Table of Contents

Introduction

The software is mainly designed to manage Movie DVDs of a community library. The design is mainly object-oriented approach having four classes. This system mainly has 2 types of users “Staff” and “Member”. Staff is mainly responsible for functions like adding and removing of movies, adding and removing of members and also searching for members. The members added by staff are given a unique password which can be used by the member to sign in the system and access its facilities. Members are verified by the First name, Last name and Password. Members can borrow or return movies, check details of a movie by searching them with the movie title and can see list of top 3 movies that are mostly being borrowed by members. The system uses hash table data structure to store movie and array to store member details. An algorithm is also designed to maintain the list of top 3 movies efficiently. Both movie title and member names are capitalized for consistency of system. Below sections will describe more about the data structure and the algorithm in detail.

System Design

The overall system consists of mainly 4 classes namely *Movie, MovieCollection, Member and MemberCollection*. Each class has its own attributes, properties and methods to complete the whole system. To access the attributes, properties are used to maintain proper encapsulation and validation. The attributes and methods are explained below (attribute and method names are in bold and class name in italic-bold):

Movie class

Attributes:

NAME	DETAILS
title, genre, classification, duration	String type. Used for storing basic movie details.
currentNumberOfDVDs	Integer type. Used for the number of DVDs that are available in the system which can be borrowed by members. Whenever staff adds a movie, they need to add a number of DVDs to the system, and this is stored in this attribute. But this number cannot be more than **maximumNumberOfDVDs.**
maximumNumberOfDVDs	Integer type. Used to denote the maximum number of DVDs that the system can have for a particular movie. Right now, it is caped to 10. It means the system cannot have more than 10 DVD copies of the same movie.
numberOfDVDsCurrentlyBorrowed	Integer type. Keeps count of how many DVDs are currently borrowed by members.
nameOfMemberCurrentlyBorrowingThisMovie	String type array. Stores the full name of the members who are currently borrowing that particular movie.

totalNumberOfBorrowedFrequency	Integer type. Keeps the count of how many times the movie has been borrowed by any member since the movie was added to the system. The movie that has the highest value of this attribute is considered as the most borrowed movie in the system.

Methods:

NAME	DETAILS
AddMemberFullnameToArrayMemberCurrentlyBorrowingThisMovie(string memberFullName)	Void type method where member's full name is passed as reference to save member full name in the array **nameOfMemberCurrentlyBorrowingThisMovie**. Stores full name of the members who are borrowing that particular movie.
RemoveMemberFullnameFromArrayMemberCurrentlyBorrowingThisMovie(string memberFullName)	This method does exactly opposite of previous method. It removes the name of the member who returns the movie back to system after borrowing.
PrintMembersNameBorrowingMovie(string userInputTitle)	This method mainly prints the elements of array **nameOfMemberCurrentlyBorrowingThisMovie**. And the reference is used to show the title of the movie in console output.

Constructors:

NAME	DETAILS
public Movie()	Default constructor used for hashtable in ***MovieCollection*** class.
public Movie(Movie top3movies)	This constructor is used for **top3MoviesArray** in ***MovieCollection*** class. The main reason to create two different constructors is to make sure any change done to default constructor's reference does not affect the other. A ***Movie*** type object is passed to this constructor.

MovieCollection class

Attributes:

NAME	DETAILS
table	A ***Movie*** type array. Used as the main hashtable of this system. ***Movie*** *objects* are stored in this hashtable.
count	Integer type. Used to keep track of current elements in **table.** Whenever a movie is added to the **table** this variable is incremented.
buckets	Integer type. Used to denote the maximum number of movies that can be stored in the **table**. Right now, it is caped to 1000.
deleted	String type. Used to specially mark indexes in the **table** where previously a movie was added but it is now deleted by staff. The string sentinel value "Movie Deleted By Staff" is used to denote deleted movie from the **table**.

top3MoviesArray	String type array. Stores the ***Movie*** objects which have the highest number of borrowed frequencies. Top 3 highest ***Movie*** objects are stored here.

Methods:

NAME	DETAILS
Hashing(int key)	Returns integer value after performing the division hash function. As input, it takes the int value that is obtained from movie title.
ConvertStringTitleToIntKey(string title)	Returns an integer value after converting the movie title to an accumulated integer value.
FindInsertionBucket(string stringKeyMovieTitle)	Method mainly returns the index or key where the movie will be inserted in the **table** after calculating all the hash functions.
Insert(string key, Movie value)	Method uses the title of the movie and a ***Movie*** object to insert into the **table.**
SearchUsingTitleInputToDisplatInformation(string userInputTitle)	Uses the movie title to search for the movie if it is present in the **table** by calculating hash functions and returns the int index value where the ***Movie*** object is stored.
DeleteWholeMovieInformationUsingTitleInputByUser(int bucketKey)	This method uses the index value where the movie is stored in the **table** and deletes the movie. It replaces all the string values of ***Movie*** object to **deleted** sentient value.
UpdateNumberOfDVDsOfExistingMovie(int indexNumberOfBucketContainingMovieObjectDetails, int newNumberOfDVDs)	Updates the current number of DVDs if staff wants to add new number of DVDs to an already existing movie.
DeleteNumberOfDVDsOfExistingMovie(int indexNumberOfBucketContainingMovieObjectDetails, int toBeDeletedNumberOfDVDs)	Method is called when staff wants to decrease current number of DVDs that is available for borrowing.
PrintAllMovies()	Simply print the elements of **table** containing movies.
PrintTop3Movies()	Prints the elements of **top3MoviesArray.**
FindTop3Movies(int indexOfBucketHoldingMovie)	This method is the algorithm calculating which movies has the most borrowed frequency from the **table** containing all the movies. The input is the index number of the ***Movie*** object that is stored in the hash **table.**

More details on the hash-table and the algorithm are provided in next sections.

Member class

Attributes:

NAME	DETAILS
firstName, **lastName**, **fullName**, **phoneNumber**, **password**.	String type. These attributes store basic information of a ***Member*** object. The **firstName, lastName** and **password** will be used for member login which will be assigned by staff while adding member.
maxNumberOfBorrowedMovies	Integer type. Denotes max number of movies that a member can borrow. Currently it is initialized as 5 because a single member cannot borrow more than 5 movies at a time.
countBorrowedMovies	Integer type. keeps count of how many movies member is currently borrowing**.**
arrayToStoreTitleOfBorrowedMovies	String type array. Stores the names of movies member is borrowing.

Methods:

NAME	DETAILS
AddBorrowedMoviesNameToArray(string inputTitle)	Method takes movie title as input and adds the title in **arrayToStoreTitleOfBorrowedMovies** only if **countBorrowedMovies** value is less than **maxNumberOfBorrowedMovies.**
SearchArrayOfBorrowedMovies(string movieTitle)	This method performs linear search in **arrayToStoreTitleOfBorrowedMovies** by using movie title to see if the movie is already in that array**.** This returns the index of the movie if it exists in the array.
PrintListBorrowedMovies()	Prints **arrayToStoreTitleOfBorrowedMovies** array elements.
RemoveMovieTitleFromArray(int index)	It takes the index number of movie title as an input from **SearchArrayOfBorrowedMovies.** Then the index is passed here to remove the movie title from array **arrayToStoreTitleOfBorrowedMovies.**

MemberCollection class

Attributes:

NAME	DETAILS
arrayToHoldMemberObject	***Member*** type array. Used to store the *Member* object.
maxmumCapacity	Integer type. Denotes the max size of **arrayToHoldMemberObject** array. It is set as 1000 now.
currentMembersCount	Integer type. Keeps track of how many members are currently added in the array **arrayToHoldMemberObject.**

Methods:

NAME	DETAILS
InsertMemberInArray(string fullNameOfMemberToBeAdded, Member objectHoldingMemberDetails)	Adds ***Member*** object in the array **arrayToHoldMemberObject.** This method adds the member based on ascending order of the member's full name. So, member's full name and ***Member*** object has to be passed using the parameter.
SearchIfMemberAlreadyExist(string staffInputOfMemberFullName)	Performs linear search to check if the member already exists in the array. If exists it returns the index value.
PrintMemberList()	Prints all the elements of **arrayToHoldMemberObject** array.
CheckPasswordMatch(int indexNumberOfElementFullnameMatchingWithInputFullname, string passwordInputByUser)	Checks if the password in array **arrayToHoldMemberObject** matches with password that user has entered. After getting the index value from **SearchIfMemberAlreadyExist** it is passed to this method along with the password that user has entered.
DeleteMember(int index)	It deletes the member from array **arrayToHoldMemberObject.** **SearchIfMemberAlreadyExist** returns an index number and then it is passed as reference to this method.

Hash-table Data Structure

Hash table is an efficient data structure that can perform tasks like insertion, deletion and search with a time complexity of O(1). So, for large data sets hash table is a good data structure to store and retrieve data. Hash table has hash functions that can assign unique id to each movie title. It is mainly a collection of key-and-value pairs. Information can be searched from hash table using the key. Other data structures like linked list is not efficient because insertion, deletion and search time increases as the problem size grows. In my system I am using an array that holds key pair value of ***Movie*** object. The integer value got from the title is used here as the key and the other details like movie duration, genre, classification etc is the value of the table. To get an integer key from the title, I used ASCII values to get the integer equivalent value of the string title.

```
2 references
public int convertStringTitleToIntKey(string title)
{
    int hashValue = 0;
    for (int i = 0; i < title.Length; i++)
    {
        int result = (int)title[i];
        hashValue += result;
    }
    return hashValue;
}
```

This code snippet explains the method that is used to get integer key value from movie title. The title is passed as the parameter. Then for each character the ASCII value is added to **hashValue** variable which is initialized as 0. For example, the integer key value of "TENET" movie will be 84+69+78+69+84 = 384. For keeping the system constant, the movie titles are converted to upper case letter after the user input is complete. This is done using String.ToUpper() method.

Division Method

To get the key, the simplest and yet one of the most popular hashing methods called division method is used. This is done mainly by dividing the key by a number "M" and using the reminder as the hash value. "M" can be any value depending on the needs. To keep things simple for this system, value of "M" is the max capacity of the hash table which is 1000. And the key is the integer value of the title. A code snippet is given below referring to division method.

```
2 references
private int Hashing(int key)
{
    return (key % buckets);
}
```

buckets represent the maximum capacity of the hash table.

Hash Table Key Collisions

Collision in hash table is a very common scenario as keys can have same integer value in many scenarios. In order to handle collisions, there are two common techniques "Open addressing" and "Separate chaining". In open addressing the array is searched in a systematic way to find empty cell without visiting all the available cells of array. But in separate chaining an array of linked list is created. In my system open addressing is used to handle collisions.

Open Addressing

Open addressing has three different types of probing method "Linear probing", "Quadratic probing" and "Double probing". Each of these methods has different advantage and disadvantages. In linear probing if collision occurs it starts incrementing

linearly till the next vacant cell is found. Linear probing can cause primary clustering problem where large number of records clusters around collision site. In quadratic probing, if collision occurs it searches for the next vacant cell by squaring the step number. Quadratic probing removes the problem of primary clustering but brings another problem named secondary clustering as it always follows the same sequence to find a vacant cell. So, double probing is the best collision handle method for this system which will reduce clustering problem dramatically.

Double probing

Double probing uses two has functions. The first one is the default division method and after finding a collision another hash function is called which determines the step size. The most popular second hash function is hash2(key) = PRIME – (key % PRIME). The prime number has to be smaller than the table size. The idea is to pick up a prime number that is half the max size of the table. A code snippet is given below including the second hash function.

```
1 reference
private int FindInsertionBucket(string stringKeyMovieTitle)
{
    int key = ConvertStringTitleToIntKey(stringKeyMovieTitle);// we get the key by converting the title to its respective ascii values
    int bucketNumber = Hashing(key);
    int i = 0;
    int offset = 499 - (key % 499);// second hash function for double probing
    while((i<buckets) && (table[(bucketNumber+(i*offset))%buckets]!=null) && (table[(bucketNumber + (i * offset)) % buckets].Title != deleted))
    {
        i++;
    }
    return (bucketNumber + (i * offset)) % buckets;
}
```

Offset value is found after calculating the second hash function 499 –(key%499). 499 is chosen because it is close to half of the total table size. For each iteration the value of i increments by 1. Double probing is considered the best probing method because it produces a uniform distribution of records in the hash table.

Algorithm Description and Analysis

Algorithm Description

Input: The input of the algorithm is the index number or the key that is gathered from the hashtable. The algorithm is called every time after a new movie is added or borrowed. For the insertion and borrowing functionalities the hash key is always calculated so this algorithm does not need to calculate hash key by its own and the index number is simply passed to this algorithm. Now the algorithm can read the updated borrowed frequency number of that ***Movie*** object.

General Description: A ***Movie*** type array called **top3MoviesArray** is used to keep track of the movies that has the highest borrowed frequency. The array has a size of 3 starting from index 0. The algorithm mainly has two parts. In the first part, the algorithm checks if

the reference ***Movie*** object matches with any of the three ***Movie*** objects in the array by executing a for loop. If it finds any match the **flag** variable becomes 1 and second part never executes. Second part is only an if-else condition. It only executes if **flag** variable is 0 which means no matches were found in the for loop of first part. Inside both the parts the comparison is almost same.

The rest functionality is very simple. For the first part, after the reference ***Movie*** object is matched with any one of the three elements from the array, it compares the **totalNumberOfBorrowedFrequency** of that object with its previous index's object. If it is larger than the previous object's value, the previous index's object is sent to the current index and previous index's object is swapped with the reference object keeping the object with highest value in top. The functionality is same for the second part as well.

In the second part, If the reference object's **totalNumberOfBorrowedFrequency** is larger than 2nd index it checks all the way up to 0th index until the reference object's value become smaller than any of the object's value in the array. Then it swaps the objects. The object with highest value starts taking place in the array from 0th index maintaining a descending order of the integer value **totalNumberOfBorrowedFrequency.**

Output: This algorithm mainly manipulates the array **top3MoviesArray** by Swapping places of ***Movie*** objects within the array to maintain descending order by the value of **totalNumberOfBorrowedFrequency.**

Example: So, let's assume three movies "TENET", "DUNE" and "AVATAR". Now "TENET" is borrowed for 4 times, "DUNE" for 2 times and "AVATAR" for 1 time. So, they are in the array **top3MoviesArray** maintaining descending order. So, when "AVATAR" will be borrowed by a member the number will increase to 2 times. But it will still be in 2nd index of the array. But if the movie is borrowed again than it will swap its place with previous index which is "DUNE". So, new array elements will be "TENET", "AVATAR" and "DUNE". Now let's say another movie "INCEPTION" is borrowed only once. So, this movie is not in the array **top3MoviesArray**. Now if the movie is borrowed again the borrowed number will be 2. Now "DUNE" and "INCEPTION" both are borrowed for 2 times, but "DUNE" is already in the array. So, "INCEPTION" will not replace "DUNE" because both has the same number of borrowed frequencies. The condition is set to work only if "INCEPTION" borrowed frequency gets larger than any of the elements inside the array. Now, if "INCEPTION" is borrowed again, its borrowed frequency will go to 3 and gets larger than "DUNE". So, "INCEPTION" will take place of "DUNE" in the array.

Algorithm Pseudocode

Array = Stores the top 3 ***Movie*** objects that has the highest borrowed frequency.
table = hashtable containing all the ***Movie*** objects.
Frequency = The number of times movie has been borrowed.
index = Index of ***Movie*** object from hashtable.

```
FindTop3Movies(integer index)
Set flag ← 0 // denotes if reference object matches with the objects inside array
for i ← 2 to i ← 0 step -1 // reverse loop. Iterate from index 2 to 0
        if Array[i].Title = table[index].Title
        Set flag ← 1; // reference object title matches with an object title in array
            if i = 2 // reference matches with index 2. Next, compare with previous index.
                if table[index].Frequency > Array[i-1].Frequency//compare with index 1
                    if table[index].Frequency > Array[i-2].Frequency//compare with index 0
                        Array[i] ← Array[i-1];//swap index 2 with 1
                        Array[i-1] ← Array[i-2];//swap index 1 with 0
                        Array[i-2] ← table[index];//replace index 0 with reference
                        break;
                    else
                        Array[i] ← Array[i - 1];//swap index 2 with 1
                        Array[i-1] ← table[index];//replace index 1 with reference
                        break;
                else
                    Array[i] ← table[index]; //replace index 2 with reference
                    break;
            else if i=1 // reference matches with index 1. Next, Compare with previous index.
                if table[index].Frequency > Array[i-1].Frequency //compare with index 0
                    Array[i] ← Array[i-1]; //swap index 1 with 0
                    Array[i-1] ← table[index]; //replace index 0 with reference
                    break;
                else
                    Array[i] ← table[index]; //replace index 1 with reference
                    break;
            else // reference matches with index 0
                Array[i] ← table[index]; //replace index 0 with reference
                break;
END for loop
if flag = 0 // reference object does not match with any object in array.
        if table[index].Frequency > Array[2].Frequency // compare with index 2
            if table[index]. Frequency > Array[1]. Frequency  // compare with index 1
                if table[index]. Frequency > Array[0]. Frequency // compare with index 0
                    Array[2] ← Array[1]; //swap index 2 with 1
                    Array[1] ← Array[0]; //swap index 1 with 0
                    Array[0] ← table[index] //replace index 0 with reference
                else
                    Array[2] ← Array[1]; //swap index 2 with 1
                    Array[1] ← table[index]; //replace index 1 with reference
            else
                Array[2] ← table[index]; //replace index 2 with reference
END FindTop3Movies
```

Algorithm Analysis

1. The algorithm is independent of input size as the input is just an integer value. This value mainly denotes the index or bucket key of the **hashtable.** By using the index, the ***Movie*** object is accessed from the hashtable and put into the algorithm for different comparisons with the array that stores the top 3 movie based on their number of borrowed frequencies. As the array has only 3 elements so the loop will always have constant time while doing the comparison.
2. The basic operation in this algorithm is Array[i].Title = table[index].Title. Basically, this operation checks if the ***Movie*** object in table[index] matches with any ***Movie*** object in **top3MoviesArray**. This operation runs exactly for 3 iterations as the array length is 3. This operation has the most influence in this algorithm as it decides the **flag** value. If the value remains 0 it means the objects never matched with each other in 3 iterations and additional if condition has to be checked to update the **top3MoviesArray.**
3. In best case the basic operation runs only one time and in worst case it can run for three times including the additional if-else condition. And in average case it can run till somewhere between best and worst case.
4. The loop can iterate up to three times and the additional if-else condition will have constant time. So,

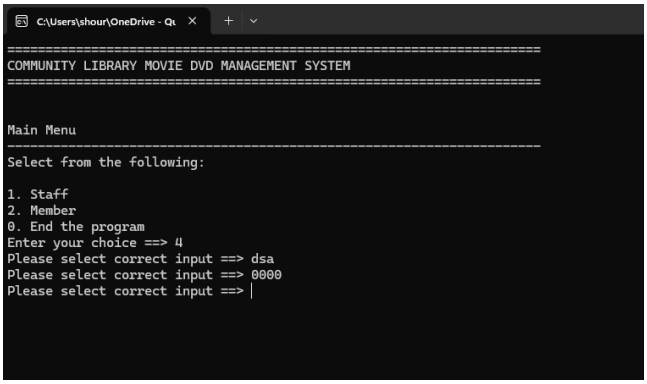
$$\sum_{1}^{3} 1 + 1 = (3 - 1 + 1) + 1 = 4$$

5. So, the efficiency of this algorithm will be Θ(4) or Θ(1) constant time. As this algorithm is independent of input length size, it will always consume constant time.

Test Case

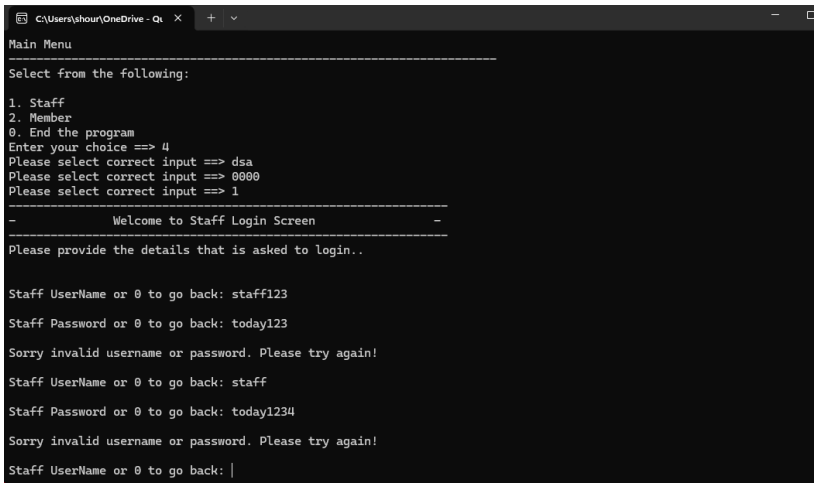
There are 12 total functionalities of this system. 6 functions are for staff and other 6 functions are for members. The system mainly has 3 menu systems main menu, staff menu and member menu. Screenshots of different testing scenarios of the overall system is given here to ensure that the system responds efficiently to any input user makes in this system. All the menus have predefined options. So, user has to select options that is shown on the console. Any input other than that will show a prompt saying given instruction is invalid. The program will continuously run until the sentinel value (0) is entered to exit the application. Various test cases have been demonstrated below. Some Movie names and Members are added to the system as hard coded array to load whenever the code is executed. This is done to help in checking all the functionalities.

Main Menu, Staff Menu and Member Menu



This screenshot shows that if anything is entered other than 1, 2 and 0 the program will keep asking for correct input. Menus for ‘Staff’ and ‘Member’ also work the same way. Other inputs will not be recognised by the system except the required ones.

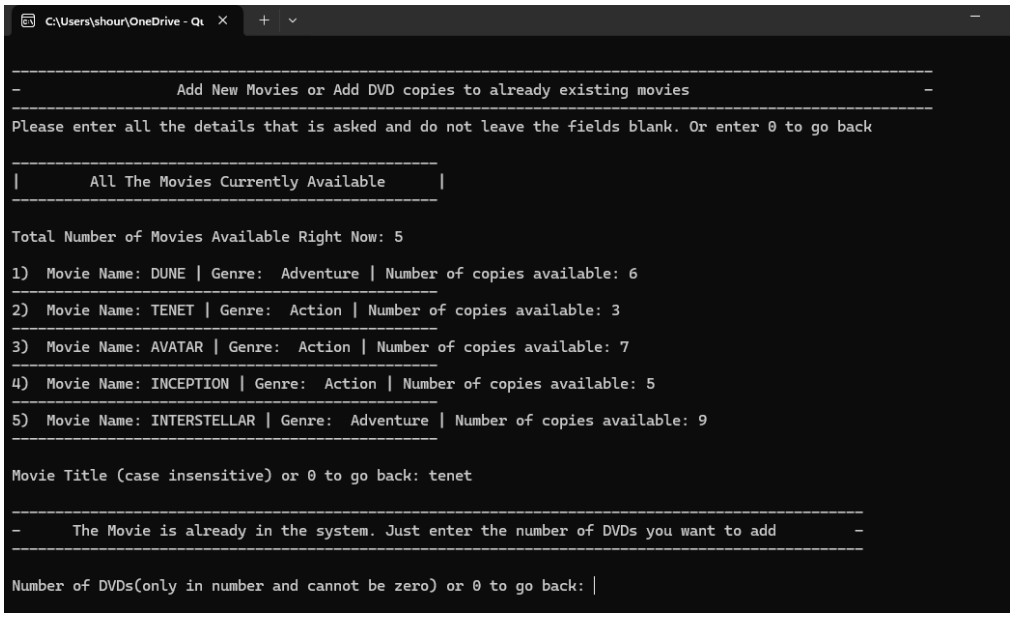
Staff Login



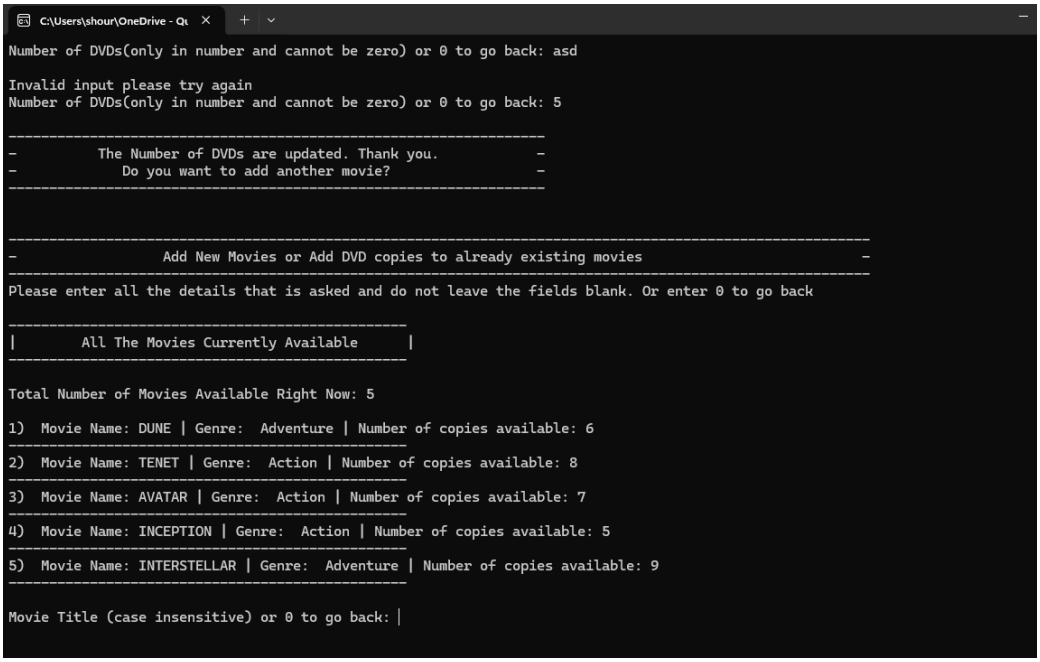
In the above screenshot, the username of staff is “staff” and password is “today123”, so inputting anything other than that will show that username or password is invalid.

Add Movies

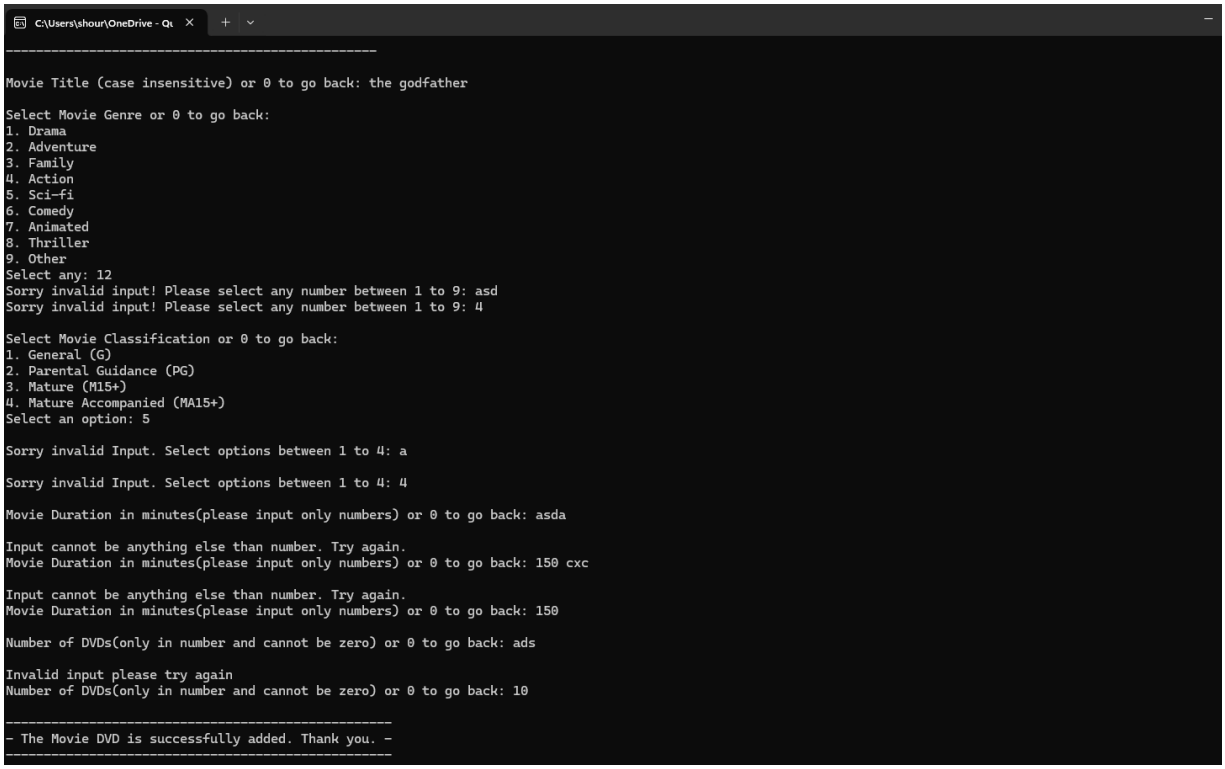
Staff can add Movie DVD to the system. If the movie is not new in system, it will ask how many numbers of DVD staff wants to add. If the movie is new in system, staff will need to put all the details of movie like title, genre etc. While adding number of DVDs it can not be more than 10 as I have already mentioned that. Right now, only 10 DVDs of per movie can exist in system.



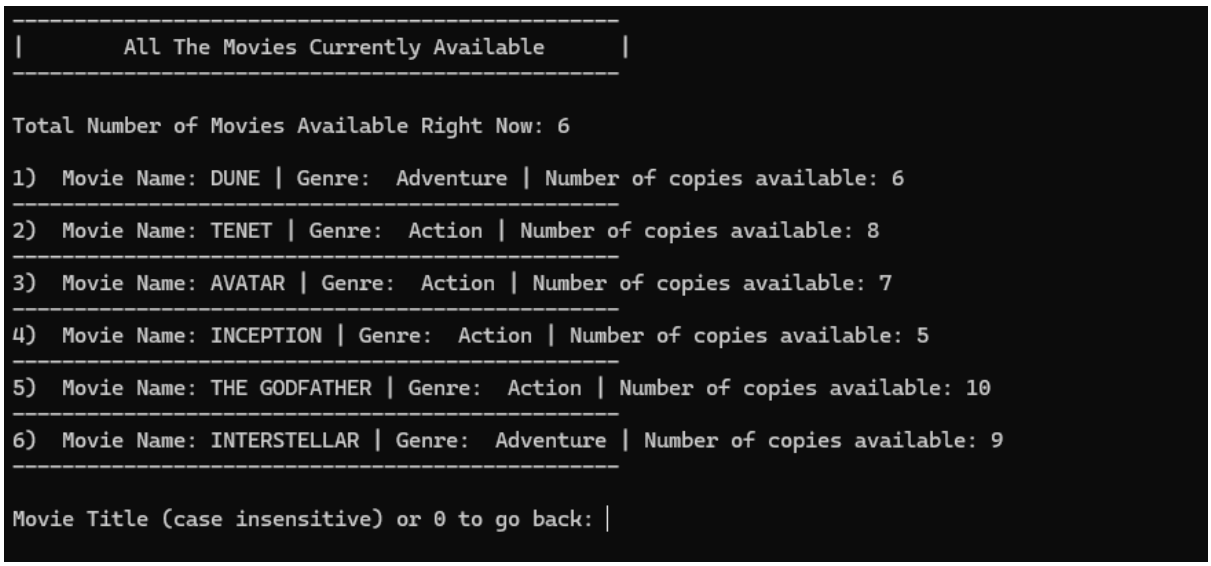
I have preadded some movie. So, when staff again try to add the same movie “TENET”, the systems shows that movie already exist and if staff wanted to add new number of DVDs of the movie. So, the new number will be added to the existing number. Now there are 3 DVDs existing in the system for “TENET”. So, now staff will add 5 more DVDs. It will be 8. Anything other than a valid number will not be counted as correct input.



First, staff input some characters it says invalid input. Then he input number 5. The number of tenet movie is changed. Now staff will add a new movie that does not exist in the system.



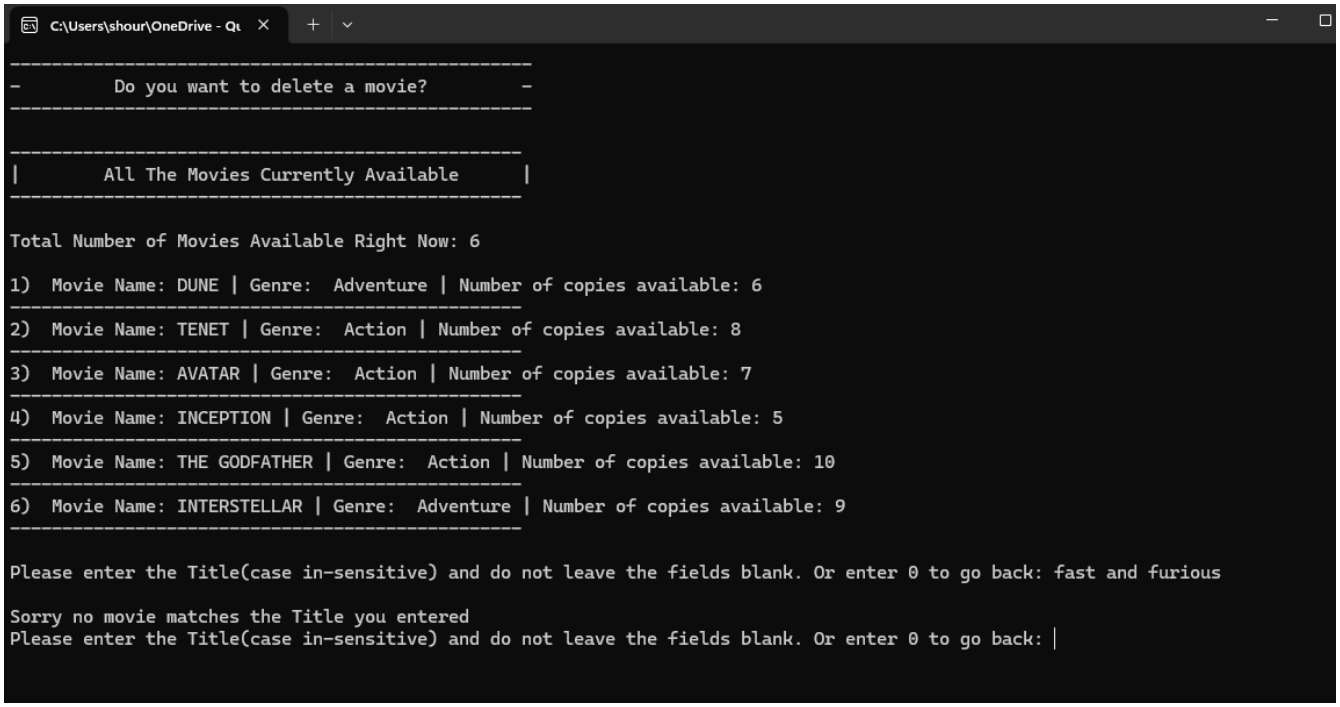
The above screen shows all prompts for adding movie. “THE GODFATHER” movie was not in the system before, so it asks for all the details. While adding genre and classification of the movie staff has to select the given options. Anything else than that will show a prompt and ask for input again. The duration and the number of DVDs have to be number inputs, or it will be considered as wrong input. After doing everything successfully, the movie will be added.



The movie is successfully added to the system.

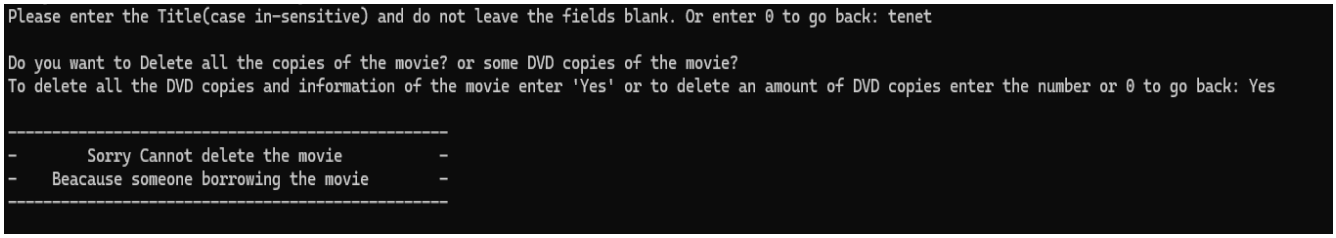
Remove Movie

To remove movies, the movie has to be in the system, or it will say input title does not matches any of the title of existing movies. The below screenshot shows that function.



To remove a movie there are two options. Staff can completely remove the whole movie including all details or can remove some number of DVDs of a movie. While removing the available number of copies, staff has to put a number that is less than currently existing number of DVDs. And if the movie is borrowed by a “Member”, it cannot be completely removed from the system. All test cases are shown below with screenshots.

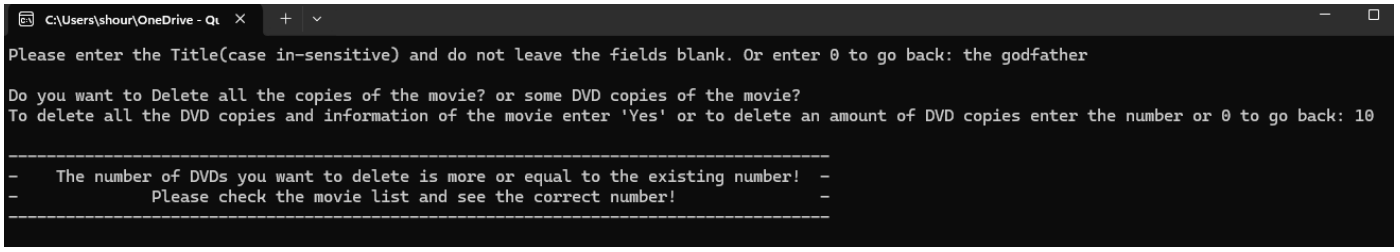
Removing whole movie and all its details:



Staff put “TENET” in input and the system finds the movie. Now system asks if he wants to delete the overall movie from the system or some numbers of DVDs. Staff wants to delete the whole movie. So, he input “Yes”(case-sensitive) and it will delete the whole movie. As “TENET” movie is being borrowed by a member so the whole movie data cannot be deleted from the system.

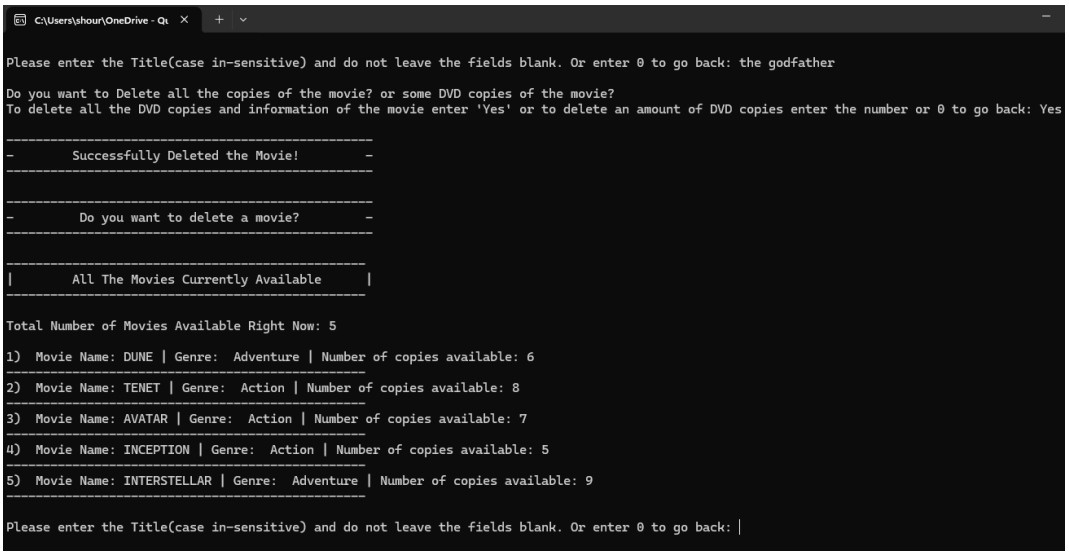
Removing only some DVDs of a movie:

Now to remove some number of DVDs of an existing movie staff needs to enter the number instead of typing “Yes”.



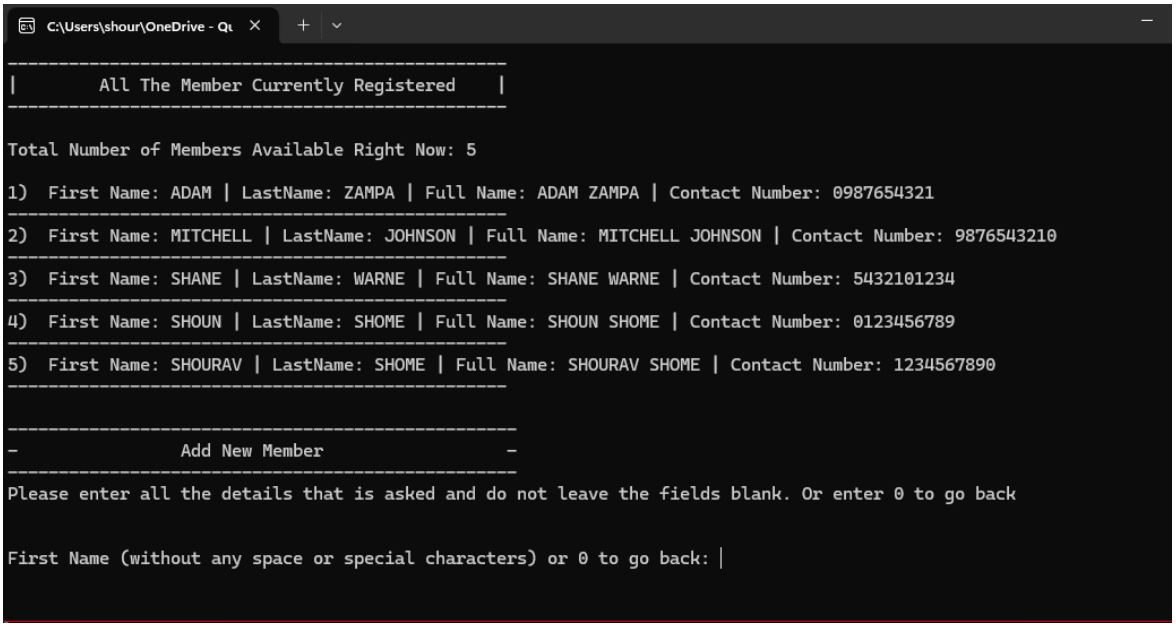
“THE GODFATHER” movie has 10 numbers of DVDs so the “Staff” needs to input less than 10.

Now staff inputs number 8. After deleting 8 number of DVDs, right now only 2 number of DVDs are available in the system for the movie “THE GODFATHER” which can be borrowed. Also to show the functionality of deleting a movie with all its details, staff deletes “THE GODFATHER” and all of its details from the system.



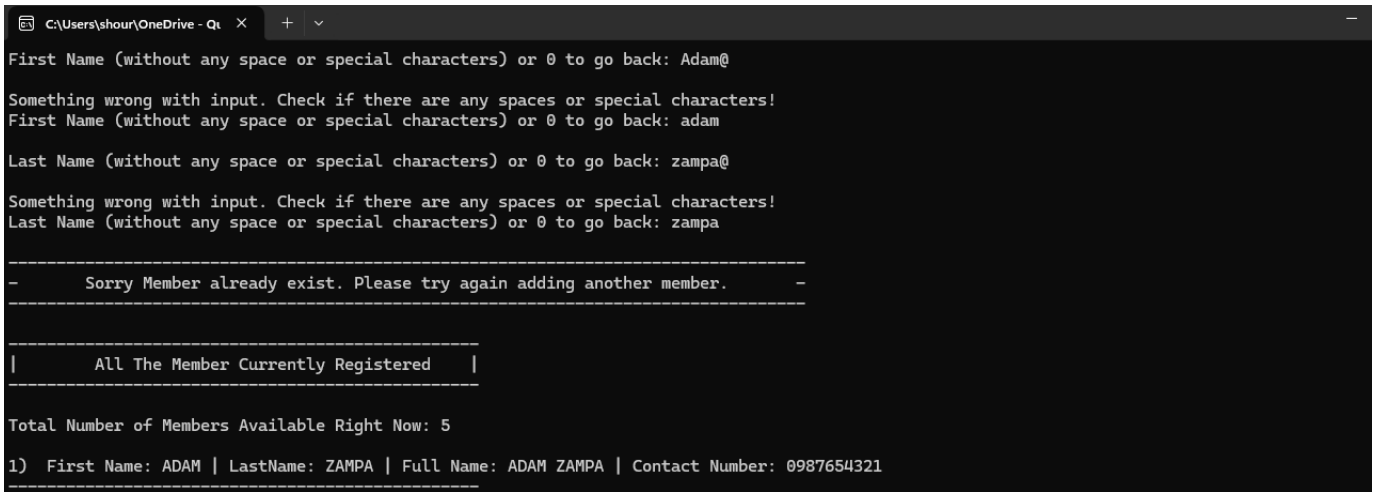
As no one is borrowing “THE GODFATHER” movie, the whole movie can be deleted.

Register/Remove Member



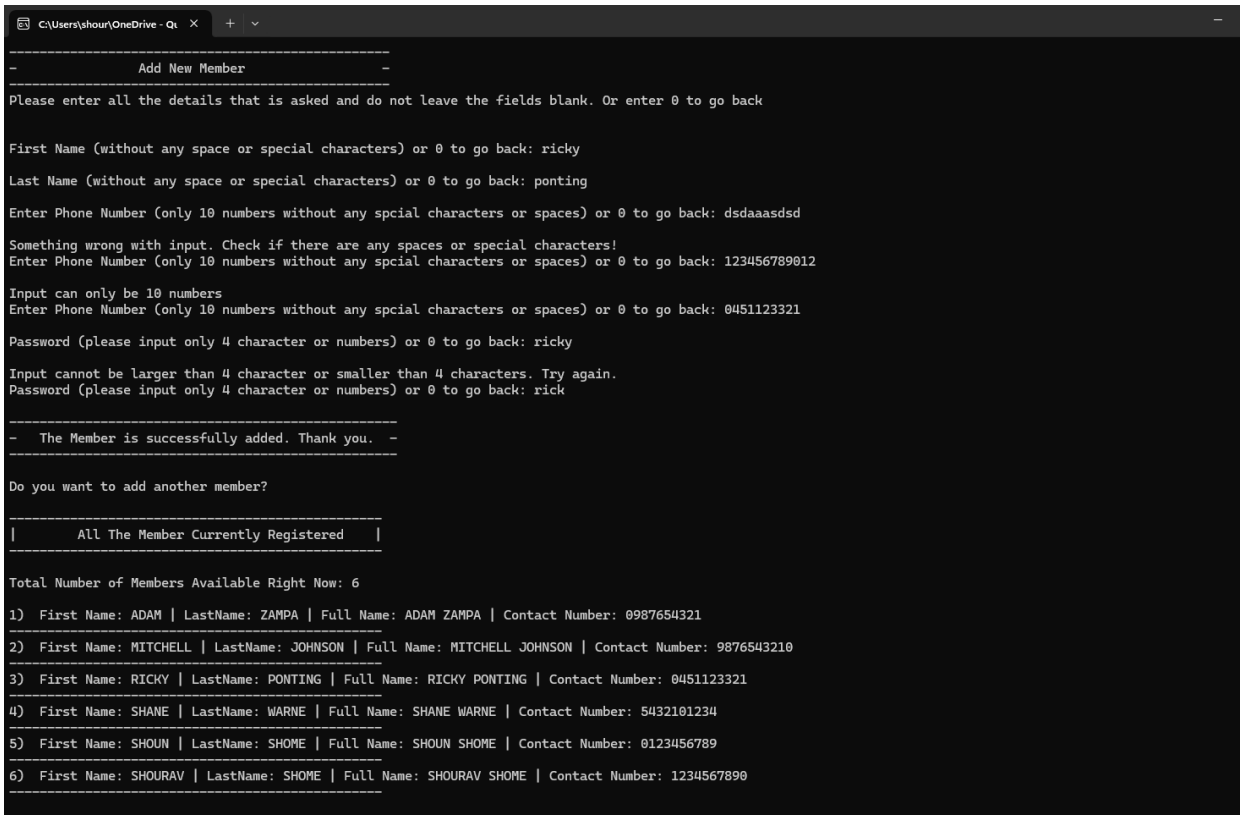
Currently these members are added to the system. After inputting first name and last name the system combines them and get full name. Then checks if the member already exists in the system. During inputting the names staff cannot add any space or any special characters to keep the system simple and consistent. In next screen first staff will try adding already existing member.

Try to add existing members:



The above screenshot shows that no special characters are allowed while inputting the first name and last name. After inputting in correct format “ADAM ZAMPA”, the system finds that the member already exists in the system, so console outputs member already exists. Next screenshot will demonstrate adding new member.

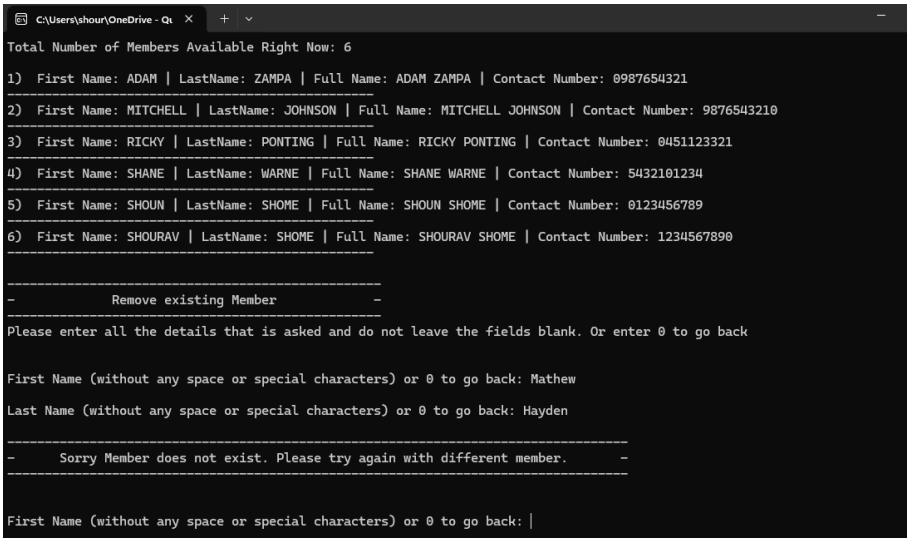
Adding new member:



In the above screenshot it is shown that phone number has to be numbered input and cannot be more than 10 digits and password can be any character but cannot be more than 4 digits as per requirements. The member is successfully added. The next screenshot is about removing member. It also works in same way as adding. After inputting first name and last name the system will check if the member exists in the system.

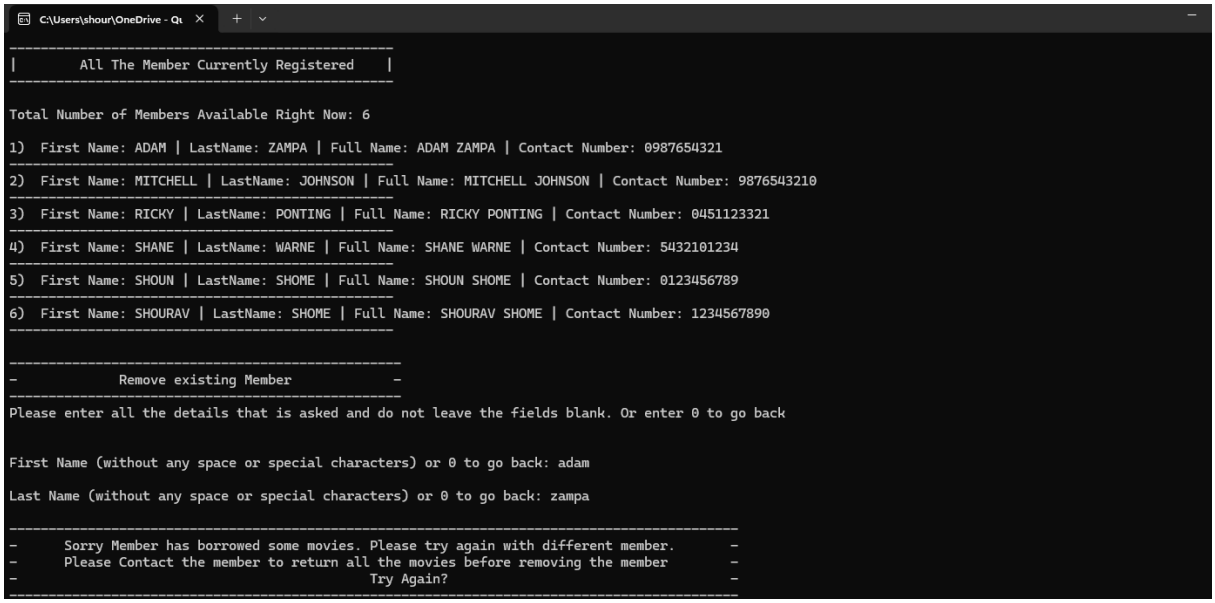
Removing member:

If the member does not exist in the system, it will say member does not exist.

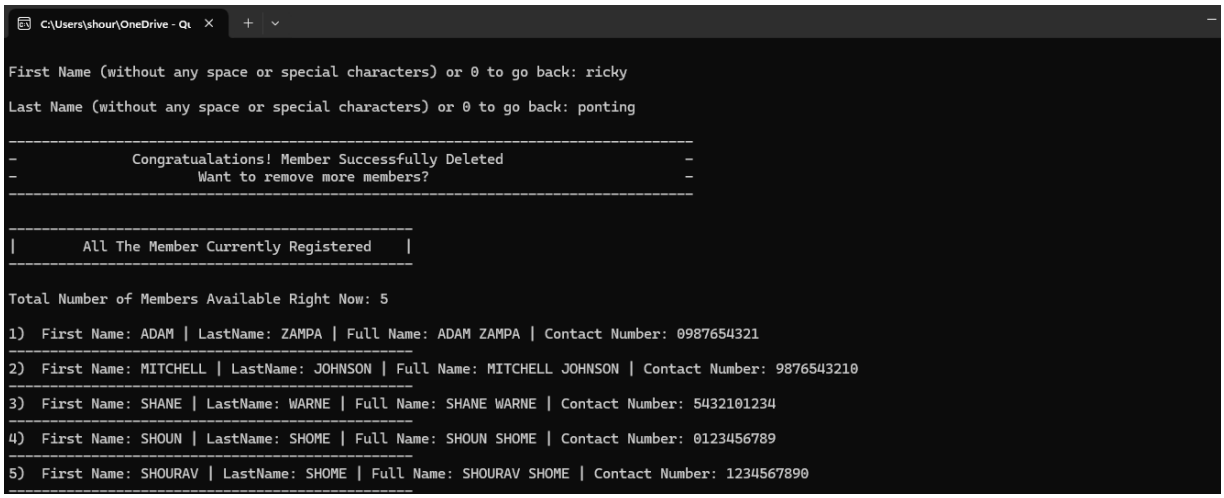


In above screenshot, as no member name “MATHEW HAYDEN” is registered in the system, it says member does not exist.

Deleting Member borrowing any movie:



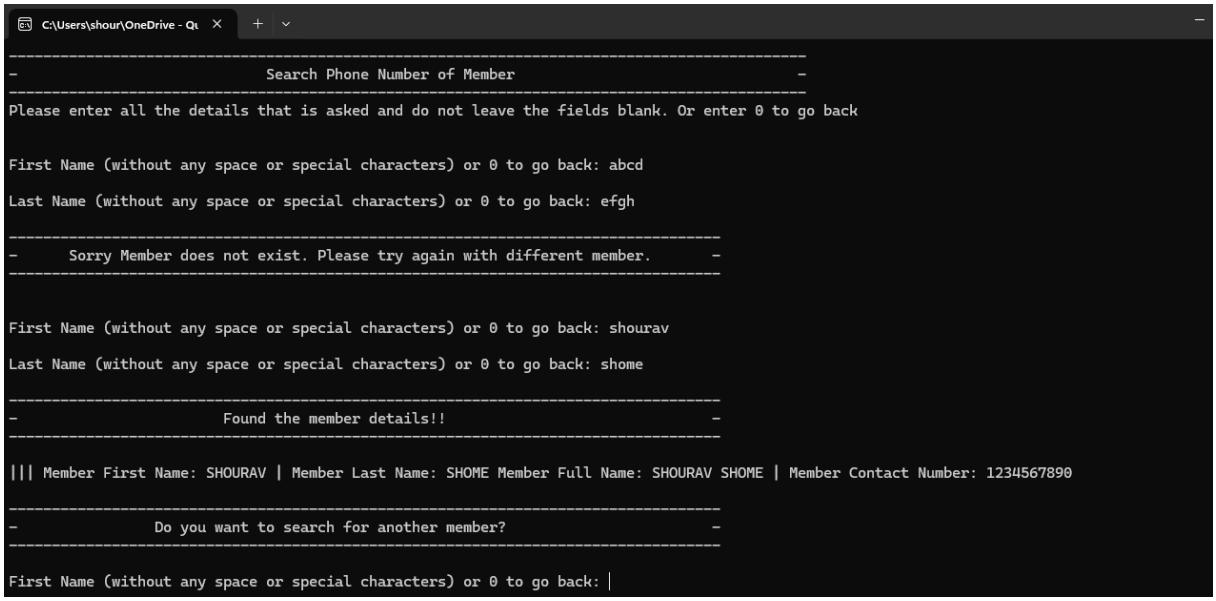
As the member is currently borrowing a movie the member cannot be removed from the system.



As the member is not borrowing a movie the member can be removed from the system.

Searching Member Details

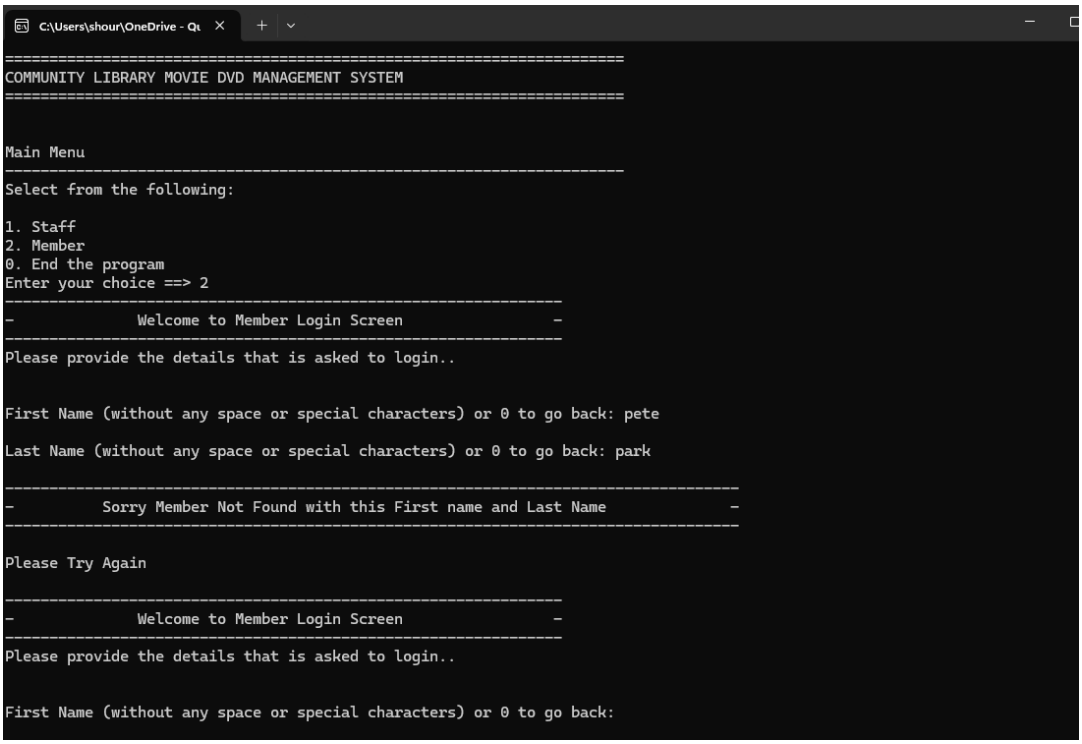
To search for member details staff can use the first name and last name to look for member. If the input does not match with any name, it will show member does not exist. Or else it will show the member details.



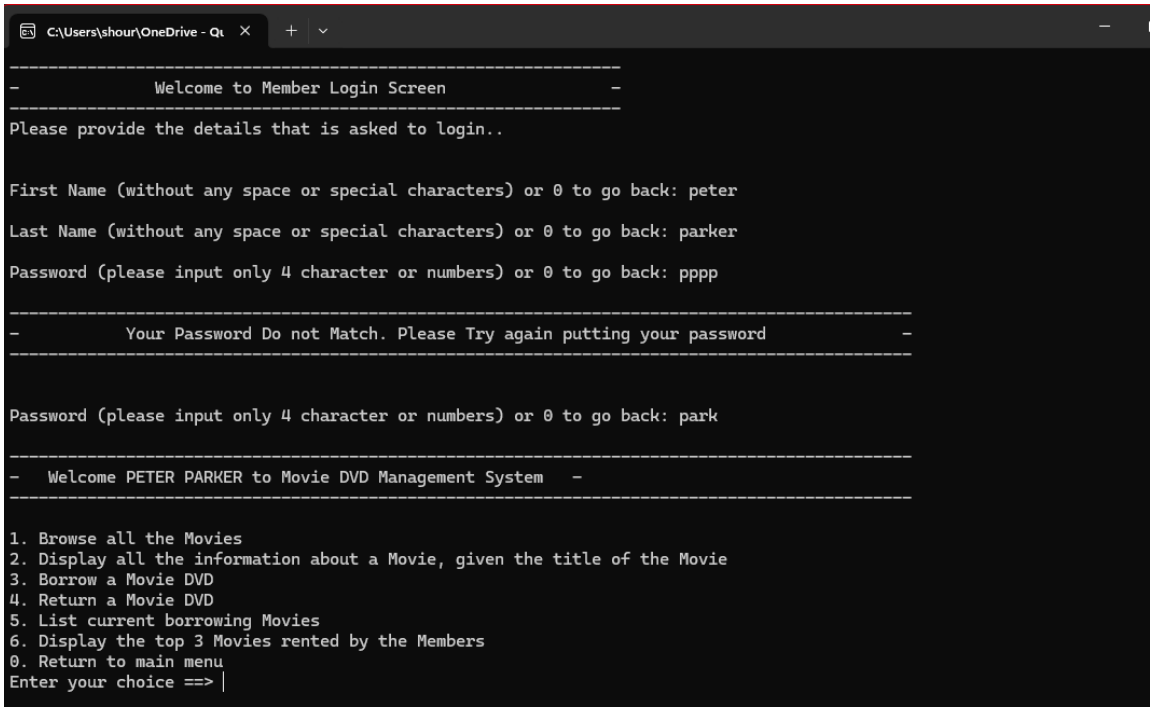
In first input staff put random name “ABCD EFGH”. As, no member exists in this name the system shows an error. But after putting “SHOURAV SHOME” it matches an existing name and shows the details of that member.

Member Login

Now, a new member is added first name: PETER, Last name: PARKER and password: park. The “Member” himself will try to login in this system. Proper login validation will be tested.



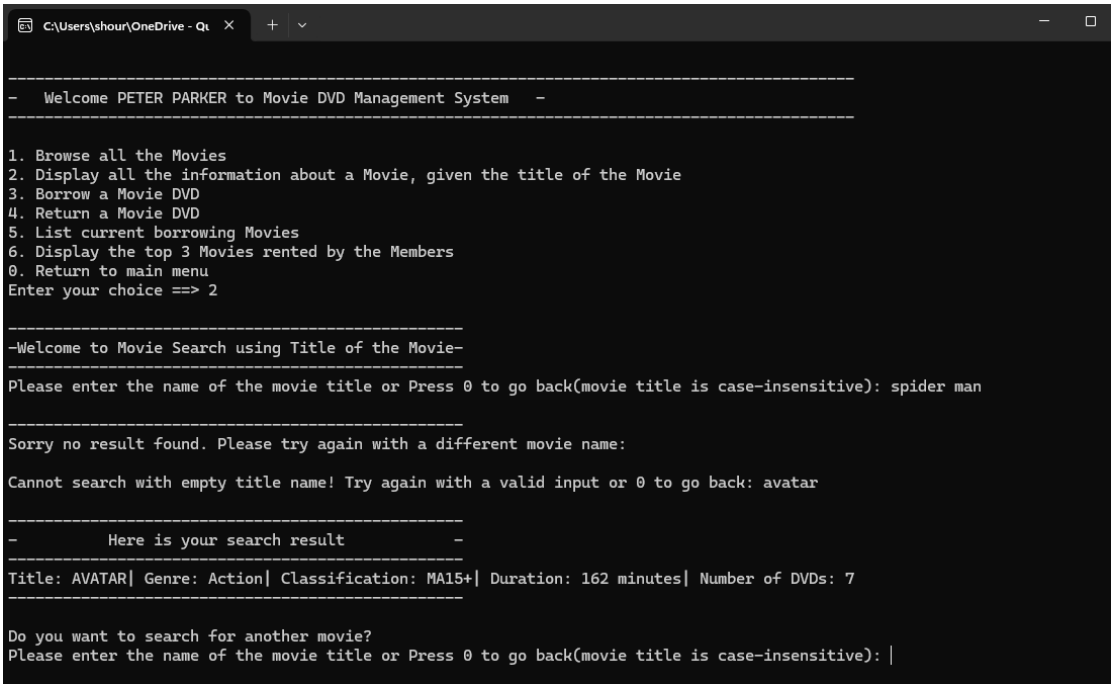
After inputting first name and last name the system will check if the name matches with any existing member names. Here member put “PETE” instead of “PETER” and “PARK” instead of “PARKER”, so the system says no member found.



Now the above screenshot shows, as the name matches with existing member name system asks for the password. But as member puts the password wrong it says password not matched. So, again it asks for correct password. After giving the correct password system shows the member menu along with the name of that member.

Searching Movie using Movie Title

When searching for a movie the title is used. If the movie is not found than it says no result found. Also, member cannot leave the field blank.

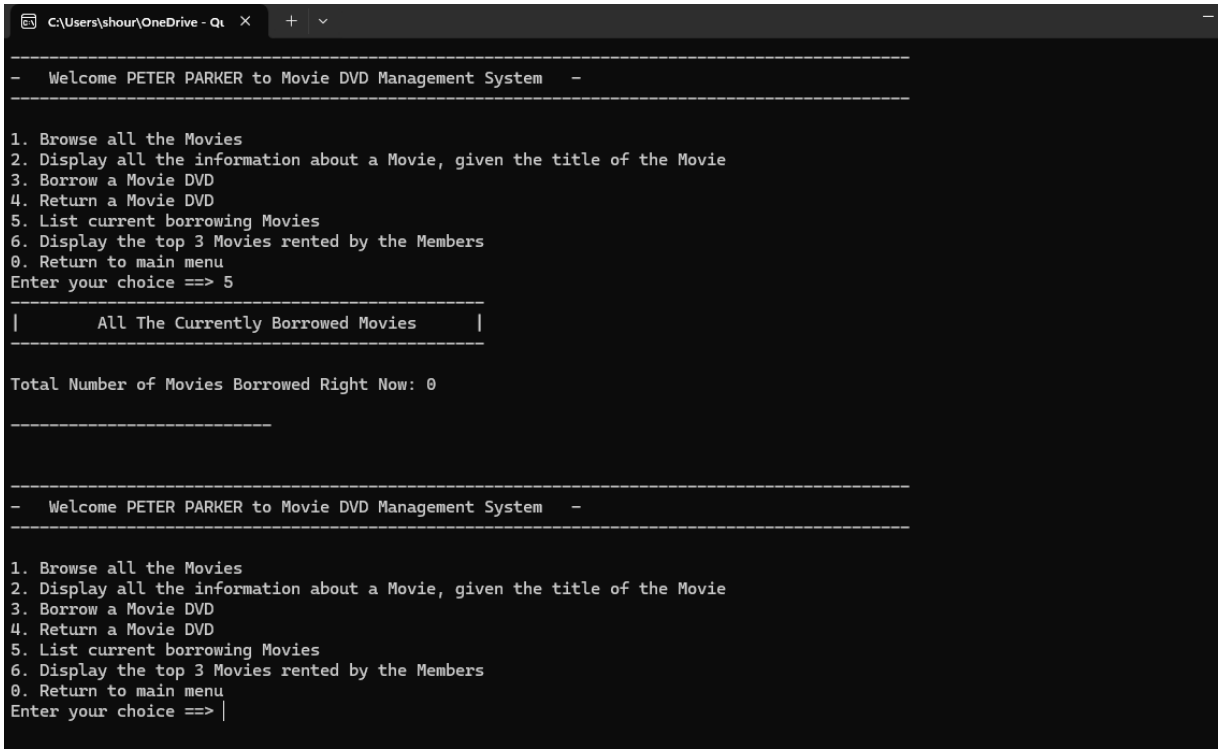


After finding the movie the system shows the details of the movie.

Borrowing/Returning Movie

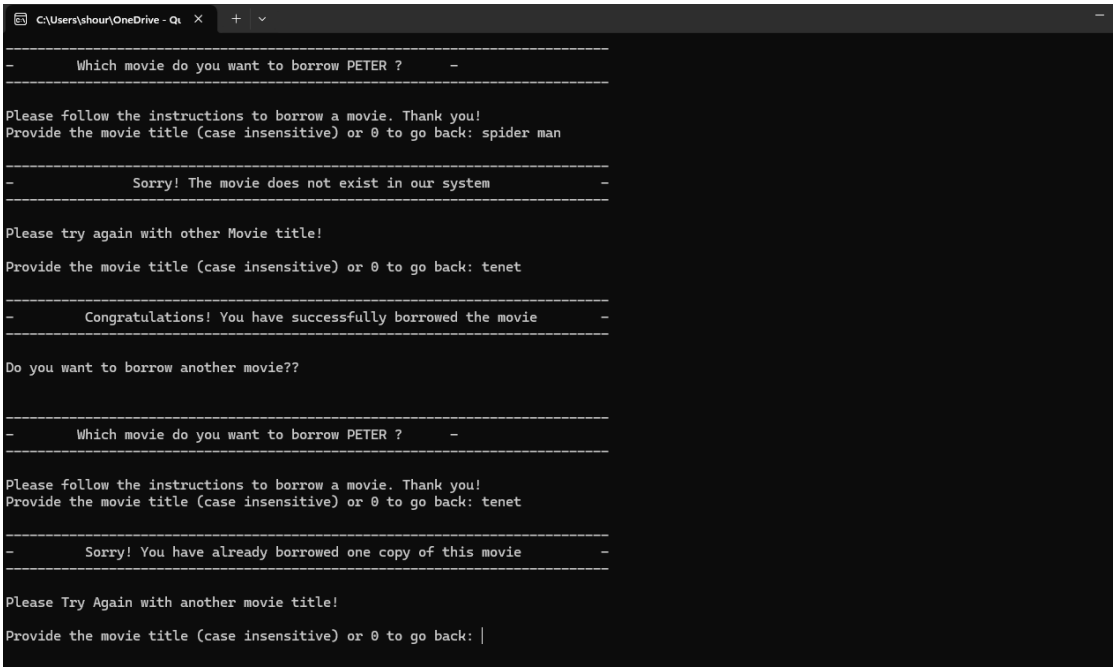
Borrowing Movie:

Now “PETER PARKER” is not borrowing any movie. So, system will show number of borrowing movies =0.



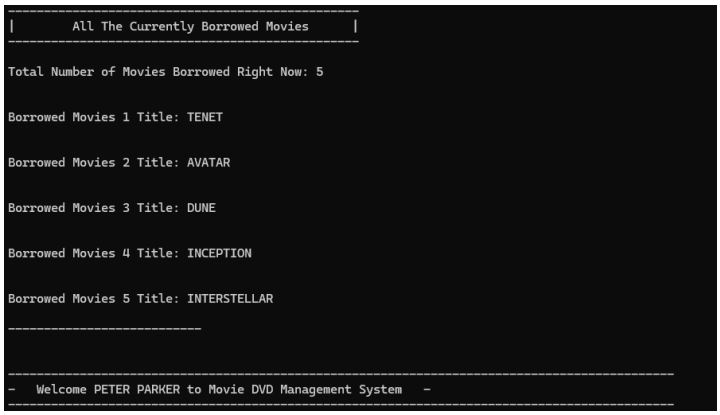
The above screenshot shows that the number of borrowed movies is zero.

Now the below screenshot shows different scenarios of borrowing movie in one screenshot.



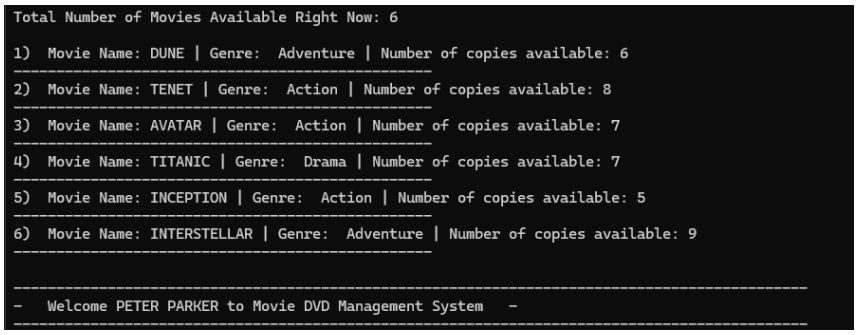
3 test cases have been shown on above screenshot. In the first search “SPIDER MAN” movie was searched but it does not exist in the system. So, system returned movie does not exist. In second search “TENET” was searched. And as “TENET” movie exists in the system member has successfully borrowed “TENET” movie. But in third test case “TENET” is already borrowed by that member so he cannot borrow any more copies of that movie.

In next screenshots “PETER PARKER” will borrow 4 more movies to show the test case that a member cannot borrow more than 5 movies. As “Member” borrows the movies the current available number of DVDs will also decrement by 1. To show that, a before and after borrowing screenshot will be added.

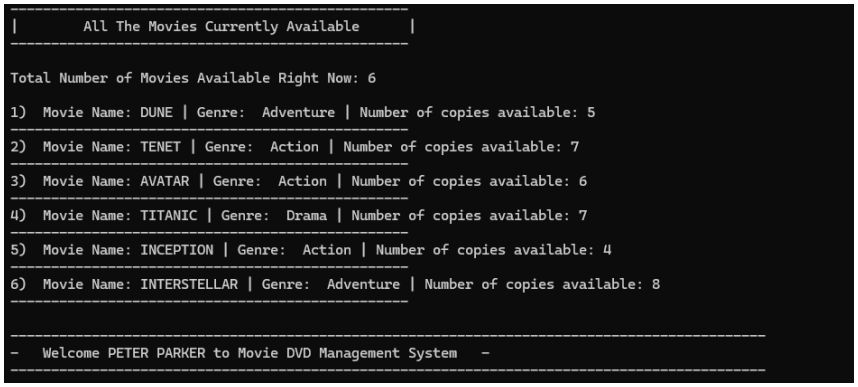


That is screenshot of five movies borrowed by PETER PARKER.

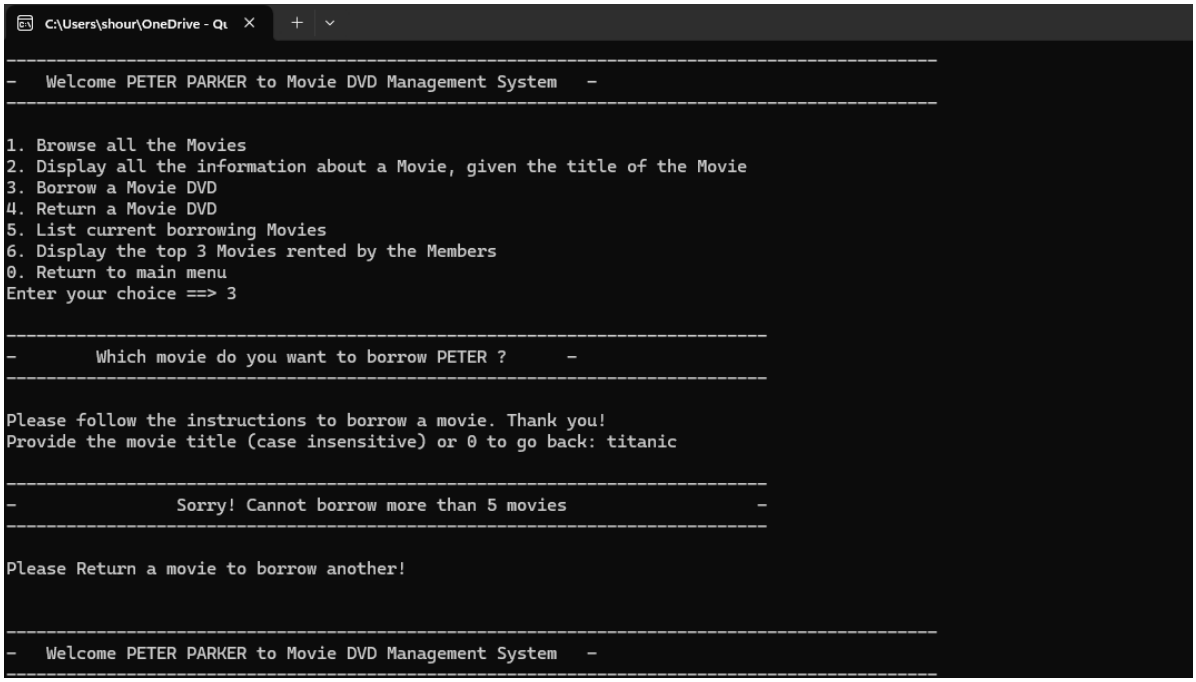
Before borrowing the movies, number of copies available,



After borrowing the movies, number of copies available,



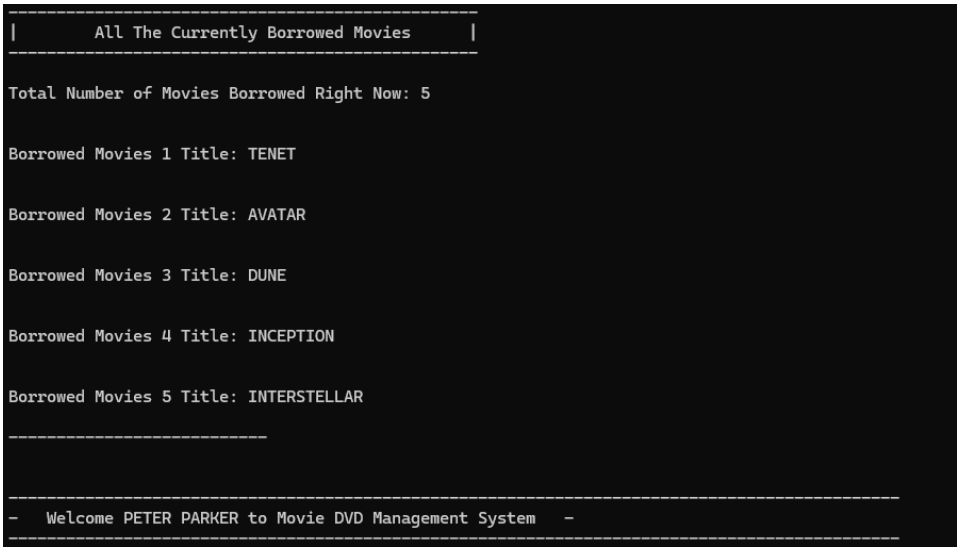
The number of DVDs/copies of borrowed movies are decremented by 1. Now “PETER PARKER” will try to borrow another movie. But the system will not allow to do it because he already reached maximum number of DVDs a member can borrow.



The system responded “PETER PARKER” needs to return a DVD of any movie that he is currently borrowing to borrow another one. To return a movie, member has to input the title. The title will be matched with already borrowed movies. If it matches the system will say successfully returned the movie. It is demonstrated in next screenshot.

Returning DVDs:

The borrowed movies of “PETER PARKER”



In next screenshot, “PETER PARKER” tries to return movie “TITANIC”, but he did not yet borrow that movie so the system will say the movie is not in the borrow list.

```
------------------------------------------------------------------------
-       Which movie do you want to return PETER ?      -
------------------------------------------------------------------------

Please follow the instructions to return a movie. Thank you!
Provide the movie title (case insensitive) or 0 to go back: titanic

------------------------------------------------------------------------
-          Sorry! The movie does not exist in your borrow list           -
------------------------------------------------------------------------

Please try again with other Movie title!

Provide the movie title (case insensitive) or 0 to go back: dune

------------------------------------------------------------------------
-         Congratulations! You have successfully returned the movie       -
------------------------------------------------------------------------

Do you want to return another movie??


------------------------------------------------------------------------
```

As “TITANIC” was not in the borrow list the system responded with a prompt that movie does not exists in PETER’s borrowed movies. But as PETER borrowed “DUNE” he can successfully return “DUNE”.

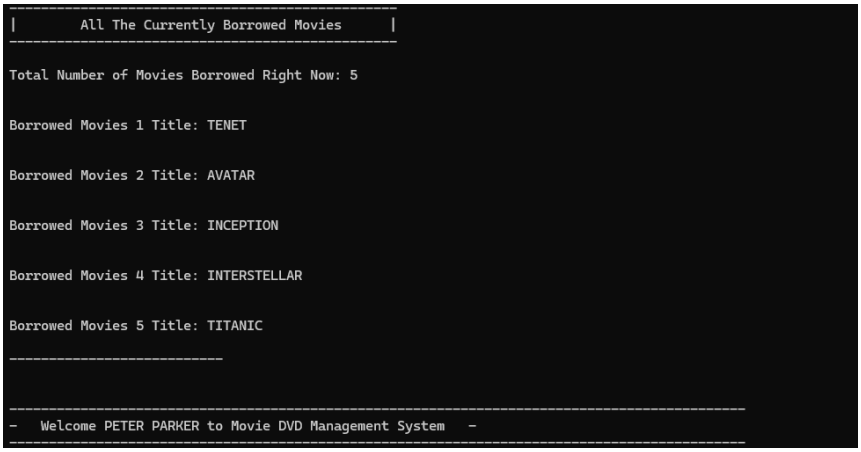
```
Total Number of Movies Borrowed Right Now: 4


Borrowed Movies 1 Title: TENET


Borrowed Movies 2 Title: AVATAR


Borrowed Movies 3 Title: INCEPTION


Borrowed Movies 4 Title: INTERSTELLAR

---------------------------


------------------------------------------------------------------------------------------------------
-   Welcome PETER PARKER to Movie DVD Management System    -
------------------------------------------------------------------------------------------------------
```

Now, PETER has 4 borrowed movies and can borrow another movie. The next two screenshots demonstrate that.

“PETER PARKER” trying to add “TITANIC” movie.

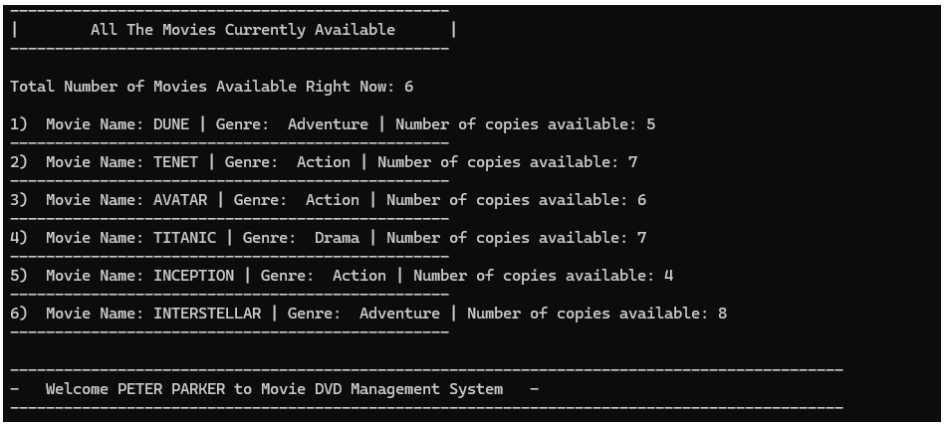
```
------------------------------------------------------------------------
-       Which movie do you want to borrow PETER ?      -
------------------------------------------------------------------------

Please follow the instructions to borrow a movie. Thank you!
Provide the movie title (case insensitive) or 0 to go back: titanic

------------------------------------------------------------------------
-         Congratulations! You have successfully borrowed the movie        -
------------------------------------------------------------------------

Do you want to borrow another movie??


------------------------------------------------------------------------
```

The updated borrowed movie list of “PETER PARKER”

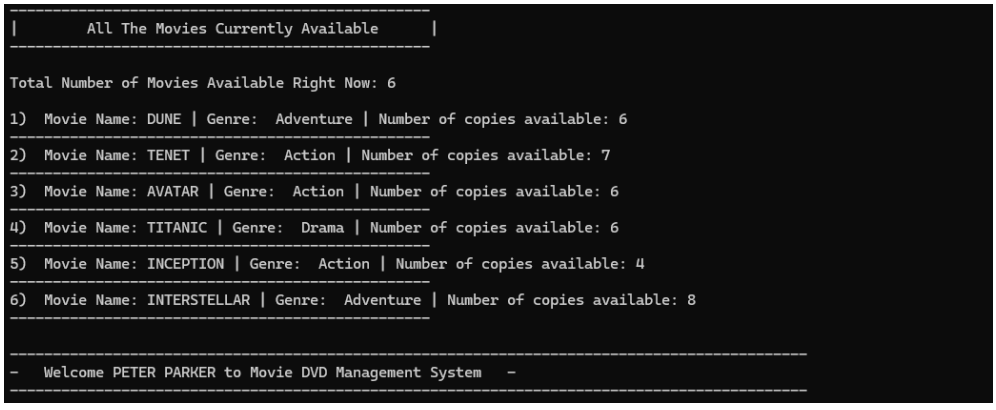


“TITANIC” is successfully borrowed. The next screenshot shows that number of copies is for “TITANIC” is decreased. And as, he returned “DUNE” movie, the number of copies for “DUNE” movie has increased.

Before borrowing and returning,



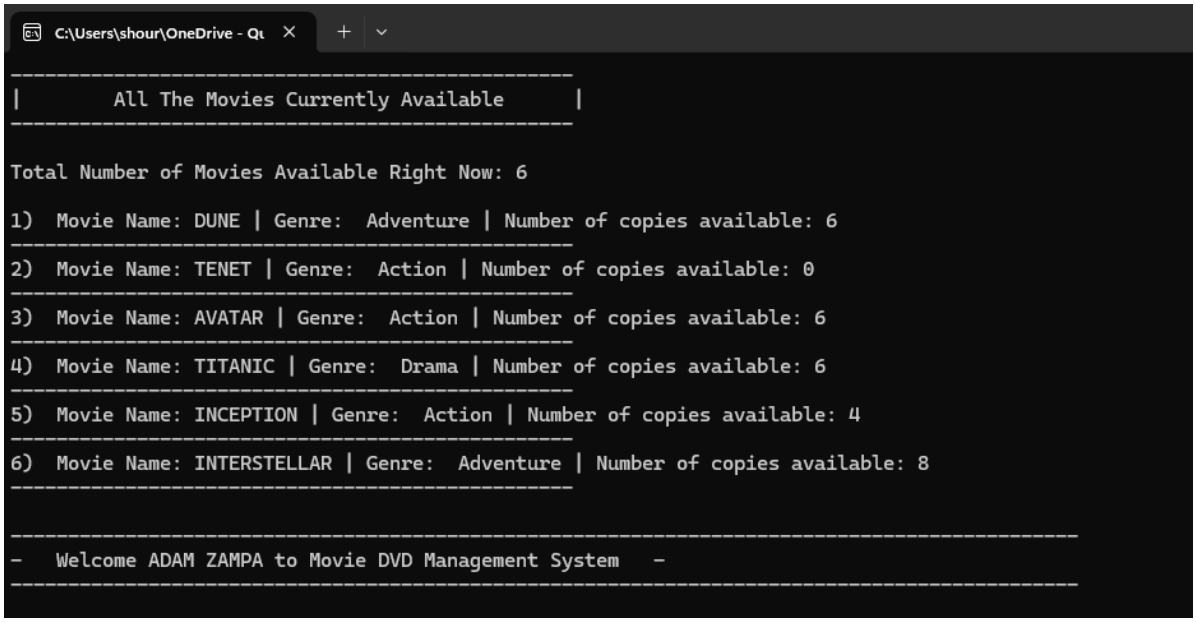
After borrowing and returning,



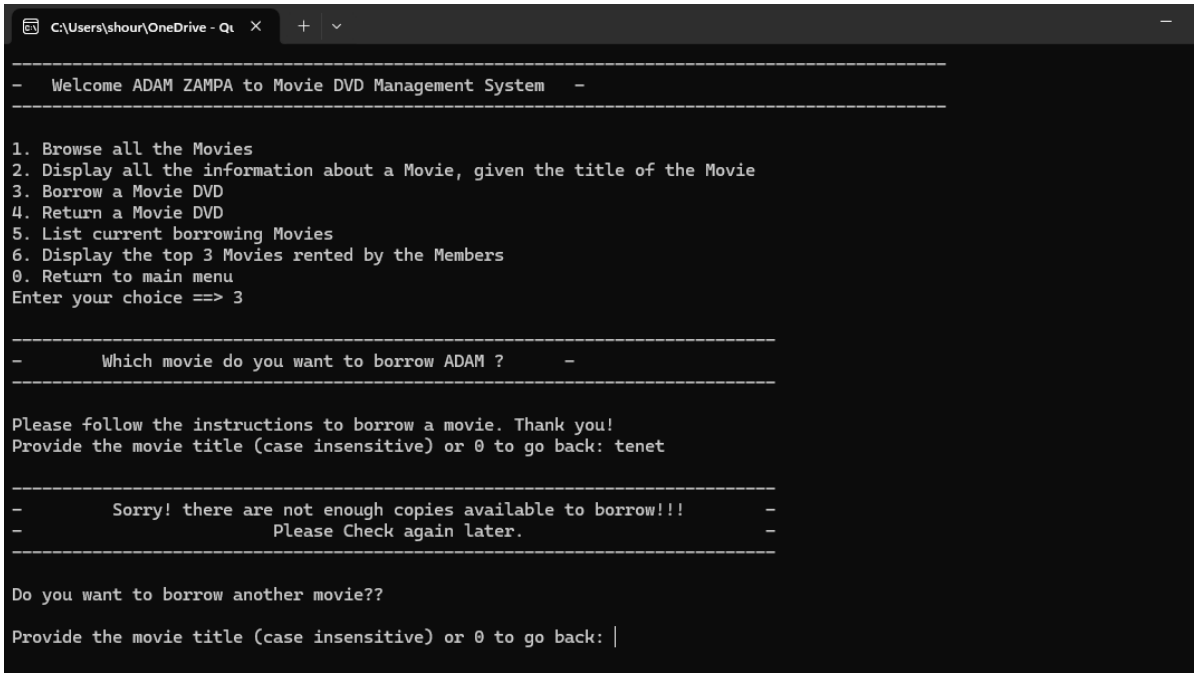
Number of “DUNE” changed from 5 to 6 and “TITANIC” from 7 to 6.

Borrowing movie with 0 number of copies:

Now “ADAM ZAMPA” wants to borrow a DVD of “TENET” movie. So, he checks currently available movies.



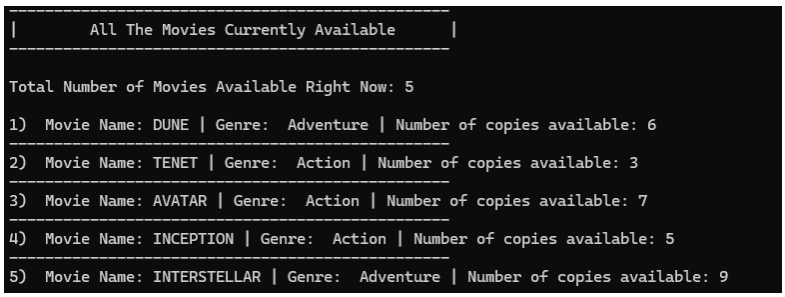
But he sees that there are no copies available. But instead, he tries to borrow a copy to check if the system lets him to do so.



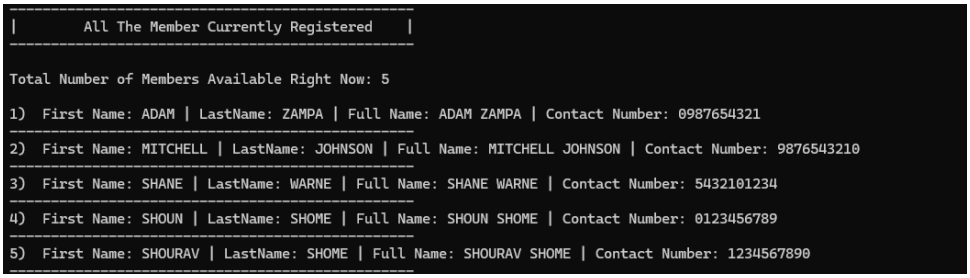
The system gives him a prompt saying that there are not enough copies available to borrow right now. So, he needs to check back later again.

Display Top 3 Movies Borrowed and Find Members Currently Borrowing a Particular Movie

To test this functionality, I have to borrow movies from different “Members” and keep checking how the values changes when movies are borrowed. Initially 5 movies and 5 members are hard coded in the system. And each member borrows a movie from these 5 movies. So, all the movies are borrowed once. Staff can check which movies are borrowed by which member. Screenshots are added below.

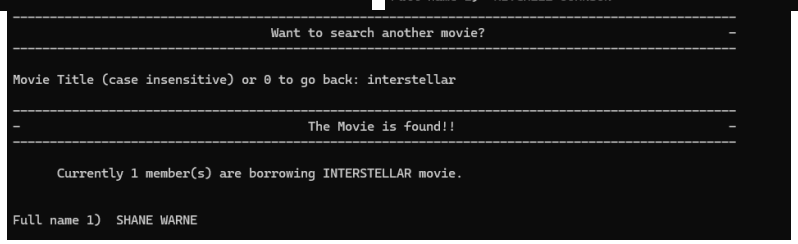


Above screenshot shows currently available movies.

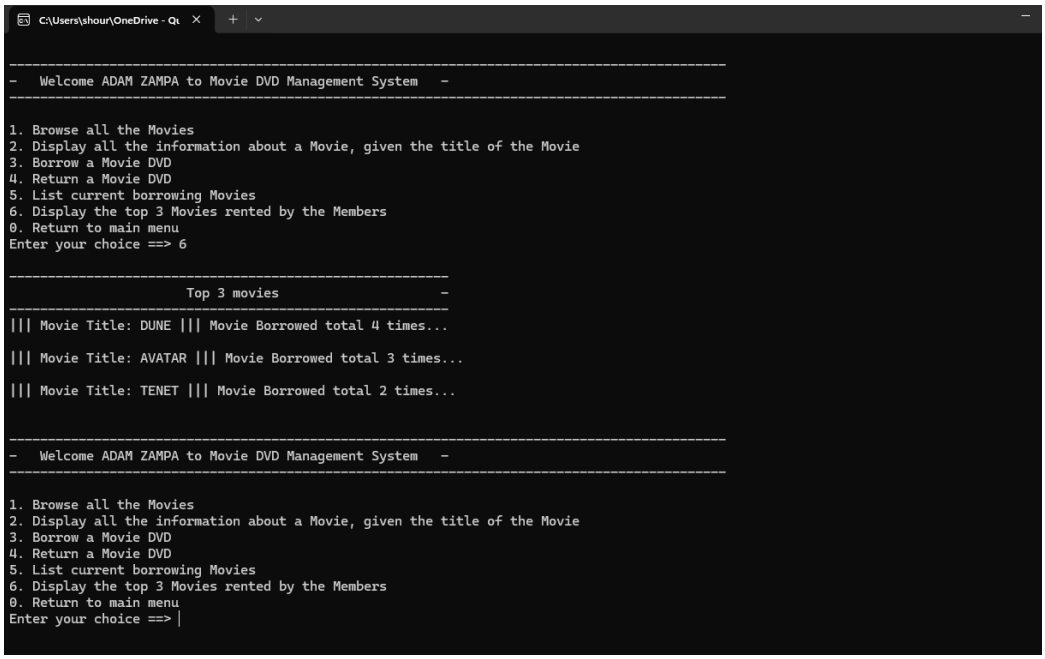


This above screenshot shows currently available members.

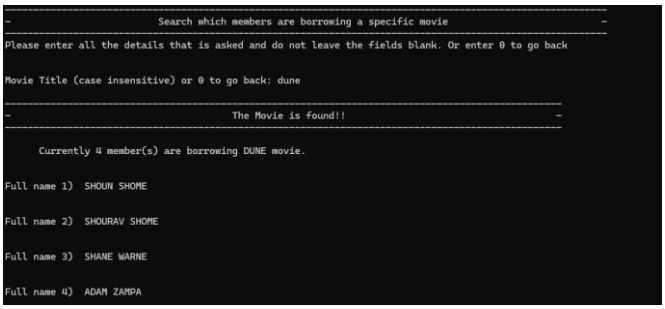
Now in below screenshots, each movie is borrowed by each member.

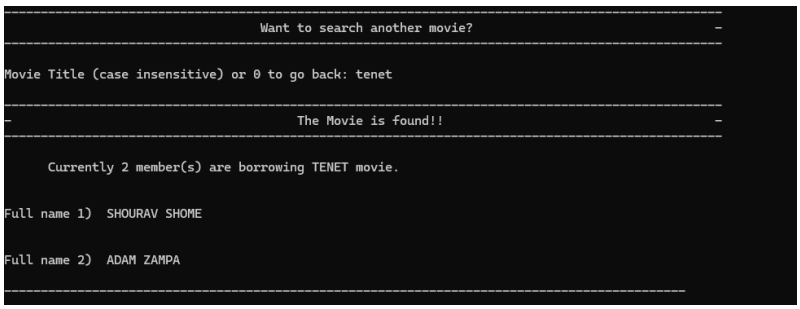


These above screenshots, show that all movies are borrowed by each member. Now, Member SHOURAV, SHANE and ADAM will borrow “DUNE” movie. So, DUNE will be borrowed 3 times and as it is already borrowed by SHOUN total borrowed frequency will be 4. Same way AVATAR will be borrowed by SHOURAV and MITCHELL. So, it will have a total borrowed frequency of 3. And TENET will be borrowed by ADAM so it will have total borrowed frequency of 2. So, when a member will look for top 3 movies DUNE, AVATAR and TENET will come up in sequence from highest to lowest. The screenshot attached below is taken after completion of the above borrowing scenarios.

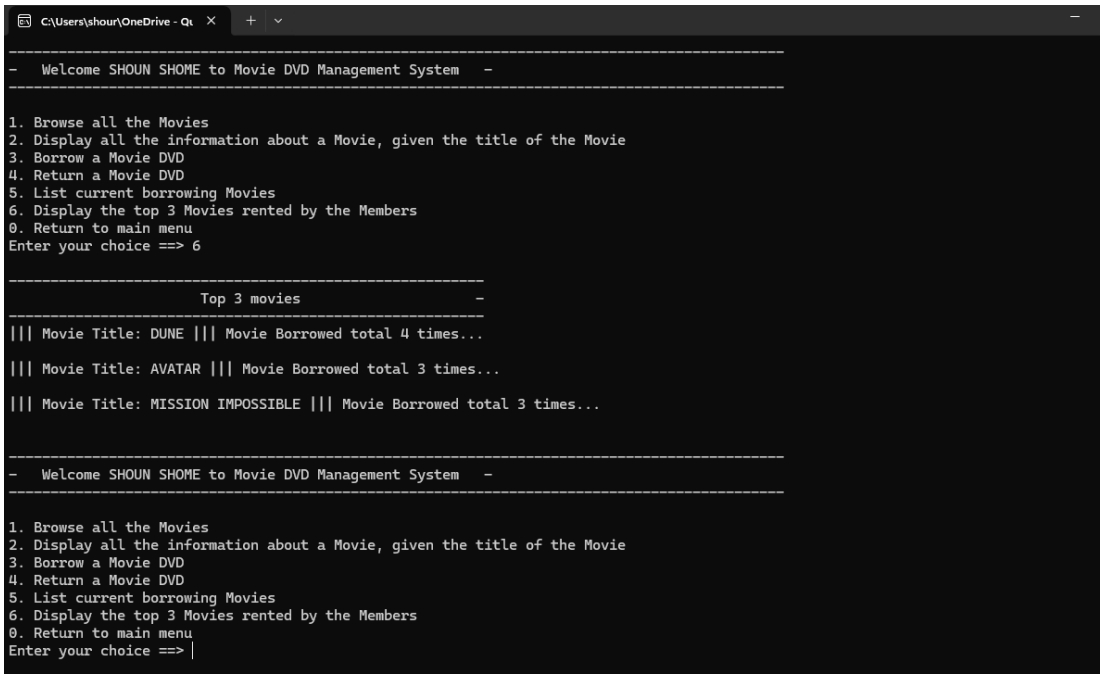


The above screenshot shows the scenario after the intended borrowing is completed. ADAM sees that DUNE is borrowed the maximum times and AVATAR and TENET is right after that.

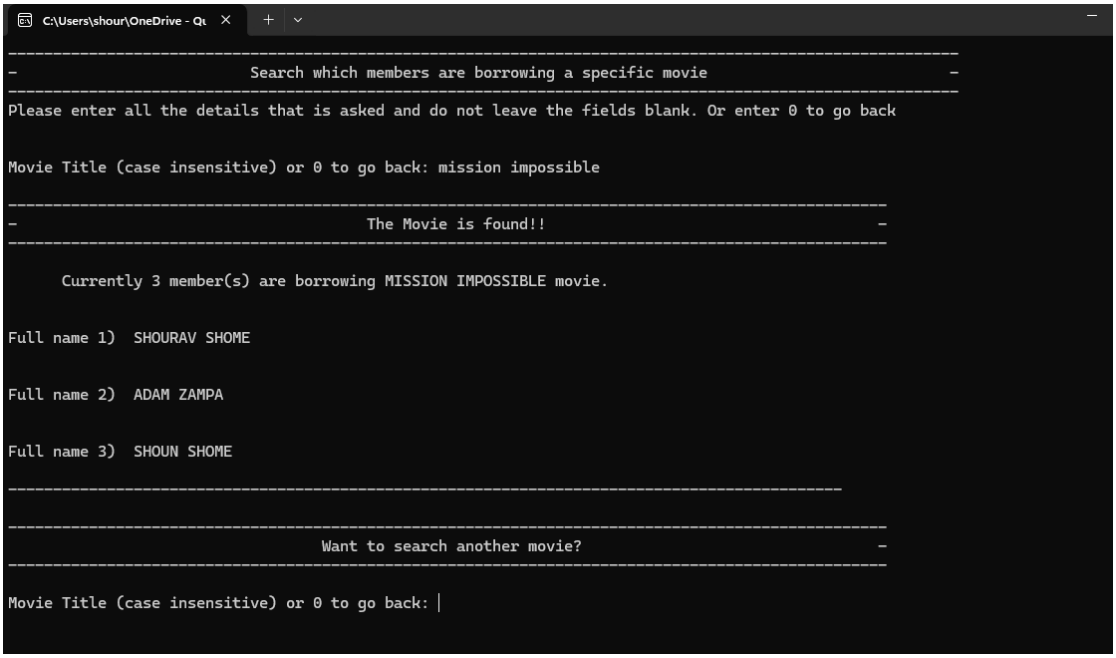




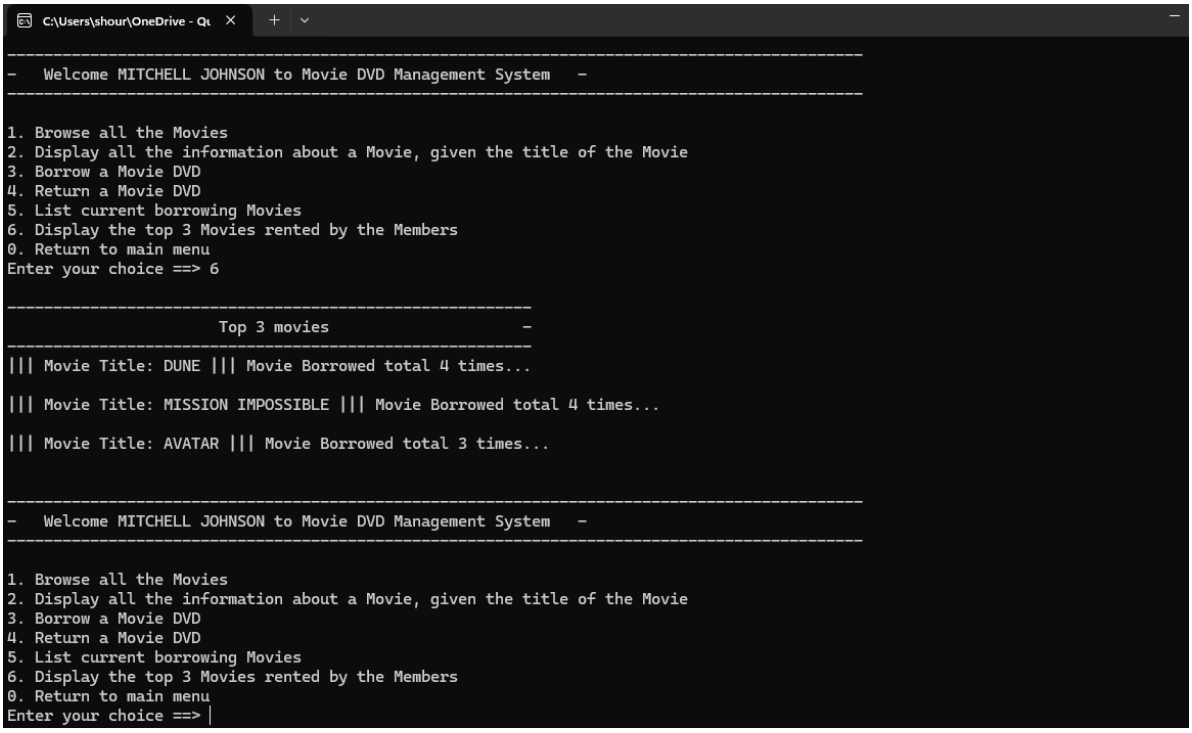
Above screenshots are the proof that these movies are currently borrowed by these members. The above screenshots show that after borrowing “Dune” for 4 times, “AVATAR” for 3 times and “TENET” for 2 times, the system is showing the movies that have highest borrowed frequency. To further test the algorithm another movie called “MISSION IMPOSSIBLE” will be added and borrowed by all the members. So, this movie will have highest borrowed frequency. Step by step screenshot will be added to show how this movie will get to the top from bottom.



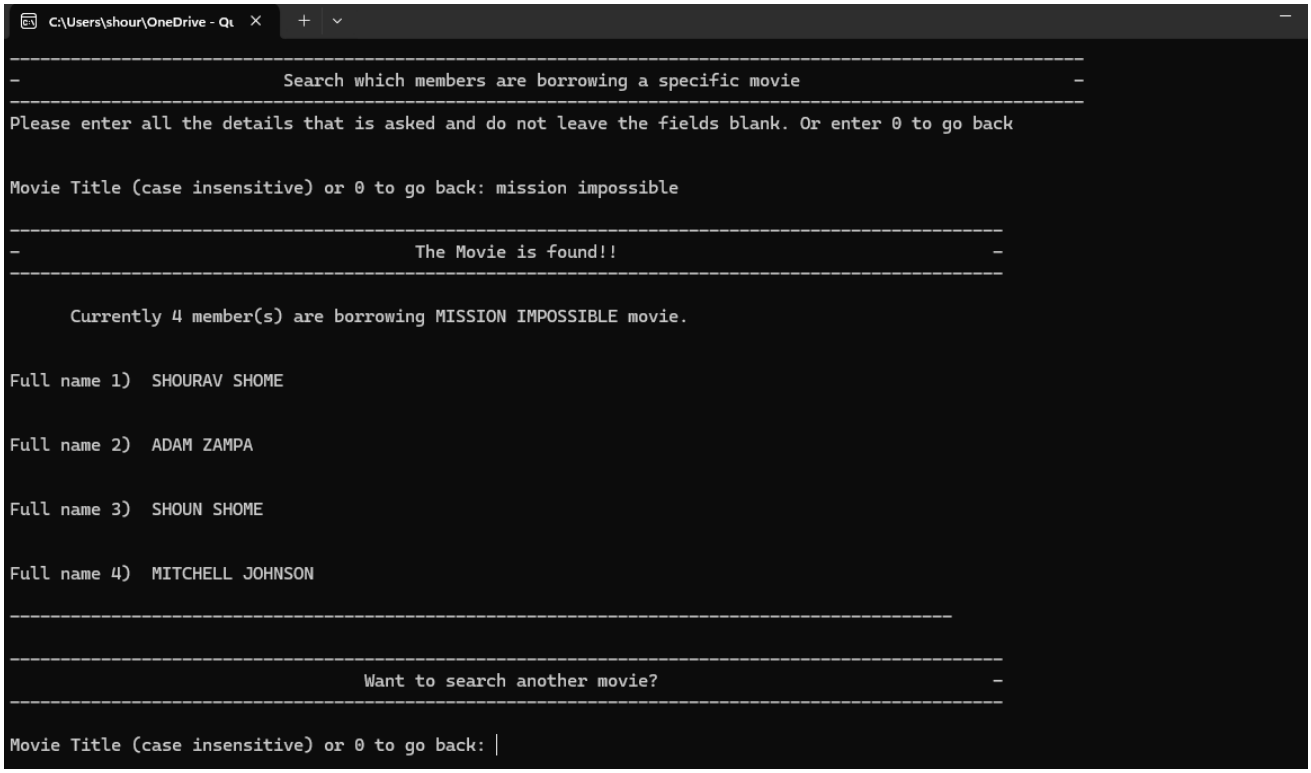
After borrowing the movie for the 3rd time, it claimed up and replaced “TENET” as that was borrowed 2 times.



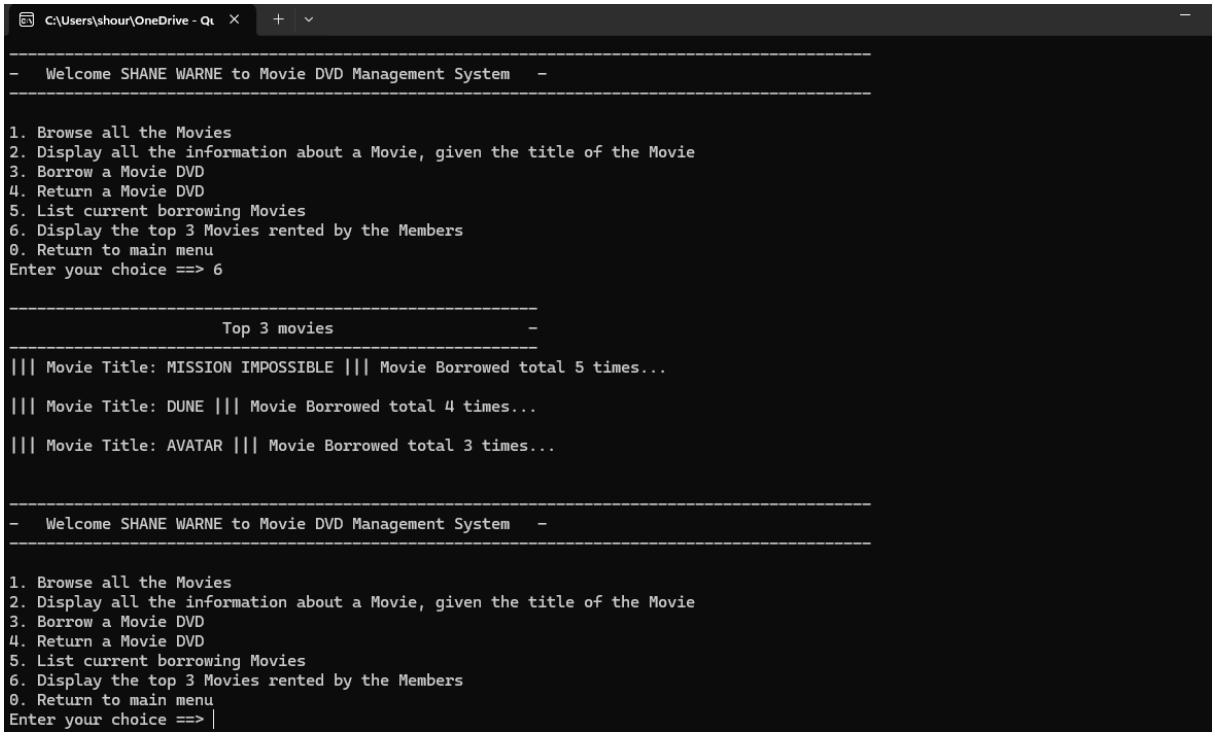
Above screenshot shows that “MISSION IMPOSSIBLE” is borrowed by three members.



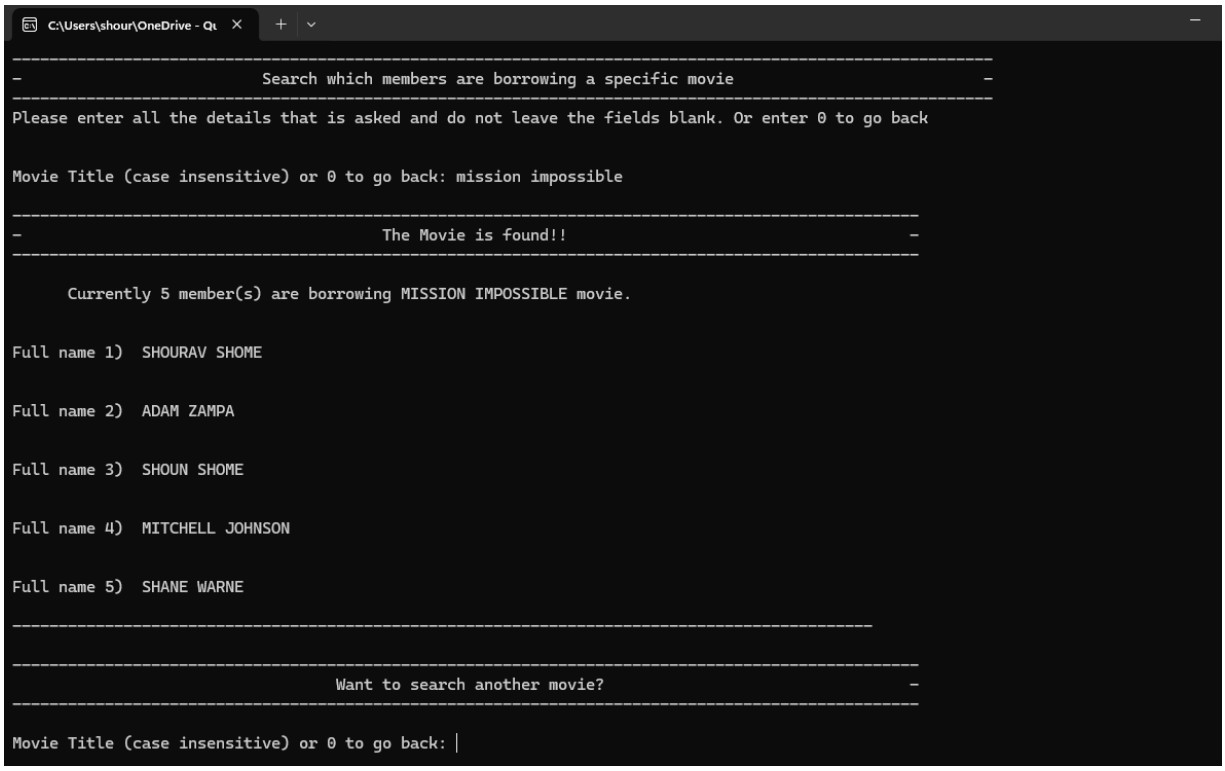
Above screenshot is taken After borrowing for 4th time. The movie came to 2nd position as it is borrowed more times than “AVATAR”.



Above screenshot shows that “MISSION IMPOSSIBLE” movie is borrowed for 4th time.



Above screenshot is taken after borrowing the movie for 5th time. As, this movie has the highest borrowing frequency among all the movies it claims the first position when a "Member" wants to look at top 6 borrowed movie from the system.



Above screenshot further clarifies that the movie is being borrowed by 5 members and claims the highest borrowing frequency.

References

GeeksforGeeks. (2024a, March 29). *Double hashing*. GeeksforGeeks. https://www.geeksforgeeks.org/double-hashing/

GeeksforGeeks. (2024b, May 7). *Insertion sort data structure and algorithm tutorials*. GeeksforGeeks. https://www.geeksforgeeks.org/insertion-sort/

GeeksforGeeks. (2024, May 8). *Hash table data structure*. GeeksforGeeks. https://www.geeksforgeeks.org/hash-table-data-structure/