesticStudent and InternationalStudent lists.

4. **!!IMPORTANT NOTE!!** You have to write your own code for all the above tasks, no library function calls (e.g., sort, list/deque containers, etc.) are allowed.
5. **!!IMPORTANT NOTE!!** Make sure you maintain a low algorithm complexity for inserting into the linked lists in sorted order and merging two lists. The algorithm complexity also counts for marks.
6. You should maintain the modular code in different files and update your Makefile.

Part 2: More robust error checking (3 marks)

Some basic error checking should have already been added in your interim project code. Now let's add more to make your code more robust. You have to catch and handle the following errors. You are free to use regular if/else, switch/case statements, or try-throw-catch syntax to implement the error checking, unless otherwise specified.

1. If any input field of a Domestic (and International) student in the input file is missing, catch the error, print out a message to hint the user that one filed is missing, and exit the program.
    a. E.g., if one line of input in the domestic-stu.txt looks like this "Mary,White,4.00,85", you should catch the error, and report that one field is missing.
2. Make sure all string matching (e.g., search using firstName) in your program is case insensitive: e.g., "Mary", "mary", "maRY" should be treated the same. Note this applies to all cases including all the new error checking added in the final project portion.
3. Make sure every field of a Domestic (and International) student in the input file has a valid range. Specifically, please add the following checks

a. Province must be one of the following: NL, PE, NS, NB, QC, ON, MB, SK, AB, BC, YT, NT, NU. Otherwise, print out an error message that it's not a valid province and exit the program
b. Country should be one of the following (note this is specific to our input test case, in real life, you should include all valid country names): Canada, China, India, Iran, Korea. Note there is a specific typo in your input file, which your program should detect and automatically fix it: your program should detect if the country name is "Idian", print out a warning message telling the user it's a typo and your program automatically fixes it to "India". For all other mismatches, print out an error message that it's not a valid country and exit the program

4. Make sure when a user types in a selection option which is not available (e.g., some option other than insert, search, delete, and merge), print out a hint to user: you typed in an option which is not available, here are the available options with detailed explanation… Then continue asking the input from the user. Do NOT exit the program.
5. Use try-throw-catch to add robust error checking whenever you allocate new memory space.
6. Create an error input for each of the above case, and demo to your TA that your code can handle all the above errors.

Part 3: Unit Test (3.5 marks, each item 0.5 mark)

Please review the concepts of unit test in Lecture 7, and write unit tests to validate the following major functions that you wrote work correctly, including

1. Insert a DomesticStudent (and InternationalStudent) object into the DomesticStudent (and InternationalStudent) singly linked list in order.
2. Search existing DomesticStudent (and InternationalStudent) objects in the DomesticStudent (and InternationalStudent) linked list based on the user input information "application id", or "cgpa", or "researchScore".
3. Search existing DomesticStudent (and InternationalStudent) objects in the DomesticStudent (and InternationalStudent) linked list based on the user input information "firstName and lastName".
4. Delete existing DomesticStudent (and InternationalStudent) objects in the DomesticStudent (and InternationalStudent) linked list based on the user input information "firstName and lastName".
5. Delete both the head node and tail node from the DomesticStudent (and InternationalStudent) linked list in a single delete function.
6. Merge the two sorted DomesticStudent and InternationalStudent linked lists into a single Student linked list.
7. Search existing Student objects in the merged linked list based on the user input information "cgpa_threshold and researchScore_threshold".

Please make sure that your unit test covers: the "normal" cases, the boundary/corner cases, and the illegal cases. Add the unit test as an option into the user menu (i.e., in the printed menu, users can select to perform a unit test). Also make sure to print out the testing results in a meaningful way.

Part 4: Algorithm Complexity Analysis (1.5 marks)

Analyze the complexity of your insert, search, deletion, and merge functions (i.e., all those functions mentioned in Part 3) using the big-O notation.

Part 5: Your own innovation (5 marks)

Part 1 to 4 only count for 20 marks in total. We reserve 5 marks for your own innovation into this project. You may enhance the performance of your code, add new features, make it even more robust. Add anything new that your team wants, which is not already covered by the project description but adds more value.

Part 6: Slides presentation (5 marks)

We also reserve 5 marks for slides presentation during the last three lectures (Jul 29th, Aug 3rd, and Aug 5th lecture hours). Note this will be F2F in our classroom and it is in addition to your code demo to TAs in the lab. While your code demo to TAs focuses on the code itself, this slide presentation focuses on more high-level design and project management, including 1) how you design system features, 2) user interface, 3) core classes and functions, 4) your own innovation, 5) project management including weekly schedule and work partition, 6) lessons learned from the project, etc. Also, the slide presentation should cover the entire project from beginning to end, while your code demo to TAs only cover the second half of the project. Each project group has around 15 mins to present their slides and 2-3 mins for Q/A; I might ask questions during your presentation, not necessary at the end of your presentation.

Other notes:

Part 1 to Part 5 (25 marks) use the two-step individual and team marking. For Part 6 (5 marks), all group members will have the same marks.

In addition to the actual coding implementation, you need to provide good commenting, naming, and other good coding styles; all these count in your project marking.

For grading logistics and remote machine access, please refer to the course website. If you have any questions, please post them on the discussion board.

Final Report Submission:

There are two separate submission for code and slides, both of which will be submitted electronically through Canvas.

1. For the final project code, you will need to submit a single proj.zip file. Please include a simple pdf file in proj.zip indicating the task distribution among members, your own innovations, as well as big O analysis results. Failure to comply with this format could result in zero score on this project. To zip your files in Linux,

1) Go to your proj directory

2) make clean //make sure you clean your files
3) cd .. //go one level up
4) zip -r proj.zip proj //zip all your proj files into a single proj.zip

2. For the final slides, you will have to submit a single .pptx file. Note you can only present with the slides that you submitted.

Final Report Submission Deadline:

Your final project report is due at **11:59:59pm on Thursday, Jul 28th, 2022**. You need to meet the deadline: every 10 minutes late submission, you lose 2 marks; that is, 150 minutes late, you will get zero on this final project report.

Final Report Code Demonstration:

You will have to demo your final project to your TA **in the lab during last three lecture days (Jul 29th, Aug 3rd, and Aug 5th lecture hours); we will rotate groups for code demo and slides presentation on these three lecture days**. Only code from your Canvas submission is allowed in the lab demo. Each student group has around 20 minutes to explain your code to the TA. If you fail to do the demo (without a medical note), or if it is determined that you do not understand the code being evaluated, you will be awarded zero on final project report. Also please show up in the demo day on time (TA will send out your scheduled time), otherwise you will lose 0.5 mark.