Project 1 – Recommendations

CS 251, Spring 2021, Reckinger

[UPDATE, 01/18/2021, 3:20 pm] - The autograder is now final. Please see the starter code folder for the data files containing all the student ratings you submitted for books, movies, and songs. Enjoy!

Collaboration Policy: By submitting this assignment, you are acknowledging you have read the collaboration policy in the syllabus for projects. This project should be done individually. You may not receive any assistance from anyone outside of the CS 251 teaching staff. Also, please see copyright at the bottom of this description.

Late Policy: You are given a total of 5 late days to use at your discretion (for all projects, not each project). You may use the late days in 24-hour chunks (either 1 day at a time or all five at once). You do not need to alert the instructor to ask permission to use late days. You can manage your use of late days on Mimir directly.

Early Bonus: If you submit a finished project early (by Wednesday, January 20th at 11:59 pm), you can receive 10% extra credit. Please note, your submission needs to pass 100% of the tests cases to receive the early bonus.

Test cases/Submission Limit: Unlimited submissions.

What to submit: main.cpp, ourvectorAnalysis.txt, and fill out this ratings form. Your main.cpp should include your own Creative Component. Your ourvectorAnalysis.txt should include your analysis of your usage of vectors.

.pdf starterCode solution.exe

Project Summary

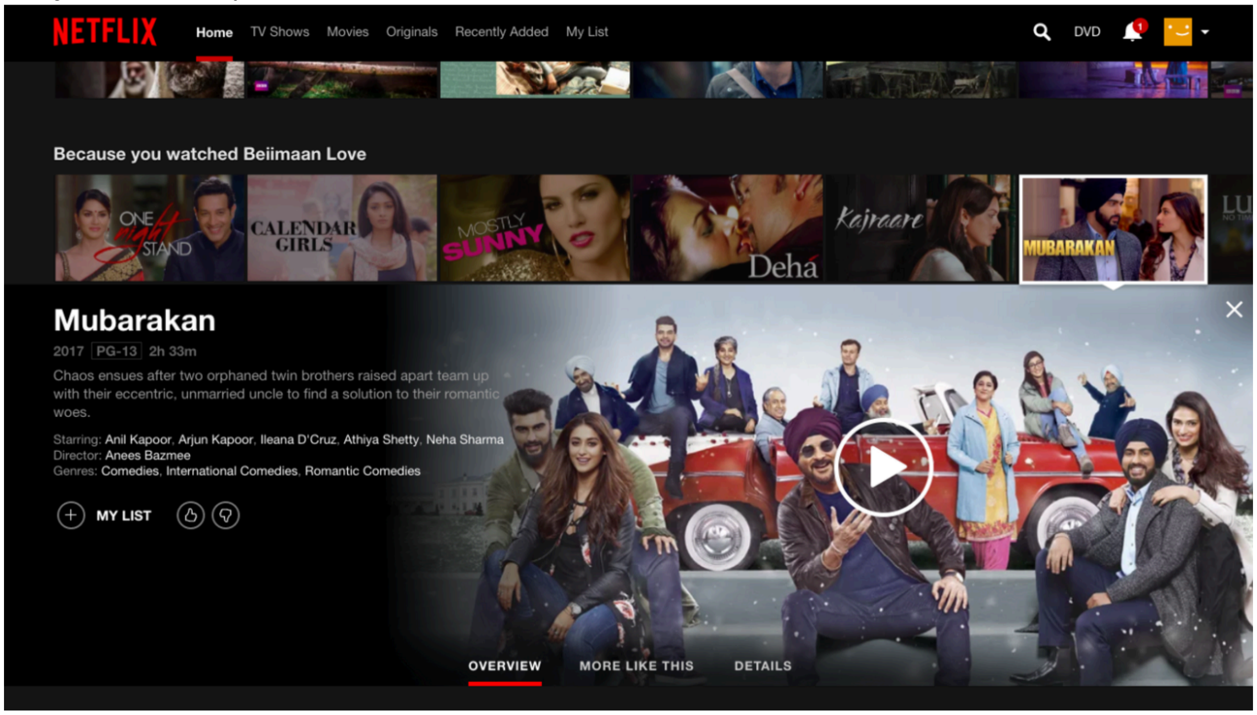

NETFLIX
Home
TV Shows
Movies
Originals
Recently Added
My List
DVD
Because you watched Beiimaan Love
Mubarakan
2017 PG-13 2h 33m
Chaos ensues after two orphaned twin brothers raised apart team up with their eccentric, unmarried uncle to find a solution to their romantic woes.
Starring: Anil Kapoor, Arjun Kapoor, Ileana D'Cruz, Athiya Shetty, Neha Sharma
Director: Anees Bazmee
Genres: Comedies, International Comedies, Romantic Comedies
MY LIST
OVERVIEW
MORE LIKE THIS
DETAILS

Image Credit: medium.com

If you've ever bought a anything online, the website has probably told you what other items you might like. This is handy for customers, but also very important for business. Making good

predictions about people's preferences is important to most companies. It is also a very current area of research in machine learning, which is part of the area of computer science called artificial intelligence. Back in 2009 when this area of research was in its infancy, Netflix awarded $1M to a team who could come up with the an algorithm for making recommendations that improved its own recommendations by 10% (Netflix Prize).
First, take a look at the app by downloading the solution executable (solution.exe). Download the file to your local machine. Upload it to your Mimir project1 folder. Inside the terminal, type "chmod a+x solution.exe". Then type "./solution.exe" to run the solution. You can now use this if you want to see the behavior of the solution code. However, you can also see it by submitting to the test cases and seeing the expected output.
For information about how to use Mimir and how to compile/run your code on Linux, see the last section in this document called "Getting Started in Mimir".

Approaching this Project

You might be able to complete this project by looking at the solution.exe behavior and the test cases on Mimir, plus reading some of the details below. Many people might prefer that. However, lots of students who are new to these larger projects find them a bit overwhelming. Therefore, I will break down the project by milestone.

Milestone 1 – Fill out your ratings

In order to give us some interesting data to use, you must fill out this ratings form by Sunday, January 17th, 2021. It is required that you fill this out by the deadline to receive full credit.

Milestone 2 – Load the data

In order for the recommendations app to run, the first step is to load the data (again, see solution.exe to try it out). The command for loading the data is (using the tiny data with podcasts as an example, note keyboard input is always in red in my descriptions):

```
Enter command or # to quit: load podcastsTiny.txt ratingsTiny.txt
```

Your program should do all file reading when this command is run. Check out the format of those files in the starter code. The first file in the command lists the "items". This could be podcasts (like this example), books, movies, etc. Those are the items being rated and recommended. The second file is the ratings. The format of the file is:

```
Jesse
-3 5 -3 0 -1 -1 0 0 5
Shakea
5 0 5 0 5 0 0 0 1
```

First, will come the name of the user (spaces are allowed in the user name), and the line immediately following is the user's ratings. You can assume the ratings are always ordered in the same order that they items are listed in the items file. Your app should work for any number of items and any number of users that are contained in those files. Error handling for this step will not be tested in this project, however, for ease of testing, we strongly encourage you to handle the user typing in an invalid file. How you handle is up to you as we will not be testing it.

The ratings 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 users 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.

For this project, you are restricted to using only one type of container: ourvector. No other containers are allowed (including vector, list, etc.) The implementation file is provided in the starter code (ourvector.h). You will need to read this file to start getting familiar with how to use the class. You will also find examples in the lecture slides. In general, ourvector works similarly to the vector class provided with C++. Remember, no global variables are allowed, you pass your containers in and out of functions.

As you are writing out the code for this first Milestone, you should be brainstorming how to design your code. Function decomposition is critical for this project. It is possible to receive deductions of up to 50 points if your code is not properly decomposed. See lecture on style and function decomposition for details.

Milestone 3 – User login

Next up, you should write the code that allows the user to login. Notice, we will not be using password for our app, but you can easily imagine how this could be extended to include them. The command to login is:

```
Enter command or # to quit: login Maria
```

You will need to design your code to keep track if a user is logged in and which user is logged in. Also, check out the test cases and solution.exe for how to handle edges cases (if user name is entered incorrectly, does not exist, etc.)

Milestone 4 – Show user's ratings

Write the code that shows the user what items they have rated and what those ratings are. The command for this:

```
Enter command or # to quit: show
```

See the test cases and solution.exe for details on the format for this.

Milestone 5 – Basic recommendations

How might we write an algorithm to make recommendations? Consider a user named Maria. How is it that the program should predict podcasts Maria might like? The simplest approach would be to make almost the same prediction for every user. In this case the program would simply calculate the average rating for all the podcasts in the database and then suggest the top 5 podcasts that Maria hasn't already rated. When calculating the mean you will want to include all ratings in the mean calculation, including the zeros (i.e. user has not listened). With this simple approach, the only information unique to Maria used by the prediction algorithm was whether or not Maria had already listened to a specific podcast.

For this milestone, add the functionality to give basic recommendations. The command for this is:

```
Enter command or # to quit: basic
```

This should produce a list of the top five items recommended based on the basic recommendation algorithm. If there are less than 5 recommended items, show however many there are. Ties should be handled using insertion order. See the test cases and solution.exe for formatting details of this output.

Milestone 6 – Advanced recommendations

We could make a better prediction about what Maria 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 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 podcasts in our data-base and Maria rated them [5, 3,-5], Suelyn rated them [1, 5,-3], Bob rated them [5, -3, 5], and Kalid rated them [1, 3, 0]. The similarity between Maria and Bob is calculated as: (5 x 5) + (3 x -3) + (-5 x 5) = 25 - 9 - 25 = -9. The similarity between Maria and Suelyn is: (5 x 1) + (3 x 5) + (-5 x -3) = 5 + 15 + 15 = 35. The similarity between Maria and Kalid is (5 x 1) + (3 x 3) + (-5 x 0) = 5 + 9 + 0 = 14. We see that if both people like a podcast (rating it with a positive number) it increases their similarity and if both people dislike a podcast (both giving it a negative number) it also increases their similarity.

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

For this milestone, add the functionality to give advanced recommendations. The command for this is:

```
Enter command or # to quit: advanced
```

This should produce a list of the top five items recommended based on the advanced recommendation algorithm. If there are less than 5 recommended items, show however many there are. Ties should be handled using insertion order. Please note that with small data files, the top five recommendations may include items that the most similar user has not seen (rated 0) or even rated poorly (rated negatively). The algorithm states you should **recommend the top five items of the most similar user that the current user has not currently rated**. See the test cases and solution.exe for formatting details of this output.

Milestone 7 – print

Next, you should add two commands to your program:

```
Enter command or # to quit: print users
Enter command or # to quit: print items
```

The first should print out a list of all users and the second should print out a list of all items (podcasts, books, or etc…which should be numbered starting at 0). You will see in the next milestone that you are able to add users and ratings to your program, therefore, these print commands will help with those commands.

Milestone 8 – add

Finally, you should add two commands to your program:

```
Enter command or # to quit: add user
Enter command or # to quit: add rating
```

The first should add a user to the data base. After the command, the user should also now be logged in. The user should have no ratings. The second command should add a rating to the currently logged in user's account. The user should be prompted to ask which item they would like to rate (using the numbered list from the print command). Check out the test cases and solution.exe to see the behavior of this. If the user already has a rating for the item, the new rating should be overwritten.

Using C++ Syntax

You should use C++ strings and C++'s string library functions. You should also use C++ streams for file reading. It is recommended to use **getline** instead of the insertion operator (<<) for the file reading. See lecture notes and lab for file reading template code. The district numbers and the vote counts will always be integers.

Note: Your code will be tested on other files of the same format. Do not assume that the files only have some number of lines or anything else about the files other than what has been described here about their format.

Note 2: Think about how you are going to store this data. Think about it…now. Don't get too deep in file reading and string parsing and calculations before reading through the rest of the project and deciding where you want to put all this. One ourvector? Multiple ourvectors? Structs? The choice is yours. No abstractions or data structures are allowed other than ourvecotor. A little planning will go a long way!

Creative Component

As a final functionality requirement, you must write your own command of your choice. Therefore, in addition to your code responding to quit, load, show, basic, advanced, etc. you must develop your own command and its functionality. This open-ended piece will be graded using the rubric item in Mimir. For full credit: (1) you must have instructions in the header comment at the top of your file on how to use your command; (2) must use data provided in two data files given (no additional files allowed); (3) must be either a calculation(s) or a visualization of data; (4) must be relevant to the project and interesting; (5) the output must look nice and be well thought out; (6) must be original and your own work, your work should not be similar to others; (7) extraordinary work may receive up to 5% extra credit with instructor approval.

- Use a more sophisticated recommendation algorithm. The algorithm we've described used the ratings of every customer to calculate how similar Maria is to every other user in the database and find the single user who is most similar to Maria. Once the algorithm finds that Suelyn is the closest match to Rabia, it only uses Suelyn's ratings to make suggestions for Rabia and doesn't consider the ratings of anyone else at all. Wouldn't it be more interesting to consider the ratings of a number of different people who are similar to Rabia? But how many and how much weight should we give to the different recommendations? This is just one suggestion; you are free to invent something completely different.
- Incorporate passwords. You will need to save the passwords in a file between runs of the program. Be sure to not change the login command, leave it as is and add your own (like login2).

- Have a special "admin" account. Give the admin account special privileges. For example, perhaps only the admin can create or delete accounts.
- Incorporate a feature to search for all items beginning with a given string, or including a certain string. Use this to make other operations easier for the user.
- Keep additional information about podcasts or users. For example, you could store a cover picture for each podcast (url) and display it when members are rating the podcast.

The additional features you implement through the creative component could be practically anything! Please don't ask how many points your enhancements will earn: we can't evaluate your code before the due date! For a good grade on this aspect, we do expect to see significant enhancements.

Ourvector Analysis

Open ourvector.h and go to the member function _stats. Make sure this portion of the code is not commented out:

```
    cerr << "**********************************************************" << endl;
    cerr << "ourvector<" << name << "> stats:" << endl;
    cerr << " # of vectors created:   " << Vectors << endl;
    cerr << " # of elements inserted: " << Inserts << endl;
    cerr << " # of elements accessed: " << Accesses << endl;
    cerr << "**********************************************************" << endl;
```

Run your code (make sure to run all commands, including to quit) with this version of ourvector.h and you should see some vector stats print at the bottom of your code:

```
*********************************************************
ourvector<T> stats:
 # of vectors created:   <X>
 # of elements inserted: <Y>
 # of elements accessed: <Z>
*********************************************************
```

Submit a file named “ourvectorAnalysis.txt”. Copy and paste what you get for the report above into your text file (include all input, output, and the ourvector report). In it you must write approximately 250 words to justify why, when, and how your code has created X number of vectors, has inserted Y elements, and accessed Z elements. Your analysis should be specific and quantitative. For the vectors created, you should identify all line numbers of your code where those vectors are created. You should be specific enough that your counts add up to exactly the amount of vectors. For the number of elements inserted and number of elements accessed, you can estimate more approximately based on the sizes of the vectors and the types of operations you are doing.

Your code should minimize number of vectors created, elements inserted, and elements accessed. One way to do this is to make sure you are passing by reference all of your large data containers. Another way is avoid searching too often and avoid sorting (think of doing a partial sort or something equivalent). Each piece of data should only be stored exactly once, and you should justify that this is the case. Only make copies of data when it is absolutely necessary (and justify this). Structs are encouraged!

Learning C++ and More about ourvector

You’ll need some string processing, namely finding characters within a string, and extracting a substring. While you can certainly write these functions yourself, it’s expected that you’ll use the .find() and .substr() functions built into the **string** class provided by C++: http://www.cplusplus.com/reference/string/string/. Don’t forget to #include <string>.

Your solution is required to store all data in a **vector<T>** class --- to be precise, in a vector-compatible implementation we are providing: **ourvector<T>**. For the purposes of this assignment, always start with an empty vector, and then add data to the vector by calling the **push_back()** member function. When you need to access an element of the vector, use the **.at()** function, or the more convenient and familiar **[ ]** syntax. To empty a vector, use the **.clear()** function. For more info on vector<T>: http://www.cplusplus.com/reference/vector/vector/.
Don't forget to #include "ourvector.h" (it is already in the starter code).
Finally, to work with files, #include <fstream>. To read from a file, use an **ifstream** object, and use the >> operator when inputting a single value (e.g. integer or single word). When you need to input 1 or more words into a single string variable, use **getline(infile, variable)**.

Programming Environment

You are required to program your project in Mimir environment. Mimir allows us to all work in the same space and allows all teaching staff to best assist you. It also minimizes messing with cross platform problems. Mimir tracks your progress as you write the code, which will assist us when grading your work.
Mimir is set up to automatically use your highest scoring submission.
On Mimir, we are compiling via **g++** with **-std=C++11**. Do not ask us to change the C++ version; we are compiling against C++ 11.
See the section below called "Getting Started in Mimir" if you need help…getting started in Mimir. If you are new to a Linux environment (most students are), please pay special attention to what director (folder) you are in. Lots of students struggle with this.

Coding Style

You will be asked adhere to the Google C++ coding style guidelines, which are described here: https://github.com/google/styleguide/blob/gh-pages/README.md
This is autograded using our style test. You should also review the style guidelines from lecture and the style rubric items on Mimir. Your code will be graded for style and any violations to our style criteria will be a deduction on your final score.

Requirements

1. You must use **ourvector<T>** ("ourvector.h") for storing all data. No other containers may be used.
2. You are allowed to use and add other libraries (make sure to **include** them at the top of your file). However, you do not need many.
3. Each input file may be opened and read exactly once; store the data in ourvector<T> if need be. The file reading must occur only when load is called.
4. The algorithm for searching must be written by you, no library functions allowed. Sorting all data should not be necessary and is discouraged (think of something like a partial sort to find only the top five).
5. You should be able to identify and count all ourvectors created. The # of inserts and the # of accesses must be reasonable. You should determine that yourself and justify in your submission (in "ourvectorAnalysis.txt").
6. Your main.cpp program file must have a header comment with your name and a program overview. Each function must have a header comment above the function, explaining the function's purpose, parameters, and return value (if any). Inline comments should be supplied as appropriate; comments like "*declares*

variable" or "*increments counter*" are useless. Comments that explain non-obvious assumptions or behavior *are* appropriate.

7. Each command must be implemented using a function; this implies a complete solution must have at least as many functions as there are commands. However, a good solution will have many more functions to decompose your code properly.
8. No global variables; use parameter passing and function return.
9. The **cyclomatic complexity** (CCN) of any function should be minimized. In short, cyclomatic complexity is a representation of code complexity --- e.g. nesting of loops, nesting of if-statements, etc. You should strive to keep code as simple as possible, which generally means encapsulating complexity within functions (and calling those function) instead of explicitly nesting code. Here's an example of simpler code with low CCN:

```
while (…) {
   if (searchFunctionFindsWhatWeNeed(…))
     doSomething();

   next value;
}
```

   Here's an example of complex code with high CCN:

```
while (…) {
   for (…) {  // loop to do search
      if { (search condition is met)
         for { (…) // now do something:
           …
         }
      }
   }

   next value;
}
```

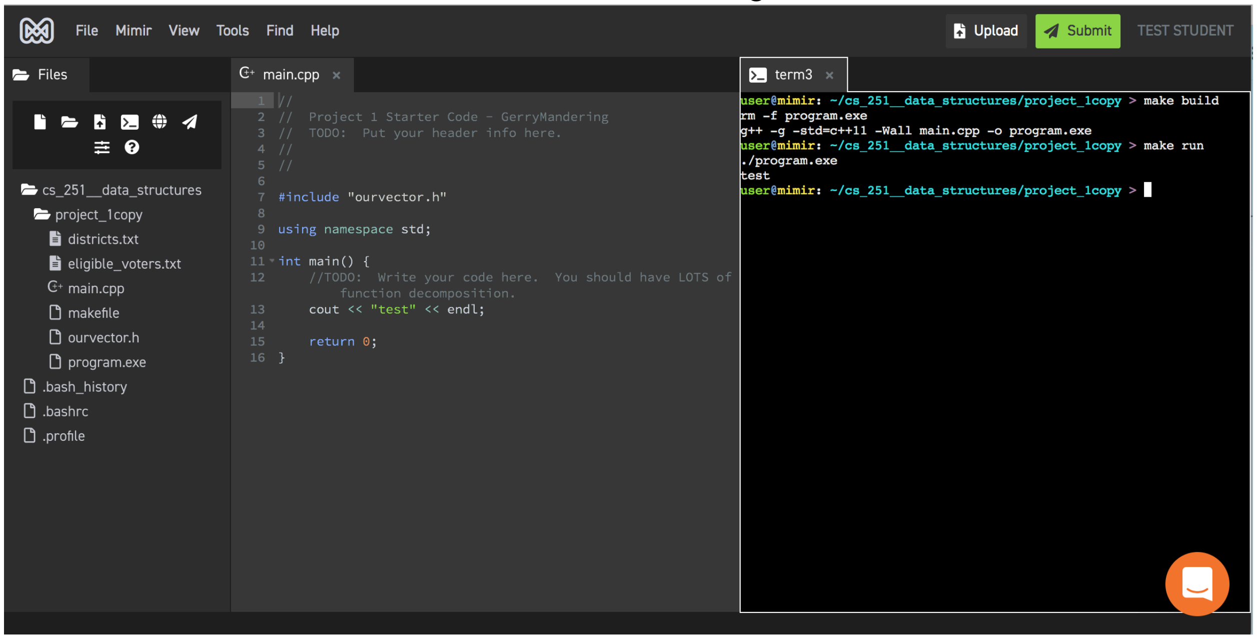
As a general principle, if we see code that has **more than 2 levels** of explicit looping --- an example of which is shown above --- that score will receive grade penalties. The solution is to move one or more loops into a function, and call the function.

Getting Started in Mimir

Click on "Open in Mimir IDE".

Click on the course, then the project folder (yours will be called project1_recommendations, below it shows project1_copy). Inside of the Project 1 folder, open the main.cpp file and add a few cout statements to the code. Look at how the terminal below AND the directory on the left from which we opened main are the same. Don't forget to save your code, it does not save automatically. To compile, type "make build" in command window. To run, type "make run". You will need to save the file, compile, and run each time you make changes to your code.

To test your code against the test cases and submit, click submit. Make sure to click the correct file to submit and the correct test cases to submit against.



```
//
//   Project 1 Starter Code - GerryMandering
//   TODO:  Put your header info here.
//
//

#include "ourvector.h"

using namespace std;

int main() {
    //TODO:  Write your code here.  You should have LOTS of
        function decomposition.
    cout << "test" << endl;

    return 0;
}
```

```
user@mimir: ~/cs_251__data_structures/project_1copy > make build
rm -f program.exe
g++ -g -std=c++11 -Wall main.cpp -o program.exe
user@mimir: ~/cs_251__data_structures/project_1copy > make run
./program.exe
test
user@mimir: ~/cs_251__data_structures/project_1copy >
```

NOTE: Once you have the IDE open, you may want to adjust the settings. Click File->Settings. We recommend unchecking the box, "use spaces instead of tabs".
TIP: Ctrl+C will cancel a run command if you are stuck in a stream/elsewhere.
TIP2: You can set the max character width on screen, which helps with coding style requirement: file->settings-> print margin column = 80, file->settings-> show print margin line = on.
TIP3: File -> Settings -> *Uncheck* Show intercom help bubble

Citations/Resources

Assignment is inspired by Michelle Craig, University of Toronto.