CS 201, Spring 2024
Homework Assignment 1

Due: 23:59, March 17, 2024

In this homework, you will implement a DVD Store Management System which is a software application designed to efficiently manage the operations of a DVD store, providing staff with tools to handle tasks such as DVD management, customer management, DVD rental, and return, and retrieval of transaction history. It is designed to streamline store operations, improve accessibility to DVDs, and enhance the overall user experience for customers. In your implementation, you **MUST** use dynamically allocated arrays for storing the DVD and customer information.

The DVD Store Management System will have the following functionalities. The details of these functionalities are given below:

1. Add a new DVD to the inventory
2. Remove an existing DVD from the inventory
3. Add a new customer to the database
4. Remove an existing customer from the database
5. Allow a customer to rent a DVD from the store
6. Allow a customer to return a rented DVD to the store
7. Show the list of all DVDs in the inventory
8. Show the list of all customers registered with the store
9. Show detailed information about a specific DVD
10. Show detailed information about a specific customer
11. Show the transaction history of the store

Add a DVD: The system will allow a store staff to add a new DVD to the store's inventory. The DVD information includes its serial number, title, and director. The serial number serves as a unique identifier for each DVD, ensuring that no two DVDs have the same serial number. Thus, the system should check whether or not the specified DVD already exists, and if it exists, it should not allow the operation and display a warning message. Initially, a DVD is available for renting. You can assume that a DVD with the same serial number as a previously removed DVD will not be added to the system in the tests.

Remove a DVD: Store staff can remove DVDs from the store's inventory based on their serial numbers. If the DVD does not currently exist in the system, the system should not allow the operation and display a warning message. If the DVD is currently rented, the system should not allow the operation and display a warning message.

Add a customer: Store staff can add new customers to the system by assigning them a unique customer ID and recording their name. The customer ID serves as a unique identifier for each customer, enabling staff to track individual customer activity within the system. Thus, the system should check whether or not the specified customer already exists, and if they exist, it should not allow the operation and display a warning message. Initially, a customer has no DVD rented. You can assume that a customer with the same ID as a previously removed customer will not be added to the system in the tests.

Remove a customer: This feature enables store staff to remove customers from the system based on their customer IDs. If the customer does not currently exist in the system, the system should not allow the operation and display a warning message. If the customer has currently rented at least one DVD, the system should not allow the operation and display a warning message.

Rent a DVD: Allows customers to rent DVDs from the store by providing their customer ID and the serial number of the DVD they wish to rent. Once a DVD is rented, its availability status is updated as "Rented". If a DVD with the specified serial number does not exist, the system should not allow the operation and display a warning message. If a DVD with the specified serial number exists but is not available, the system should not allow the operation and display a warning message. If a customer with the specified ID does not exist, the system should not allow the operation and display a warning message. If neither the DVD nor the customer exists in the system, the warning message should explain this situation (see examples).

Return a DVD: Enables customers to return DVDs to the store by providing their customer ID and the serial number of the DVD they are returning. When a DVD is returned, its status becomes "Available". If a DVD with the specified serial number does not exist, the system should not allow the operation and display a warning message. If a DVD with the specified serial number exists but is not currently rented by this customer, the system should not allow the operation and display a warning message. If a customer with the specified ID does not exist, the system should not allow the operation and display a warning message. If neither the DVD nor the customer exists in the system, the warning message should explain this situation (see examples).

Show all DVDs: The system will allow to display a list of all DVDs that currently exist in the system. The list includes, for each DVD, its serial number, title, director, and status (Rented or Available). Note that the order of DVDs in the list IS important; you MUST list in the order they are added to the system.

Show all customers: The system will allow to display a list of all customers currently registered in the system. The list includes, for each customer, the customer ID, name, and the number of DVDs currently rented by this customer. Note that the order of customers in the list IS important; you MUST list in the order they are added to the system.

Show a specific DVD: The system will allow to display information about a specific DVD. This information includes its serial number, title, director, and status (Rented or Available). If the DVD does not currently exist in the system, the system should not allow the operation and display a warning message.

Show a specific customer: The system will allow to display information about a specific customer. This information includes the customer ID, name, and the number of DVDs currently rented by this customer. If the customer does not currently exist in the system, the system should not allow the operation and display a warning message.

Show transaction history: The system will allow to display detailed information about all past transactions including rentals and returns. The list includes, for each transaction, the serial number of the DVD, the customer ID, and the type of the transaction (Rental or Return). The list must include all past transactions, including operations corresponding to removed DVDs and removed customers. Note that the order of transactions in the list IS important; you MUST list in the order they are performed on the system.

Below is the required `public` part of the `DVDStoreManagementSystem` class that you must write in this assignment. The name of the class must be `DVDStoreManagementSystem`, and must include these public member functions. The interface for the class must be written in the file called `DVDStoreManagementSystem.h` and its implementation must be written in the file called `DVDStoreManagementSystem.cpp`. You can

define additional public and private member functions and data members in this class. You can also define additional classes in your solution and implement them in separate files.

```
class DVDStoreManagementSystem {

public:
    DVDStoreManagementSystem();
    ~DVDStoreManagementSystem();

    void addDVD( const string serialNo, const string title, const string director );
    void removeDVD( const string serialNo );
    void addCustomer( const int customerID, const string name );
    void removeCustomer( const int customerID );
    void rentDVD( const int customerID, const string serialNo );
    void returnDVD( const int customerID, const string serialNo );
    void showAllDVDs() const;
    void showAllCustomers() const;
    void showDVD( const string serialNo ) const;
    void showCustomer( const int customerID ) const;
    void showTransactionHistory() const;
};
```

Here is an example test program that uses this class and the corresponding output. We will use similar programs to test your solution so make sure that the name of the class is `DVDStoreManagementSystem`, its interface is in the file called `DVDStoreManagementSystem.h`, and the required functions are defined as shown above. Your implementation MUST use exactly the same format given in the example output to display the messages expected as the result of the defined functions.

Example test code:

```
#include <iostream>
using namespace std;

#include "DVDStoreManagementSystem.h"

int main() {

    DVDStoreManagementSystem dvdsms;

    cout << "Initial state:" << endl << endl;

    dvdsms.showAllDVDs();
    cout << endl;

    dvdsms.showAllCustomers();
    cout << endl;

    dvdsms.showTransactionHistory();
    cout << endl;

    // Add some DVDs
    dvdsms.addDVD("001", "Inception", "Christopher Nolan");
    dvdsms.addDVD("002", "The Shawshank Redemption", "Frank Darabont");
    dvdsms.addDVD("003", "Pulp Fiction", "Quentin Tarantino");
    dvdsms.addDVD("004", "The Godfather", "Francis Ford Coppola");
```

```
    dvdsms.addDVD("005", "The Dark Knight", "Christopher Nolan");

    cout << endl << "After adding DVDs:" << endl << endl;
    dvdsms.showAllDVDs();
    cout << endl;

    // Add an existing DVD
    dvdsms.addDVD("003", "Pulp Fiction", "Quentin Tarantino");
    cout << endl;

    // Remove a few DVDs
    dvdsms.removeDVD("004");
    dvdsms.removeDVD("005");

    cout << endl << "After removing DVDs:" << endl << endl;
    dvdsms.showAllDVDs();
    cout << endl;

    // Remove a non-existing DVD
    dvdsms.removeDVD("006");
    cout << endl;

    // Add some customers
    dvdsms.addCustomer(1001, "Alice");
    dvdsms.addCustomer(1002, "Bob");
    dvdsms.addCustomer(1003, "Charlie");
    dvdsms.addCustomer(1004, "Diana");
    dvdsms.addCustomer(1005, "Edward");

    cout << endl << "After adding customers:" << endl << endl;
    dvdsms.showAllCustomers();
    cout << endl;

    // Add an existing customer
    dvdsms.addCustomer(1001, "Alice");
    cout << endl;

    // Remove a few customers
    dvdsms.removeCustomer(1004);
    dvdsms.removeCustomer(1005);

    cout << endl << "After removing customers:" << endl << endl;
    dvdsms.showAllCustomers();
    cout << endl;

    // Remove a non-existing customer
    dvdsms.removeCustomer(1006);
    cout << endl;

    // Rent some DVDs (successful)
    dvdsms.rentDVD(1001, "001");
    dvdsms.rentDVD(1002, "002");
    dvdsms.rentDVD(1003, "003");

    cout << endl << "After renting DVDs:" << endl << endl;
    dvdsms.showTransactionHistory();
```

```
    cout << endl;

    // Rent a non-existing DVD with an existing customer
    dvdsms.rentDVD(1001, "006");

    // Rent an existing DVD with a non-existing customer
    dvdsms.rentDVD(1006, "001");

    // Rent a non-existing DVD with a non-existing customer
    dvdsms.rentDVD(1006, "006");

    // Rent a non-available DVD with an existing customer
    dvdsms.rentDVD(1002, "001");
    cout << endl;

    // Return a few DVDs (successful)
    dvdsms.returnDVD(1001, "001");
    dvdsms.returnDVD(1002, "002");

    cout << endl << "After returning DVDs:" << endl << endl;
    dvdsms.showTransactionHistory();
    cout << endl;

    // Return a non-existing DVD with an existing customer
    dvdsms.returnDVD(1001, "999");

    // Return an existing DVD with a non-existing customer
    dvdsms.returnDVD(999, "001");

    // Return a non-existing DVD with a non-existing customer
    dvdsms.returnDVD(999, "999");

    // Return a rented DVD with an existing customer but not rented by this customer
    dvdsms.returnDVD(1001, "003");

    // Remove a rented DVD
    dvdsms.removeDVD("003");

    // Remove a customer who rented a DVD
    dvdsms.removeCustomer(1003);
    cout << endl;

    // Show an existing DVD
    dvdsms.showDVD("002");

    // Show a non-existing DVD
    dvdsms.showDVD("999");

    // Show an existing customer
    dvdsms.showCustomer(1003);

    // Show a non-existing customer
    dvdsms.showCustomer(999);

    cout << endl << "Final state:" << endl << endl;
```

```
    dvdsms.showAllDVDs();
    cout << endl;

    dvdsms.showAllCustomers();
    cout << endl;

    dvdsms.showTransactionHistory();

    return 0;
}
```

Output of the example test code:

```
Initial state:

DVDs in the system:
None

Customers in the system:
None

Transactions in the system:
None

DVD with serial number 001 successfully added.
DVD with serial number 002 successfully added.
DVD with serial number 003 successfully added.
DVD with serial number 004 successfully added.
DVD with serial number 005 successfully added.

After adding DVDs:

DVDs in the system:
DVD: 001, Title: Inception, Director: Christopher Nolan, Available
DVD: 002, Title: The Shawshank Redemption, Director: Frank Darabont, Available
DVD: 003, Title: Pulp Fiction, Director: Quentin Tarantino, Available
DVD: 004, Title: The Godfather, Director: Francis Ford Coppola, Available
DVD: 005, Title: The Dark Knight, Director: Christopher Nolan, Available

Cannot add DVD. DVD with serial number 003 already exists.

DVD with serial number 004 successfully removed.
DVD with serial number 005 successfully removed.

After removing DVDs:

DVDs in the system:
DVD: 001, Title: Inception, Director: Christopher Nolan, Available
DVD: 002, Title: The Shawshank Redemption, Director: Frank Darabont, Available
DVD: 003, Title: Pulp Fiction, Director: Quentin Tarantino, Available

Cannot remove DVD. DVD with serial number 006 not found.

Customer with ID 1001 successfully added.
Customer with ID 1002 successfully added.
Customer with ID 1003 successfully added.
```

```
Customer with ID 1004 successfully added.
Customer with ID 1005 successfully added.

After adding customers:

Customers in the system:
Customer: 1001, Name: Alice, DVDs Rented: 0
Customer: 1002, Name: Bob, DVDs Rented: 0
Customer: 1003, Name: Charlie, DVDs Rented: 0
Customer: 1004, Name: Diana, DVDs Rented: 0
Customer: 1005, Name: Edward, DVDs Rented: 0

Cannot add customer. Customer with ID 1001 already exists.

Customer with ID 1004 successfully removed.
Customer with ID 1005 successfully removed.

After removing customers:

Customers in the system:
Customer: 1001, Name: Alice, DVDs Rented: 0
Customer: 1002, Name: Bob, DVDs Rented: 0
Customer: 1003, Name: Charlie, DVDs Rented: 0

Cannot remove customer. Customer with ID 1006 not found.

DVD with serial number 001 successfully rented by customer with ID 1001.
DVD with serial number 002 successfully rented by customer with ID 1002.
DVD with serial number 003 successfully rented by customer with ID 1003.

After renting DVDs:

Transactions in the system:
Transaction: Rental, Customer: 1001, DVD: 001
Transaction: Rental, Customer: 1002, DVD: 002
Transaction: Rental, Customer: 1003, DVD: 003

Cannot rent DVD. DVD with serial number 006 not found.
Cannot rent DVD. Customer with ID 1006 not found.
Cannot rent DVD. Customer with ID 1006 and DVD with serial number 006 not found.
Cannot rent DVD. DVD with serial number 001 is not available.

DVD with serial number 001 successfully returned by customer with ID 1001.
DVD with serial number 002 successfully returned by customer with ID 1002.

After returning DVDs:

Transactions in the system:
Transaction: Rental, Customer: 1001, DVD: 001
Transaction: Rental, Customer: 1002, DVD: 002
Transaction: Rental, Customer: 1003, DVD: 003
Transaction: Return, Customer: 1001, DVD: 001
Transaction: Return, Customer: 1002, DVD: 002

Cannot return DVD. DVD with serial number 999 not found.
Cannot return DVD. Customer with ID 999 not found.
```

```
Cannot return DVD. Customer with ID 999 and DVD with serial number 999 not found.
Cannot return DVD. DVD with serial number 003 not rented by customer with ID 1001.
Cannot remove DVD. DVD with serial number 003 is currently rented by a customer.
Cannot remove customer. Customer with ID 1003 has rented DVDs.

DVD: 002, Title: The Shawshank Redemption, Director: Frank Darabont, Available
DVD with serial number 999 not found.
Customer: 1003, Name: Charlie, DVDs Rented: 1
Customer with ID 999 not found.

Final state:

DVDs in the system:
DVD: 001, Title: Inception, Director: Christopher Nolan, Available
DVD: 002, Title: The Shawshank Redemption, Director: Frank Darabont, Available
DVD: 003, Title: Pulp Fiction, Director: Quentin Tarantino, Rented

Customers in the system:
Customer: 1001, Name: Alice, DVDs Rented: 0
Customer: 1002, Name: Bob, DVDs Rented: 0
Customer: 1003, Name: Charlie, DVDs Rented: 1

Transactions in the system:
Transaction: Rental, Customer: 1001, DVD: 001
Transaction: Rental, Customer: 1002, DVD: 002
Transaction: Rental, Customer: 1003, DVD: 003
Transaction: Return, Customer: 1001, DVD: 001
Transaction: Return, Customer: 1002, DVD: 002
```

IMPORTANT NOTES:
Do not start working on your homework before reading these notes!!!

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. **You MUST use dynamically allocated arrays with only the necessary amount of memory in your implementation.** That is, if there are 10 DVDs in the system, it should use memory only for these 10 DVDs. In other words, you cannot initially allocate a large array for DVDs and expect it to get filled later. The same argument applies to space used to store customers and transactions. **You will get no points if you use fixed-sized arrays, linked lists or any other data structures such as vectors/arrays/etc. from the standard library.** However, if necessary, you may define additional data members and member functions.

2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.

3. Output message for each operation MUST match the format shown in the output of the example code. Your output will be compared verbatim with the expected output during evaluation.

4. Your code MUST NOT have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at `http://valgrind.org`.

NOTES ABOUT SUBMISSION:

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. Your class name MUST BE `DVDStoreManagementSystem` and your file names MUST BE `DVDStoreManagementSystem.h` and `DVDStoreManagementSystem.cpp`. Note that you may write additional class(es) in your solution.

2. The code (`main` function) given above is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function called `main`.

3. You should put all of your `.h` and `.cpp` files into a folder and zip the folder (in this zip file, there should not be any file containing a `main` function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded.

4. Make sure that each file that you submit (each and every file in the archive) contains your name, section, and student number at the top as comments.

5. You are free to write your programs in any environment (you may use Linux, Windows, MacOS, etc.). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. Your code will be tested by using an automated test suite that includes multiple test cases where each case corresponds to a specific number of points in the overall grade. We will provide you with example test cases by email. Thus, we strongly recommend you to make sure that your program successfully compiles and correctly works on dijkstra.ug.bcc.bilkent.edu.tr before submitting your assignment. If your current code does not fully compile on dijkstra before submission, you can try to comment out the faulty parts so that the remaining code can be compiled and tested during evaluation.

6. This assignment is due by 23:59 on Sunday, March 17, 2024. You should upload your work to Moodle before the deadline. No hardcopy submission is needed. Late submissions will not be accepted (if you can upload to Moodle, then you are fine). There will be no extension to this deadline.

7. We use an automated tool as well as manual inspection to check your submissions against plagiarism. Please see the course home page for further discussion of academic integrity and the honor code for programming courses in our department.

8. For questions regarding academic integrity and use of external tools, please refer to the Honor Code for Introductory Programming Courses (CS 101/102/201/202) at `https://docs.google.com/document/d/1v_3ltpV_lClLsROXrMbojyuv4KrFQAm1uoz3SdC-7es/edit?usp=sharing`.

9. **This homework will be graded by your TA Aqsa Shabbir (aqsa.shabbir at bilkent.edu.tr). Thus, you may ask your homework related questions directly to her. There will also be a forum on Moodle for questions.**