COMP 2404 – Midterm Project

Due: Thursday, November 5 at 11:59 pm

1. Goal

For this project, you will write a C++ program to manage a book club with books and book club members. Those members will have the ability to view different kinds of information, and to rate the books that they read. Your code will be correctly separated into object design categories, and it will meet every principle of good software engineering that we have covered in class.

The grading of projects in this course will be rubric-based. The **mandatory minimum criteria** for grading is that all the requirements must be implemented, they must execute correctly, and sufficient datafill must be provided. Submissions that are incomplete, or that don't work correctly, or that don't provide sufficient datafill will earn a grade of zero.

2. Program Requirements

You will implement a program that manages the data for a book club that has multiple books and many members of the club (the people who read the books). The users of our program will be the book club members. They will be given the option of viewing the data, and they will be allowed to enter a rating (between 1 and 10) for a particular book that they read. They will also be able to print out the best rated book, based on the average of all ratings by all members, and the most rated book, based on the number of members who rated each book.

Your program will implement the following requirements:

2.1. The program will be separated into control, view, entity, and collection objects. The classes will include, at minimum:

- 2.1.1. one control class that drives the control flow
- 2.1.2. one view class that's responsible for most communication with the end user
- 2.1.3. at least five (5) different entity classes
  - (a) both books and club members must have a unique identifier that is automatically generated when they are created, as we did in the coding example of section 3.1, program #6; you must use inheritance to model this commonality
  - (b) all entity objects must be dynamically allocated
- 2.1.4. at least two (2) different collection classes
  - (a) one will be a dynamically allocated array of object pointers, as we saw in the coding example of section 1.6, program #5
  - (b) the other will be a singly linked list, based on the coding example of section 3.1, program #7, but with two modifications:
    - (i) the list will maintain a tail, in addition to a head
    - (ii) every time the list is printed, after the data is shown, the head and tail data will be printed a second time, indicating which is the head and which is the tail

2.2. The `main()` function will contain only two lines of code: one to declare a control object, and the other to call the control object's launch function.

2.3. A menu will be presented to the end user, who is a member of the book club. The user will select an option, and the program will perform the corresponding action. The menu will be displayed again, until the user chooses to exit. The main menu will have the following five (5) options:

- 2.3.1. print the data for all the members in the book club
- 2.3.2. print the data for all the books in the book club
- 2.3.3. allow the club member to rate a specific book, giving it a numeric value between 1 and 10

2.3.4. compute and print out the best rated book (the book with the highest average rating entered by the members who rated that book) and the most rated book (the book with the greatest number of ratings) in the book club

2.3.5. exit the program

2.4. The information stored about each club member will include their first name, last name, and a collection of the books that they have rated. This collection will be stored as a linked list, as described in instruction 2.1.4.

2.5. The information stored about each book can be found in the coding example of section 3.1, program #6.

2.6. The book club will have a name, a collection of all its books stored as a linked list (as described in instruction 2.1.4) ordered first by book author then by title, a collection of its members stored as a dynamically allocated array of pointers (described in instruction 2.1.4) ordered by id, and a collection of ratings stored as a statically allocated collection of pointers.

2.7. The ratings will be modelled by an *association class*, as we saw in Assignment #2. Each rating will represent the fact that a specific club member has rated a specific book, with a numeric rating between 1 and 10, where 10 is the best rating, and 1 is the worst.

2.8. You must provide initialization functions that create sufficient data to thoroughly test your program. At minimum, you must provide data that initializes:

2.8.1. ten (10) different books

2.8.2. five (5) different club members

2.8.3. twenty (20) different ratings involving; at least five (5) of the books must have multiple ratings from different club members

3. Constraints

Your program must comply with all the rules of correct software engineering that we have learned during the lectures, including but not restricted to:

3.1. The code must be written in C++98, and it must compile and execute in the default course VM. It must not require the installation of libraries or packages or any software not already provided in the VM.

3.2. Your program must follow correct encapsulation principles, including the separation of control, view, entity, and collection object functionality.

3.3. Your program must not use any classes, containers, or algorithms from the C++ standard template library (STL), unless explicitly permitted in the instructions.

3.4. Your program must follow basic OO programming conventions, including the following:

3.4.1. Do not use any global variables or any global functions other than `main()`.

3.4.2. Do not use `struct`s. You must use classes instead.

3.4.3. Objects must always be passed by reference, never by value.

3.4.4. Except for simple getter functions, data must be returned using output parameters, and not using the return value.

3.4.5. Existing functions must be reused everywhere possible.

3.4.6. All basic error checking must be performed.

3.4.7. All dynamically allocated memory must be explicitly deallocated.

3.5. All classes must be thoroughly documented in every class definition, as indicated in the course material, section 1.3.

4. Submission

4.1. You will submit in *cuLearn*, before the due date and time, the following:

4.1.1. A UML class diagram (as a PDF file), drawn by you using a drawing package of your choice, that corresponds to the entire program design.

4.1.2. One `tar` or `zip` file that includes:

(a) all source and header files

(b) a Makefile

(c) a README file that includes:

(i) a preamble (program author, purpose, list of source and header files)

(ii) compilation and launching instructions

NOTE: Do not include object files, executables, hidden files, or duplicate files or directories in your submission.

4.2. Late submissions will be subject to the late penalty described in the course outline. No exceptions will be made, including for technical and/or connectivity issues. Do not wait until the last day to submit.

4.3. Only files uploaded into *cuLearn* will be graded. Submissions that contain incorrect, corrupt, or missing files will receive a grade of zero. Corrections to submissions will not be accepted after the due date and time, for any reason.

5. Grading

The grading of this project will be rubric-based, and the total grade will be out of 60 marks.

5.1 Minimal criteria

In order to be graded, your program must meet the following **minimum criteria**:

5.1. The program must be implemented in C++98, and it must compile and execute in the default course VM.

5.2. A correct Makefile must be provided.

5.3. The program must successfully complete at least three (3) executions that meet all requirements and constraints described in sections 2 and 3.

5.4. The program must be initialized with sufficient datafill to thoroughly test the program.

Submissions that do not meet this minimum criteria will not be graded and will earn a grade of zero.

5.2 Rubric-based grading

The project will be assessed in accordance with the grading categories listed in section 5.3. Each category will be graded based on a 6-point rubric. Some categories are worth double the weight of other categories, so their value is 12 points each. These will still be assessed according to the 6-point rubric, but the values will be doubled.

The 6-point rubric is defined as follows:

- [6 points] Excellent (100%)
- [5 points] Good (83%)
- [4 points] Satisfactory (67%)
- [3 points] Adequate (50%)
- [2 points] Inadequate (33%)
- [1 points] Poor (17%)
- [0 points] Missing (0%)

5.3 Grading categories

The project will earn an overall grade out of 60 marks. This will be the sum of the grades earned for each of the following categories, based on the 6-point rubric described in section 5.2:

5.1. **UML [6 marks]:** This criteria evaluates the UML class diagram corresponding to the design and implementation of your entire program. The diagram must be drawn by you, using a drawing package, and it must be submitted as a PDF file. Other formats will not be graded. Your UML class diagram must comply with the UML format and conventions covered in the course material, section 2.3.

5.2. **Object categories [12 marks]:** This criteria evaluates how correctly the functionality of your implementation is separated into control object(s), view object(s), entity objects, and collection object(s), and how well your object design follows the correct software engineering rules of data abstraction and encapsulation.

5.3. **Function design [6 marks]:** This criteria evaluates how effectively the functionality of your implementation is separated into modular functions inside your objects, and how well it follows the correct software engineering rules of data abstraction and encapsulation.

5.4. **Inheritance [6 marks]:** This criteria evaluates the effective development and usage of inheritance in your code, specifically as it relates to the organization of the entity objects, as described in the program requirements in section 2.

5.5. **Collection classes [12 marks]:** This criteria evaluates the effective development and usage of the two required collection classes in your code, as described in the program requirements in section 2.

5.6. **Code quality [12 marks]:** This criteria evaluates the overall code quality. This includes, but is not limited to, how well your classes are implemented, how well the code is written, how well the principle of least privilege is applied, the presence of constructors and destructors where appropriate, how well the code complies with all constraints listed in section 3, etc.

5.7. **Packaging [6 marks]:** This criteria evaluates the packaging of the code into an archive file. This includes, but is not limited to, the correct separation of the code into header and source files, the presence of complete and correctly formatted Makefile and README files, and the absence of duplicate, additional, and/or unneeded files.