

# P1: Recommendations

91.5/100 Points

10/10/2023

Offline Score:  
**91.5/100** Add CommentAnonymous Grading: **no**

## ▼ Details

**Week 2: Recursion & Linked List** ([https://seattleu.instructure.com/courses/1610311/pages/week-](https://seattleu.instructure.com/courses/1610311/pages/week-2-synopsis)[2-synopsis](#))**ICE3: Recursion** (<https://seattleu.instructure.com/courses/1610311/assignments/7157052>)**ICE4: Binary Search** (<https://seattleu.instructure.com/courses/1610311/assignments/7157053>)**>>> P1: Recommendations**   **P1x: Recommendations (EC)**(<https://seattleu.instructure.com/courses/1610311/assignments/7157073>)

## P1: Recommendations


**All Projects (P)**

### Pre-Work:

- **ICE 2: P1 interface** (<https://seattleu.instructure.com/courses/1610311/assignments/7157051>) (optional)
- **Lab 1: P1 file reading** (<https://seattleu.instructure.com/courses/1610311/assignments/7157061>)
- **Lab 2: P1 implementation** (<https://seattleu.instructure.com/courses/1610311/assignments/7157062>)

This programming assignment is to help you practice main C++ constructs such as classes, dynamic memory internal to an object, object state, file I/O and public/private accessibility.

### Introduction – making recommendations

If you've ever bought a book online, the bookseller's website has probably told you what other books you might like. This is handy for customers, but also very important for business. In September 2009, online movie-rental company Netflix awarded one million dollars to the winners of the **Netflix Prize**  (<http://www.netflixprize.com/>). The competition simply asked for an algorithm that would perform 10% better than their own algorithm. Making good predictions about people's preferences is what's important to this company. It is also a very current area of research in machine learning, which is part of the area of computer science called artificial intelligence.


### A basic approach

(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)(<https://seattleu.instru>)

would be to make almost the same prediction for every customer. In this case the program would simply calculate the average rating for all the books in the database, sort the books by rating and then from that sorted list, suggest the top 5 books that Rabia hasn't already rated. With this simple approach, the only information unique to Rabia used by the prediction algorithm was whether or not Rabia had read a specific book.

## A more sophisticated approach

We could make a better prediction about what Rabia might like by considering her actual ratings in the past and how these ratings compare to the ratings given by other customers. Consider how you decide on movie recommendations from friends. If a friend tells you about a number of movies that s(he) enjoyed and you also enjoyed them, then when your friend recommends another movie that you have never seen, you probably are willing to go see it. On the other hand, if you and a different friend always tend to disagree about movies, you are not likely to go to see a movie this friend recommends.

A program can calculate how similar two users are by treating each of their ratings as a [vector](https://tutorial.math.lamar.edu/classes/calci/dotproduct.aspx)  (<https://tutorial.math.lamar.edu/classes/calci/dotproduct.aspx>) and calculating the dot product of these two vectors. (Remember that the dot product is just the sum of the products of each of the corresponding elements.) For example, suppose we had 3 books in our database:

- Rabia rated them [5, 3,-5]
- Suelyn rated them [1, 5,-3]
- Bob rated them [5, -3, 5]
- Kalid rated them [1, 3, 0].

The similarity between Rabia and Bob is calculated as:

$$(5 \times 5) + (3 \times -3) + (-5 \times 5) = 25 - 9 - 25 = -9$$

The similarity between Rabia and Suelyn is:

$$(5 \times 1) + (3 \times 5) + (-5 \times -3) = 5 + 15 + 15 = 35$$

The similarity between Rabia and Kalid is:

$$(5 \times 1) + (3 \times 3) + (-5 \times 0) = 5 + 9 + 0 = 14$$

We see that if both people like a book (rating it with a positive number) it increases their similarity and if both people dislike a book (both giving it a negative number) it also increases their similarity.

Once you have calculated the pair-wise similarity between Rabia and every other customer, you can then identify whose ratings are most similar to Rabia's. In this case Suelyn is most similar to Rabia, so we would recommend to Rabia the top books from Suelyn's list that Rabia hadn't already rated.



(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)

(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)

## Information (or classes)

1. **BookList** class – Your program must keep track of the following information about each book in the system:
  - ISBN number: A string (or int) that uniquely identifies a book.
    - Hint: to keep things simple, this can be generated by your program (i.e. use random or static int).
  - Author: The author of the book.
  - Title: There may be multiple books with the same title.
  - Year: The year (or range of years if a series) in which the book was published.
2. **MemberList** class – Your program must keep track of the following information about each member:
  - Name: The name of the member, e.g., "Diane Horton". It is possible to have two members with the same name.
  - Account: A string (or int) that uniquely identifies a member.
    - Hint: to keep things simple, this can be generated by your program (i.e. use random or static int).

Your program must also keep track of which member is currently logged in, if any. (For example, upon program startup or immediately after a member logs out, there is no one currently logged in.)

3. **RatingList** class – Your program must keep track of book ratings. Each rating has three pieces of information: the book, the member, and the rating. Each rating must be stored as an int with one of these values: -5, -3, 0, 1, 3, 5. They are to be interpreted as follows:

Rating	Meaning
-5	Hated it!
-3	Didn't like it
0	Haven't read it
1	ok - neither hot nor cold about it
3	Liked it!
5	Really liked it!

Although the data on which you test your program will likely be somewhat small, in a real system there would be a great many books and a great many users, and no user would have rated a very large fraction of the books. If you picture the ratings data as a table (rows being members and columns being books, with rating values in the cells of the table), most of the table would be empty. We call this "sparse" data. Keep this fact in mind when designing your structure for storing ratings.

- If you do decide to store ratings as a table, it may be helpful to create a 2D dynamically allocated array. Here's how to do this with an integer array: [Dynamically allocate 2D array \(https://seattleu.instructure.com/courses/1610311/pages/dynamically-allocate-2d-array\)](https://seattleu.instructure.com/courses/1610311/pages/dynamically-allocate-2d-array).



Below are the basic operations your program must provide. You will get to decide (1) how to handle all the "border" conditions that may arise and (2) what the UI will look like.

Your program must provide the operations below to the user.

- **Load external data** – Read data from a file (that was not created by this program). The user must specify the file path for the data file. Each person in the ratings file must be created as a new member, if they don't already exist, and the book and rating data must be loaded into your data structures. For testing, you can add these files (books.txt and ratings.txt) to your CLion p1/ project. Download the project files here: [p1-files.zip](https://seattleu.instructure.com/courses/1610311/files/69761757/download?wrap=1)  
(<https://seattleu.instructure.com/courses/1610311/files/69761757/download?wrap=1>) ↓  
([https://seattleu.instructure.com/courses/1610311/files/69761757/download?download\\_frd=1](https://seattleu.instructure.com/courses/1610311/files/69761757/download?download_frd=1))
- **Add a new member** – This is allowed no matter who is logged in, or even if no one is.
- **Add a new book.**
- **Login** – There are no passwords. Anyone can log in as anyone else simply by using their account.
- **Logout.**
- **View all your own ratings** – Along with these ratings, display your account name.
- **Rate a book.**
- **See recommended books.**
- **Quit.**

Many details are unspecified. For example, when rating a book, do you choose it from a list or must you type its name? When you ask to see recommended books, how many do you see? Part of your task is to make these decisions. Part of your grade will depend on the quality of those decisions.

## Recommended order of work:

This is a substantial project with a lot of moving parts. I would suggest doing the work in this order - note this is a suggestion (not a requirement).

- Read through the introduction, information, and operation sections above.
- Take notes on what should be included in each class (i.e. `BookList`, `MemberList`, `RatingList`)
- Complete **ICE 2: P1 interface**  
(<https://seattleu.instructure.com/courses/1610311/assignments/7157051>), **Lab 1: P1 file reading**  
(<https://seattleu.instructure.com/courses/1610311/assignments/7157061>), and **Lab 2: P1 implementation** (<https://seattleu.instructure.com/courses/1610311/assignments/7157062>)
- Figure out and write the code needed to read and process books.txt and ratings.txt.
  - You should only read each of these files once!
  - Reading of file is not trivial! Be sure to designate a good amount of time doing this, since most operations are dependent on having files read and processed.
  - The processing of data from files is also not trivial. Figure out what needs to happen once you've read and parsed through a line of data within the files.
  - Create objects as they come. Do not use temporary storage to "hold" info before processing.



(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)

(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)

- Note that the grader may use any file path, so please ask the user to enter the file names when coding this assignment.
- In order to grab each line (i.e. string) from a file, delimited by comma, you can use

`getline()`:

```
string filename = "sometextfile.txt";
string str;
ifstream infile;
infile.open(filename);
while(getline(infile, str, ',')) {
    cout << str << endl;
}
```

- You can also use `getline()` for grabbing a line from file, delimited by space. Here is a reference, though you'll need to modify to separate numbers instead of strings. I would suggest using method 3: <https://www.geeksforgeeks.org/split-a-sentence-into-words-in-cpp/> [↗](https://www.geeksforgeeks.org/split-a-sentence-into-words-in-cpp/)(<https://www.geeksforgeeks.org/split-a-sentence-into-words-in-cpp/>)
- You may want to first work on printing all the information read in from files, to make sure that everything is loaded properly. Here is a sample output for this portion: [P1: Load file](https://seattleu.instructure.com/courses/1610311/pages/p1-load-file) (<https://seattleu.instructure.com/courses/1610311/pages/p1-load-file>)
- Figure out what it means to be logged in versus logged out. Write the code that allows a user to login/logout.
  - How do you indicate that a person is logged in or logged out? See sample output.
  - What operations are possible when logged out? What operations can only be performed when logged in? See sample output.
  - What will your UI look like when logged in or out? See sample output.
- Add a new member – Do you need to be logged in? See sample output.
- Add a new book – Do you need to be logged in? See sample output.

And the hardest parts...

- Rate a book – Do you need to be logged in? See sample output.
- View all your ratings – Do you need to be logged in? See sample output.
- See recommended books – Do you need to be logged in? See sample output.

### Programs must:

1. Provide the requested functionality
2. Be appropriately documented (see [C++ Format Guide](https://seattleu.instructure.com/courses/1610311/pages/c++-format-guide) (<https://seattleu.instructure.com/courses/1610311/pages/c++-format-guide>) and [Programming Process & Documentation](https://seattleu.instructure.com/courses/1610311/pages/programming-process-and-documentation) (<https://seattleu.instructure.com/courses/1610311/pages/programming-process-and-documentation>))
3. Use C++ constructs as intended
4. **USE DYNAMIC MEMORY – You need to create dynamic arrays in your class(es)!**
5. You may NOT use [STL containers](https://www.geeksforgeeks.org/the-c-standard-template-library/) [↗](https://www.geeksforgeeks.org/the-c-standard-template-library/)(<https://www.geeksforgeeks.org/the-c-standard-template-library/>)



(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)

(<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>)

`p1.cpp`).

- ALL ASSUMPTIONS SHOULD BE REASONABLE AND CLEARLY STATED
- DO NOT HARD CODE
- TEST YOUR PROGRAM BEFORE SUBMISSION

### Sample output:

- [P1: Sample output \(https://seattleu.instructure.com/courses/1610311/pages/p1-sample-output\)](https://seattleu.instructure.com/courses/1610311/pages/p1-sample-output)

### Submission

You must name your files **BookList.h**, **BookList.cpp**, **MemberList.h**, **MemberList.cpp**, **RatingList.h**, **RatingList.cpp**, and **p1.cpp** (Do not create any additional classes for this project).

To submit, type the following command at the prompt in the directory where the P1 files reside on CS1:

```
/home/fac/mthayer/submit/23fq5005/script/p1_runme
```

You have read/write permissions on your submission directory at:

```
/home/fac/mthayer/submit/23fq5005/p1/yourusername
```

### ▼ View Rubric

### Select Grader

Elaine Huynh (TA)



<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>

<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>

**P1: Recommendations**

Criteria	Ratings	Pts
Dynamic Array Constructors <a href="#">view longer description</a>	<b>Comments</b> -5: missing copy constructor for all classes.  In C++, when you create a new class, it is important to define a copy constructor and an overloaded assignment operator in order to properly manage the memory and avoid unexpected behavior. Please refer to the week 2 recorded lecture if you'd like to learn more about this topic.	5 / 10 pts
Dynamic Array Destructor <a href="#">view longer description</a>		5 / 5 pts
Dynamic Array Assignment <a href="#">view longer description</a>		10 / 10 pts
Dynamic Array Resize (in insert) <a href="#">view longer description</a>		10 / 10 pts
All classes are properly defined <a href="#">view longer description</a>		10 / 10 pts
System loads files <a href="#">view longer description</a>		5 / 5 pts
Add a new member <a href="#">view longer description</a>		5 / 5 pts
Add a new book <a href="#">view longer description</a>		5 / 5 pts
Login / Logout <a href="#">view longer description</a>	<b>Comments</b> Consider implementing inside of Member class should move login/logout functionality to member; restrict functions (and include in preconditions) when not logged in (-1)	4 / 5 pts
View all your own ratings <a href="#">view longer description</a>		5 / 5 pts
Rate a book <a href="#">view longer description</a>	<b>Comments</b> No validation for rating input (can input any number)	5 / 5 pts


<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>
<https://seattleu.instructure.com/courses/1610311/modules/items/17916952>

P1: Recommendations		
Criteria	Ratings	Pts
<a href="#">view longer description</a>	-2: book recommendations should be calculated in a function in the RatingList class	
Documentation/style <a href="#">view longer description</a>	<b>Comments</b> -0.5: function documentation insufficient, need to document parameters and return types (can use Javadoc)	4.5 / 5 pts
Client program achieves goals <a href="#">view longer description</a>		15 / 15 pts
Other comments <a href="#">view longer description</a>	<b>Comments</b> Nice work!	0 / 0 pts
		Total Points: 91.5