ACADEMIC YEAR: 2023-2024

ASSIGNMENT 1

Assignment Posted: **January 27, 2024**

Assignment Due: **February 16, 2024, before 11.59pm**

Final Deadline with 20% flat Penalty: **February 19, 2024, before 11.59pm**

"Bodily exercise, when compulsory, does no harm to the body; but knowledge which is acquired under compulsion obtains no hold on the mind."

— Plato

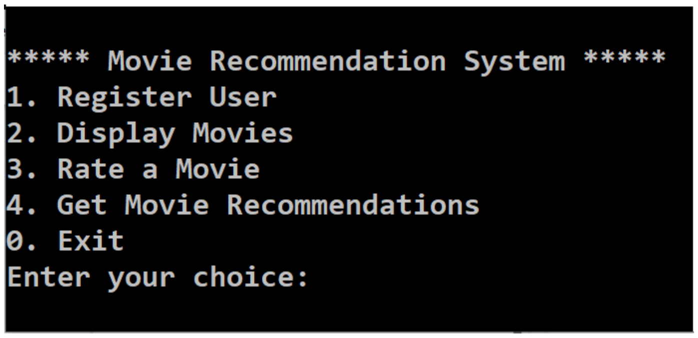
Description:

In this assignment, you will gain hands-on experience with the C programming language. While you are not required to be a C expert to complete the work, you will certainly have the opportunity to explore most of the things that we have discussed in class (and a few other things as well).

NOTE: While the assignment document is a bit long, describing the assignment itself is relatively straightforward. The majority of the content comprises helpful hints and advice intended to simplify the process of writing your first C program.

In this assignment, you'll develop a movie recommendation system using C programming language. Movie recommendation systems such as Netflix, Amazon Prime, Hulu, YouTube, etc. have become integral in the entertainment industry, providing users with personalized suggestions based on their preferences and viewing history. Throughout this assignment, you will explore the fundamentals of C programming while applying your skills to design a reliable recommendation system. This application, of course, will be MUCH simpler.

Your application must display the welcome message and a main menu as depicted in Fig.1 below. The application will have five options and you may assume a perfect user who will always enter an appropriate type input, for instance an integer for number input.



```
***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice:
```

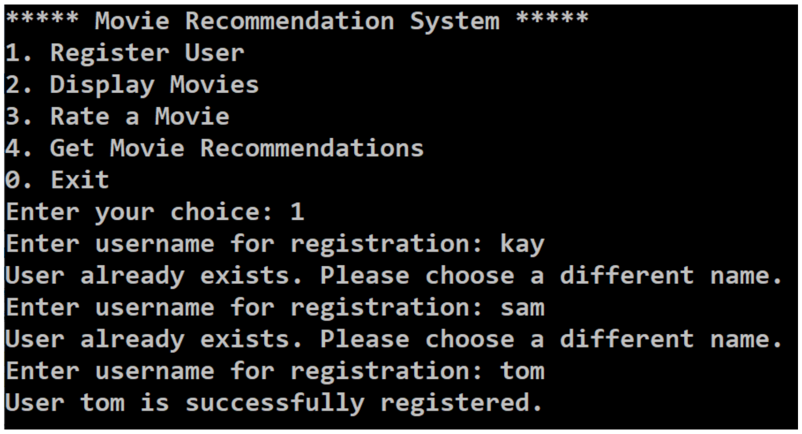
Figure 1. Main menu

Implementation Specifications:

Detailed behavior of about each menu option is mentioned below with screenshots.

1. **Register User**

   Request a user to provide a name for registration within the application. It is assumed that only the first name is required. The application is required to keep track of all the registered users in a text file named "user_data.txt" and should verify whether a user with the entered name already exists. The search for existing users should be case-insensitive, meaning that variations in capitalization (e.g., Sam, SAM, SaM, sam) should all be considered equivalent. If a user with the specified name is found in the database, application should output "User already exists. Please choose a different name." This process should continue until the user inputs a name that is not present in the database. Upon a successful registration, the application should display the message "User is successfully registered" followed by the main menu (refer to Fig. 2).



```
***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 1
Enter username for registration: kay
User already exists. Please choose a different name.
Enter username for registration: sam
User already exists. Please choose a different name.
Enter username for registration: tom
User tom is successfully registered.
```

Figure 2. Register a user

2. **Display Movies**

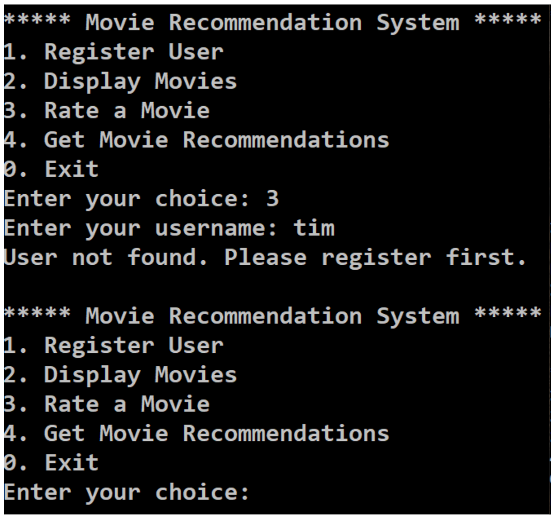
   List of all the available movies in the database, which is stored in a text file name "movie_database.txt", is displayed in an indexed manner.

3. **Rate a Movie**

   Request the user to input a username. If the provided username is not found in the user database, the application should display the message "User not found. Please register first." Subsequently, the main menu must be presented again. In the event that the entered username exists, the application should exhibit a list of all movies from the database.

   Next, the user is prompted to input a movie index number to provide a rating. It is assumed that the user will enter a number; nonetheless, the program must validate the number to ensure it falls within the specified range. For example, if there are ten movies, any numbers outside the range of 1-10 must then be considered invalid. The validation process must continue until the user enters a valid number. Once a valid movie index number is entered, the user is prompted for a rating between 1 and 5. Similar to the validation process for the movie index number, the rating value must be verified. This validation loop continues until the user inputs a valid number. Upon entering a valid rating, the program updates the rating for that particular movie and user. A message "Rating recorded successfully." is then displayed.

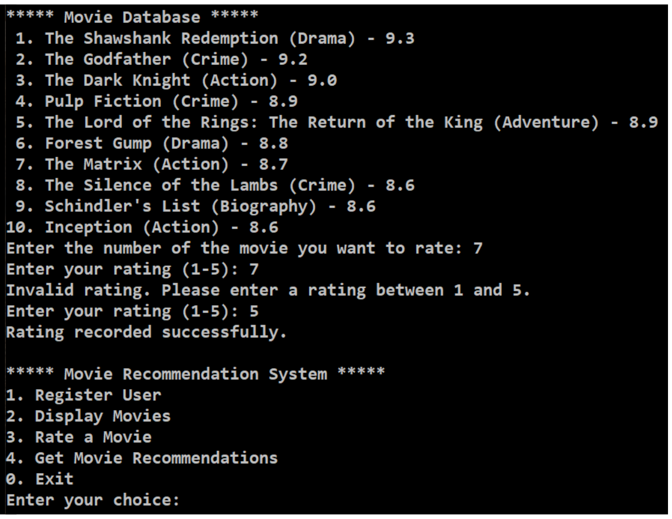
The rating information is preserved in an ***m*** x ***n*** matrix, where '***m***' represents number of movies and '***n***' represents number of users. The updated matrix is stored in a text file named "user_ratings.txt."



```
***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 3
Enter your username: tim
User not found. Please register first.

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice:
```

(a)



```
***** Movie Database *****
 1. The Shawshank Redemption (Drama) - 9.3
 2. The Godfather (Crime) - 9.2
 3. The Dark Knight (Action) - 9.0
 4. Pulp Fiction (Crime) - 8.9
 5. The Lord of the Rings: The Return of the King (Adventure) - 8.9
 6. Forest Gump (Drama) - 8.8
 7. The Matrix (Action) - 8.7
 8. The Silence of the Lambs (Crime) - 8.6
 9. Schindler's List (Biography) - 8.6
10. Inception (Action) - 8.6
Enter the number of the movie you want to rate: 7
Enter your rating (1-5): 7
Invalid rating. Please enter a rating between 1 and 5.
Enter your rating (1-5): 5
Rating recorded successfully.

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice:
```

(b)

Figure 3. Rate a movie

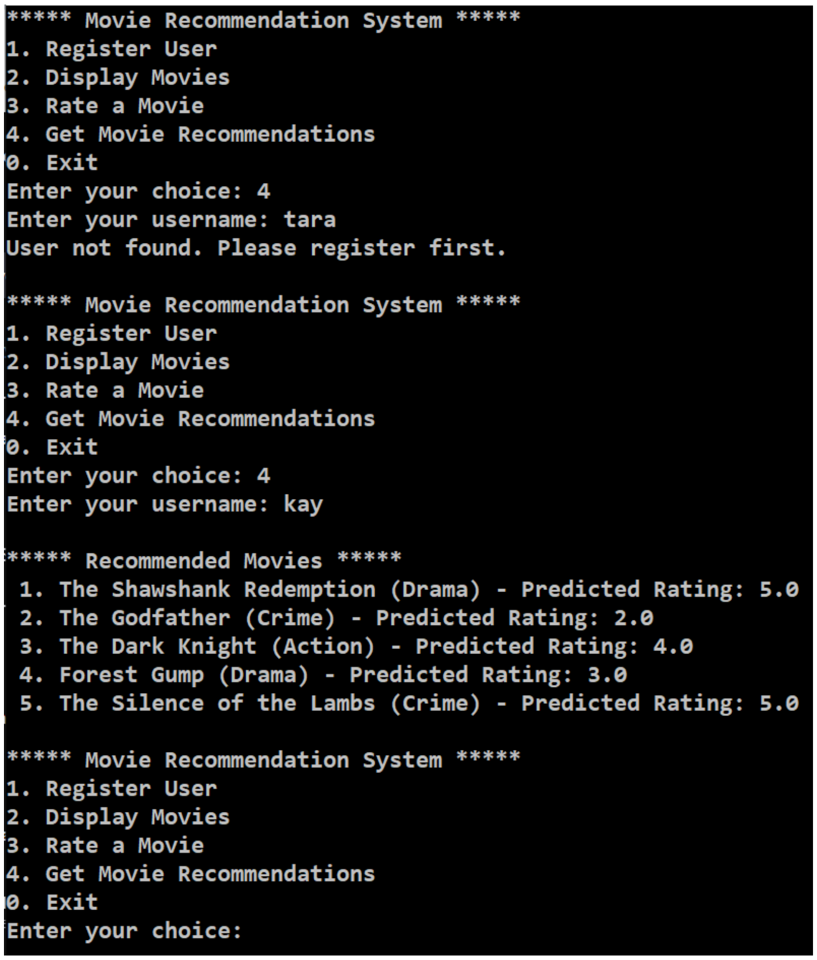
4. Get Movie Recommendations

Request user for a username. If the username doesn't exist in the user database, print "User not found. Please register first." If username is found, use the recommendation algorithm to recommend the movies and print the list of recommended movies along with the average ratings provided by other users.

Recommendation Algorithm:
The recommendation algorithm follows a collaborative filtering approach, which is fairly straightforward. The goal is to suggest movies to a user based on the preferences of other users who share similar tastes. The algorithm involves the following steps:

I. ***Filtering Rated Movies:***
Exclude movies that target user has already rated, as these movies are assumed to have been seen.

II. ***Average Rating Computation:***
For each unseen movie, calculate the predicted rating by averaging ratings given by all other users who have rated that movie.



```
***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 4
Enter your username: tara
User not found. Please register first.

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 4
Enter your username: kay

***** Recommended Movies *****
 1. The Shawshank Redemption (Drama) - Predicted Rating: 5.0
 2. The Godfather (Crime) - Predicted Rating: 2.0
 3. The Dark Knight (Action) - Predicted Rating: 4.0
 4. Forest Gump (Drama) - Predicted Rating: 3.0
 5. The Silence of the Lambs (Crime) - Predicted Rating: 5.0

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice:
```

Figure 4. Get movie recommendations

III. ***Prediction Formula:***

Let u be the target user, i be an unseen movie, and $R\,(u,\,i)$ be the predicted rating for movie i for user u. The formula for predicting the rating $R\,(u,\,i)$ is given by:

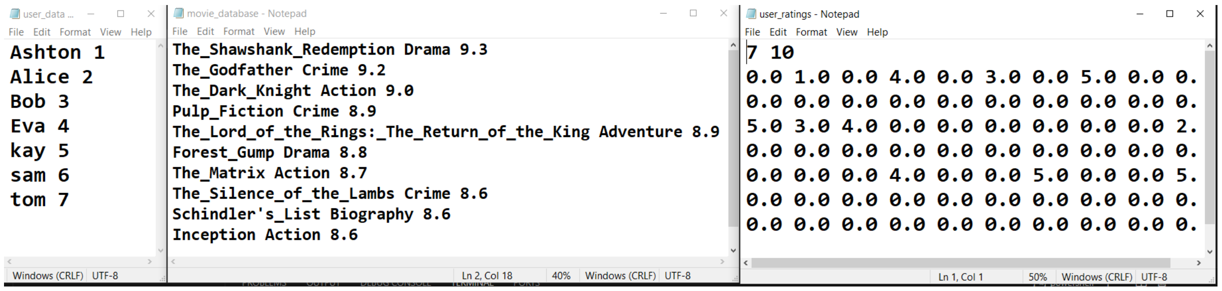
$$R\,(u,i) = \frac{\sum_v R\,(v,i)}{N}$$

where v represents all other users, $R\,(v,\,i)$ is the rating given by user v to movie i, and N is the total number of users who have rated movie i. ***N** must be **non-zero**.*

This algorithm leverages the collective wisdom of other users to predict how much the target user might like unseen movies. The exclusion of movies already rated by the user ensures that the recommendations are personalized and aligned with the user's unique preferences.

5. Exit

The application must exit gracefully and all the resources must be released upon pressing the zero ('0') key.



```
Ashton 1
Alice 2
Bob 3
Eva 4
kay 5
sam 6
tom 7
```

```
The_Shawshank_Redemption Drama 9.3
The_Godfather Crime 9.2
The_Dark_Knight Action 9.0
Pulp_Fiction Crime 8.9
The_Lord_of_the_Rings:_The_Return_of_the_King Adventure 8.9
Forest_Gump Drama 8.8
The_Matrix Action 8.7
The_Silence_of_the_Lambs Crime 8.6
Schindler's_List Biography 8.6
Inception Action 8.6
```

```
7 10
0.0 1.0 0.0 4.0 0.0 3.0 0.0 5.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
5.0 3.0 4.0 0.0 0.0 0.0 0.0 0.0 0.0 2.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 4.0 0.0 0.0 5.0 0.0 0.0 5.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
```

Figure 5. A sample set of input files

Submission:

Assignment must be submitted only through Moodle. No other form of submission will be considered. Please create a zip file containing your C code, input text files, a readme file (.txt). The zip file should be named Assignment#1_YourStudentID. In the readme file document the features and functionality of the application, and anything else you want the grader to know.

Additional Information

❖ You can use the code provided during the tutorial or lecture sessions to get started.

Extra Credit (10% Points)

A more elaborate application (including timer based input, genre based recommendation, use of detailed user database, *etc.*) at grader's discretion.

Evaluation Procedure

You MUST demonstrate your program to the grader during the scheduled demo time. You must run your submitted code, demonstrate its full functionality and answer questions about the programming aspects of your solution. Major marking is done on the spot during demonstration. Your code will be further checked for structure, non-plagiarism, *etc.*

However, ONLY the demonstrated submissions will receive marks. Other submissions will not be graded. **THIS ASSIGNMENT MUST BE SUBMITTED INDIVIDUALLY.**

HINTS:

1. Use a struct to represent movies and users for better organization.
2. Implement separate functions for user registration, movie rating, and recommendation.
3. Include error handling for file operations (e.g., opening, reading, and writing files).
4. Encourage modular programming and code reusability.

PITFALLS:

1. Be mindful of file handling errors; check for file existence before reading.
2. Avoid hardcoded values where possible; use constants for better maintainability.
3. Encourage modular programming and code reusability.

Considerations for the Assignment

When tackling this assignment, it's crucial to recognize that if you were implementing this application in a language like Python, the task would be relatively straightforward. Python offers convenient functions simplifying the development process.

However, in C, complexity increases. I want you to focus on understanding the structure of C and its distinctive features, without getting entangled in obscure issues or common pitfalls. Here are some guidelines to navigate through the intricacies:

- Start Small: Begin with manageable tasks. For instance, you might start by displaying a simple menu, then progress to reading input files, then saving them, and so forth.
- Frequent Compilation: Compile your code often to identify what's working and address current issues promptly. Delaying resolution is not a good idea.
- Library Function Reference: Refer to cppreference.com for information on standard C library functions, including clear examples of their usage.
- Segfaults and Pointers: When dealing with segmentation faults due to pointer issues, avoid relying solely on print statements. Instead, leverage the Valgrind system included with the Docker image for effective debugging.
- Simplicity in GUI: The GUI has been kept simple. Utilize the scanf function for reading keyboard input, and be mindful of newline characters when capturing user input.
- Reading and Writing Files: Reading input files involves using fopen in read mode ("r") and fgets/fscanf to read lines as strings. When writing output files, use fopen in write text ("wt") mode and fputs/fprintf to write data to the text files.
- Pointer Usage: Expect to use pointers. Be very careful with pointer operations, such as, declarations, dereferencing, *etc*. Note that a variable holding a pointer is typically declared as int *foo. Here, foo is a pointer to an int (or possibly an array of ints). So foo[3] could be used to access third int in such an array. It is also possible to dereference a pointer, which means that we are getting the data that the pointer references. So syntax like int x = *foo implies that the int pointed to by foo will now be assigned to x.
- Global Variables: While global variables are generally avoided, they may be convenient in this application. Declare shared global variables appropriately across multiple source files. When multiple source files are being used, the compiler must be told that a shared global variable is actually defined somewhere else. For example, if file1.c contains a global variable defined as int foo = 4; then file2.c would include a declaration like extern int foo. This tells the compiler that foo variable declared in file2.c is not a new variable with the same name (which would produce a compiler error) but is just a reference to the foo variable already defined and initialized in file1.cbs
- Header Files: When multiple source files are used, header files should be employed in order to provide prototype information for the compiler. Never include variables or function implementations in a header file. NEVER! They are primarily used for function prototypes, constants, and things like struct definitions.
- #define for Constants: Use #define for fixed values or constants used throughout the program.

- Dynamic Memory Allocation: Use malloc for dynamically creating data, and free for deallocating memory. Ensure proper matching of malloc and free. When de-allocating memory, you will the free function. To be clear, only use free with a matching malloc call. So a char array declared as char *buffer = (char*) malloc (10) could later be deallocated with free(buffer). NEVER try to deallocate a char buffer (or any array) defined like this: char buffer[10]. malloc was not used to create this and its memory is fully managed by the language environment.
- Structs and Pointers: Understand the distinction when working with structs and pointers. Be cautious when dealing with dynamically allocated memory for structs. If a struct is defined like struct foo {int x; inty;}; then we can declare a variable of this type as struct foo bar. We can now use bar.x or bar.y to access the internal values. But if we use malloc to dynamically create space for a new struct foo value, like struct foo *bar = (struct foo *)malloc(sizeof(struct foo)), then we must use the syntax bar->x or bar->y to access the value since we are now using a pointer to the foo struct.
- Pointers to Pointers: Recognize the use of pointers to pointers, especially when creating arrays of strings dynamically. Using pointers is crucial in C but it can be confusing at first. A couple of things to keep in mind: If we use malloc to create an array (e.g., of ints or chars), we can just use the standard array syntax (e.g., foo[4]) to access the elements of the array. The C compiler already knows that the array variable is a pointer. It is also possible to have pointers to pointers. So if we use malloc to create an array of strings called foo, this would actually consist of a pointer to the main array, which would then contain pointers for the strings. So foo would be defined as char **foo (or char* *foo). In other words, foo is pointer to an array of string pointers. This can be confusing at first and can lead to lots of unexpected segmentation faults if you are referencing the wrong pointer. Moreover, if you are deallocating this memory, you have to be REALLY careful. Specifically, the strings must be deallocated first, and then the main array. And, just to be clear, free'ing the main array does not automatically free the memory used by the strings.

This comprehensive guide aims to help you navigate the challenges of implementing the movie recommendation system in C. Follow these principles to build a robust and well-structured solution.

IMPORTANT!!!

Markers will receive a straightforward spreadsheet containing the specified criteria outlined earlier. There are no undisclosed or hidden requirements. It is emphasized that having a functional version of a somewhat limited program is preferable to having a non-functional version attempting to cover every aspect. Therefore, prioritize ensuring that your code can be successfully compiled and executed. Evaluating an assignment that doesn't run at all is virtually impossible.

Sample Run

```
***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 2

***** Movie Database *****
 1. The Shawshank Redemption (Drama) - 9.3
 2. The Godfather (Crime) - 9.2
 3. The Dark Knight (Action) - 9.0
 4. Pulp Fiction (Crime) - 8.9
 5. The Lord of the Rings: The Return of the King (Adventure) - 8.9
 6. Forest Gump (Drama) - 8.8
 7. The Matrix (Action) - 8.7
 8. The Silence of the Lambs (Crime) - 8.6
 9. Schindler's List (Biography) - 8.6
10. Inception (Action) - 8.6

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 1
Enter username for registration: kay
User already exists. Please choose a different name.
Enter username for registration: sam
User already exists. Please choose a different name.
Enter username for registration: Ravi
User Ravi is successfully registered.

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 3
Enter your username: ra
User not found. Please register first.

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 3
Enter your username: ravi

***** Movie Database *****
 1. The Shawshank Redemption (Drama) - 9.3
 2. The Godfather (Crime) - 9.2
 3. The Dark Knight (Action) - 9.0
 4. Pulp Fiction (Crime) - 8.9
 5. The Lord of the Rings: The Return of the King (Adventure) - 8.9
 6. Forest Gump (Drama) - 8.8
 7. The Matrix (Action) - 8.7
 8. The Silence of the Lambs (Crime) - 8.6
 9. Schindler's List (Biography) - 8.6
10. Inception (Action) - 8.6
Enter the number of the movie you want to rate: 5
Enter your rating (1-5): 4
Rating recorded successfully.

***** Movie Recommendation System *****
1. Register User
2. Display Movies
3. Rate a Movie
4. Get Movie Recommendations
0. Exit
Enter your choice: 3
Enter your username: ravi

***** Movie Database *****
 1. The Shawshank Redemption (Drama) - 9.3
 2. The Godfather (Crime) - 9.2
 3. The Dark Knight (Action) - 9.0
 4. Pulp Fiction (Crime) - 8.9
 5. The Lord of the Rings: The Return of the King (Adventure) - 8.9
 6. Forest Gump (Drama) - 8.8
```

Grading Rubric for Assignment 1

Total	**100 pts**
Register a User	**20 pts**
Check username for existing users in the database	10 pts
User registration	10 pts
Display Movies	**10 pts**
List the movies with required formatting	10 pts
Rate a Movie	**15 pts**
Validation of user before rating a movie	5 pts
Validating movie index and rating value	5 pts
Updating the ratings database	5 pts
Get Movie Recommendations	**15 pts**
Computations for movie recommendations	10 pts
Displaying the recommended movies	5 pts
Exit	**10 pts**
Graceful exit of the application	5 pts
Deallocation of all the resources	5 pts
Demo QnA by the Grader	**30 pts**