UNIVERSITY *of* WASHINGTON | BOTHELL

Automatic Inventory Tracking System

CSS 343: Project 4

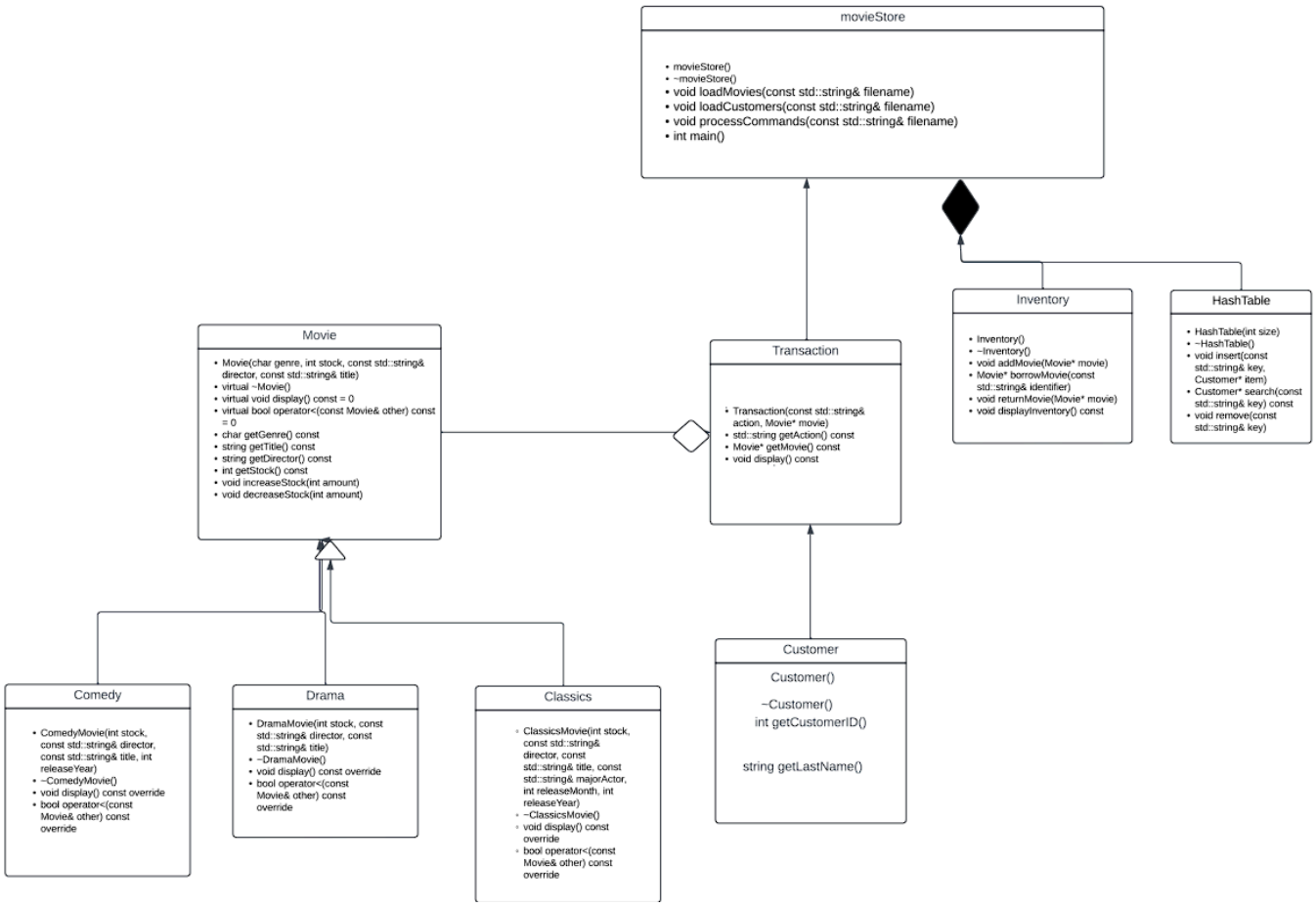
Designer and programmer: Forrest Zhang

Design Overview:

This program is designed for a movie rental store to manage inventory and track customer transactions. In the main function of this system, a MovieStore object is used to load and process data from three input files: movies, customers, and commands. The system is organized into four main categories of classes. The first category consists of the movie classes, where three specific types of movies—Comedy, Drama, and Classics—inherit from a base Movie class. The second category is the store management, encapsulated in the MovieStore class, which manages both the inventory of movies and the list of customers. The third category includes the Inventory and HashTable classes. The Inventory class manages the collection of movies, while the HashTable class efficiently stores and retrieves customer data using a hash table structure. The final category involves the Transaction class, which records each borrow and return action, linking movies and customers to track rental activities. The entire system is coordinated by the MovieStore class, which integrates all these components to handle the store's operations.

The main function serves as the entry point of the Movie Store System, where it initializes the MovieStore object and calls its run() method to load movie and customer data, and process commands such as borrowing and returning movies. The MovieStore class acts as the central controller of the system, managing the movie inventory through an Inventory object and customer data through a HashTable. It provides methods to load data from files and handle transactions, ensuring that the store's operations run smoothly by coordinating all interactions between movies and customers.

UML Diagram:



Class Design Description:

Movie

Auto_Tracking_Inventory_System (Global Scope)

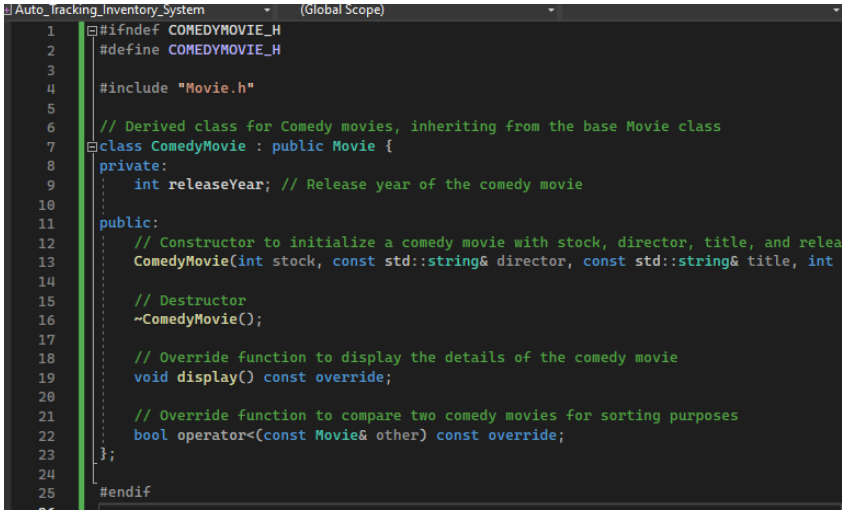
```h
#ifndef MOVIE_H
#define MOVIE_H

#include <string>

// Abstract base class for all types of movies
class Movie {
protected:
    char genre;           // Genre of the movie ('F' for Comedy, 'D' for Drama, 'C' fo
    int stock;            // Number of copies available in the inventory
    std::string director; // Director of the movie
    std::string title;    // Title of the movie

public:
    // Constructor to initialize common movie attributes
    Movie(char genre, int stock, const std::string& director, const std::string& title

    // Virtual destructor to ensure proper cleanup of derived classes
    virtual ~Movie();

    // Pure virtual function to display movie details, to be implemented by derived cl
    virtual void display() const = 0;

    // Pure virtual function to compare movies for sorting purposes
    virtual bool operator<(const Movie& other) const = 0;

    // Getter for the genre of the movie
    char getGenre() const;

    // Getter for the title of the movie
    std::string getTitle() const;

    // Getter for the director of the movie
    std::string getDirector() const;

    // Getter for the number of copies available
    int getStock() const;

    // Function to increase the stock of the movie
    void increaseStock(int amount);

    // Function to decrease the stock of the movie
    void decreaseStock(int amount);
};

#endif

```

Classics

Auto_Tracking_Inventory_System (Global Scope)

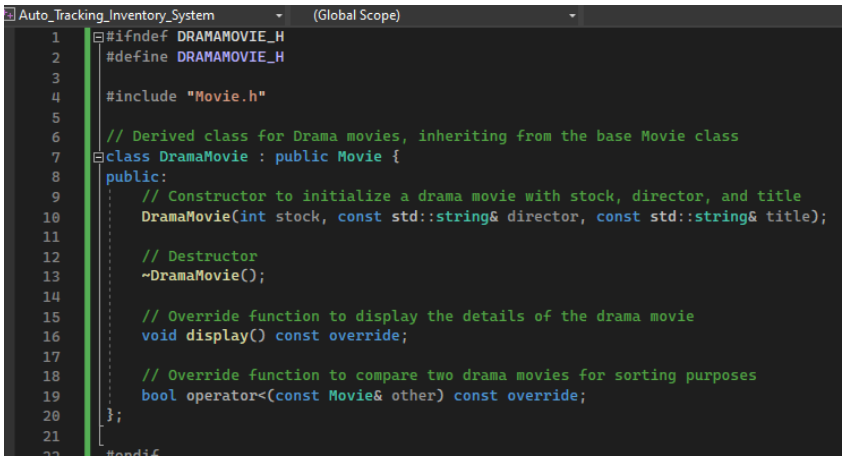
```h
#ifndef CLASSICSMOVIE_H
#define CLASSICSMOVIE_H

#include "Movie.h"
#include <string>

// Derived class for Classics movies, inheriting from the base Movie class
class ClassicsMovie : public Movie {
private:
    std::string majorActor; // Major actor in the classics movie
    int releaseMonth;       // Release month of the classics movie
    int releaseYear;        // Release year of the classics movie

public:
    // Constructor to initialize a classics movie with stock, director, title, major a
    ClassicsMovie(int stock, const std::string& director, const std::string& title, co

    // Destructor
    ~ClassicsMovie();

    // Override function to display the details of the classics movie
    void display() const override;

    // Override function to compare two classics movies for sorting purposes
    bool operator<(const Movie& other) const override;
};

#endif
```

Comedy




```
#ifndef COMEDYMOVIE_H
#define COMEDYMOVIE_H

#include "Movie.h"

// Derived class for Comedy movies, inheriting from the base Movie class
class ComedyMovie : public Movie {
private:
    int releaseYear; // Release year of the comedy movie

public:
    // Constructor to initialize a comedy movie with stock, director, title, and relea
    ComedyMovie(int stock, const std::string& director, const std::string& title, int

    // Destructor
    ~ComedyMovie();

    // Override function to display the details of the comedy movie
    void display() const override;

    // Override function to compare two comedy movies for sorting purposes
    bool operator<(const Movie& other) const override;
};

#endif
```

Drama




```
#ifndef DRAMAMOVIE_H
#define DRAMAMOVIE_H

#include "Movie.h"

// Derived class for Drama movies, inheriting from the base Movie class
class DramaMovie : public Movie {
public:
    // Constructor to initialize a drama movie with stock, director, and title
    DramaMovie(int stock, const std::string& director, const std::string& title);

    // Destructor
    ~DramaMovie();

    // Override function to display the details of the drama movie
    void display() const override;

    // Override function to compare two drama movies for sorting purposes
    bool operator<(const Movie& other) const override;
};

#endif
```

Customer

Auto_Tracking_Inventory_System (Global Scope)

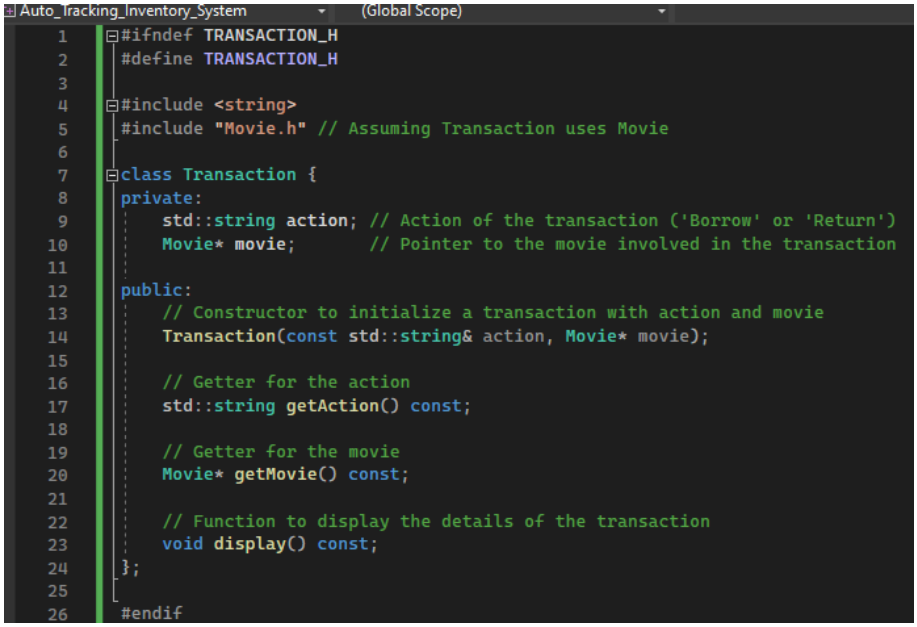
```
#ifndef CUSTOMER_H
#define CUSTOMER_H

#include <vector>
#include <string>
#include "Transaction.h"

// Class to represent a customer in the movie rental store
class Customer {
private:
    int customerID;            // Unique 4-digit customer ID
    std::string firstName;     // First name of the customer
    std::string lastName;      // Last name of the customer
    std::vector<Transaction*> transactionHistory; // List of transactions associated w

public:
    // Constructor to initialize a customer with an ID, first name, and last name
    Customer(int customerID, const std::string& firstName, const std::string& lastName

    // Destructor to clean up transaction history
    ~Customer();

    // Function to add a transaction to the customer's history
    void addTransaction(Transaction* transaction);

    // Function to display the transaction history of the customer
    void displayHistory() const;

    // Getter for the customer's ID
    int getCustomerID() const;

    // Getter for the transaction history
    const std::vector<Transaction*>& getTransactionHistory() const;
};

#endif
```

Inventory

Auto_Tracking_Inventory_System (Global Scope)

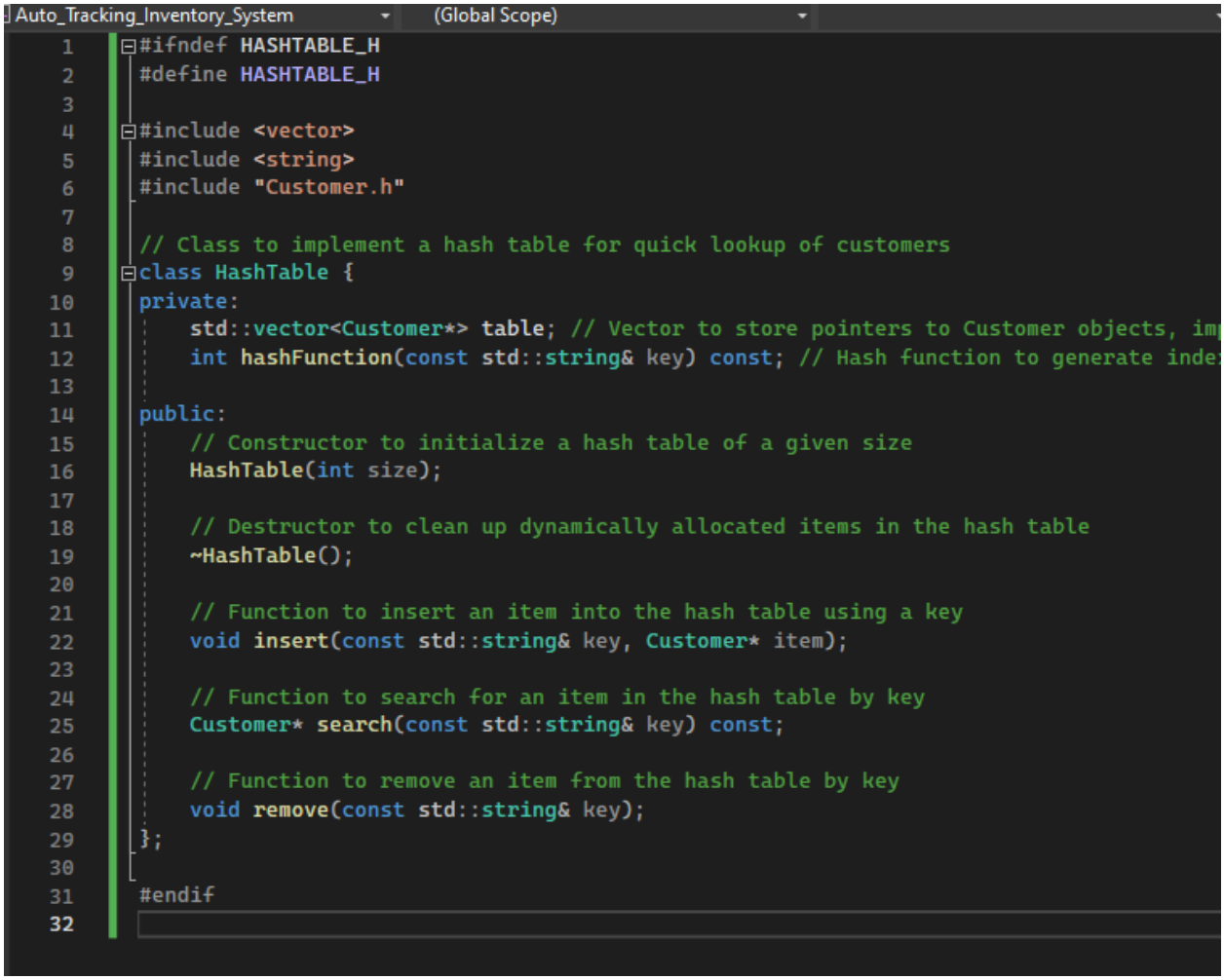
```
#ifndef INVENTORY_H
#define INVENTORY_H

#include "Movie.h"
#include <vector>
#include <map>
#include <string>

// Class to manage the inventory of movies in the rental store
class Inventory {
private:
    std::map<std::string, std::vector<Movie*>> inventory; // Map to store movies by ge

public:
    // Constructor to initialize an empty inventory
    Inventory();

    // Destructor to clean up dynamically allocated movies
    ~Inventory();

    // Function to add a movie to the inventory
    void addMovie(Movie* movie);

    // Function to borrow a movie from the inventory, reducing its stock
    Movie* borrowMovie(const std::string& identifier);

    // Function to return a movie to the inventory, increasing its stock
    void returnMovie(Movie* movie);

    // Function to display the entire inventory, sorted by genre and then by sorting c
    void displayInventory() const;
};

#endif
```

Transaction

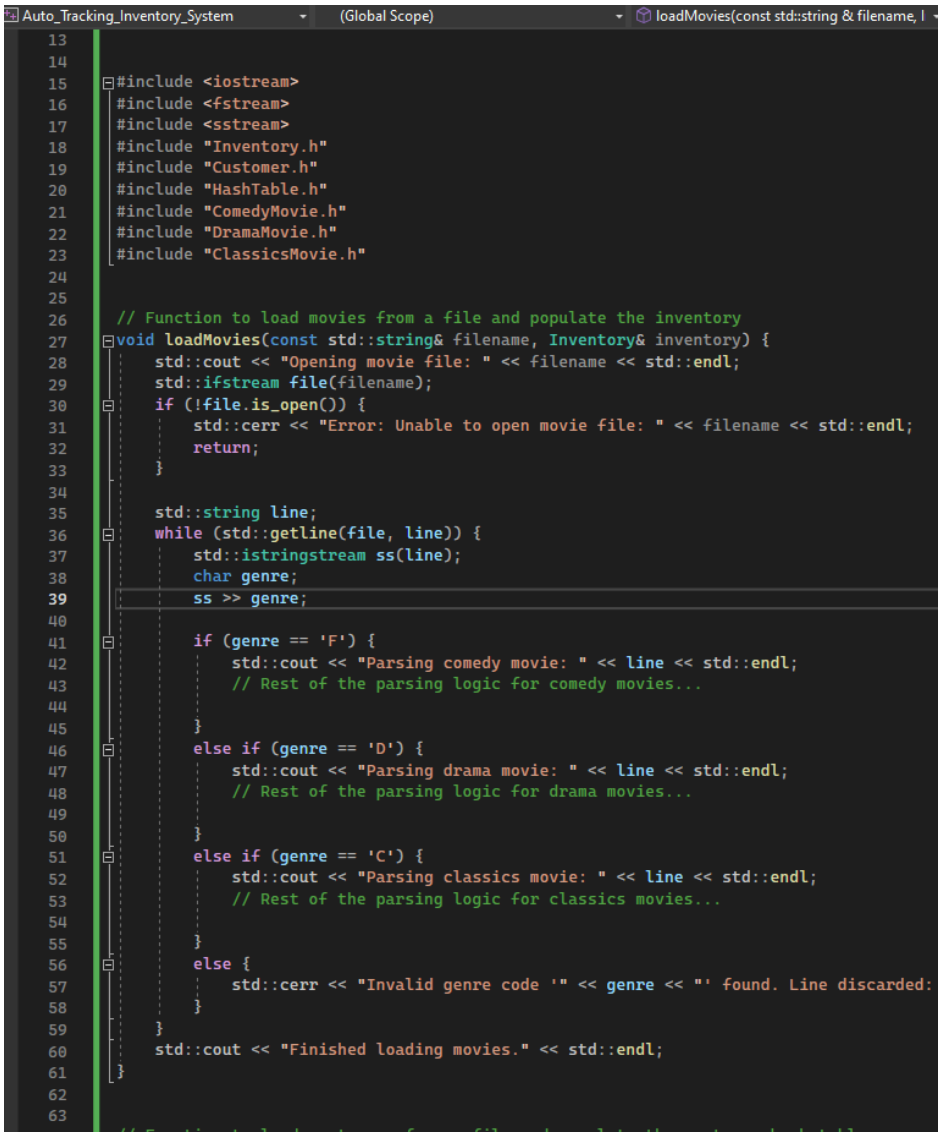



```h
#ifndef TRANSACTION_H
#define TRANSACTION_H

#include <string>
#include "Movie.h" // Assuming Transaction uses Movie

class Transaction {
private:
    std::string action; // Action of the transaction ('Borrow' or 'Return')
    Movie* movie;       // Pointer to the movie involved in the transaction

public:
    // Constructor to initialize a transaction with action and movie
    Transaction(const std::string& action, Movie* movie);

    // Getter for the action
    std::string getAction() const;

    // Getter for the movie
    Movie* getMovie() const;

    // Function to display the details of the transaction
    void display() const;
};

#endif
```

HashTable


Auto_Tracking_Inventory_System
(Global Scope)
#ifndef HASHTABLE_H
#define HASHTABLE_H
#include <vector>
#include <string>
#include "Customer.h"
// Class to implement a hash table for quick lookup of customers
class HashTable {
private:
std::vector<Customer*> table; // Vector to store pointers to Customer objects, im
int hashFunction(const std::string& key) const; // Hash function to generate inde
public:
// Constructor to initialize a hash table of a given size
HashTable(int size);
// Destructor to clean up dynamically allocated items in the hash table
~HashTable();
// Function to insert an item into the hash table using a key
void insert(const std::string& key, Customer* item);
// Function to search for an item in the hash table by key
Customer* search(const std::string& key) const;
// Function to remove an item from the hash table by key
void remove(const std::string& key);
};
#endif

MovieStore


Auto_Tracking_Inventory_System
(Global Scope)
loadMovies(const std::string & filename, I
#include <iostream>
#include <fstream>
#include <sstream>
#include "Inventory.h"
#include "Customer.h"
#include "HashTable.h"
#include "ComedyMovie.h"
#include "DramaMovie.h"
#include "ClassicsMovie.h"
// Function to load movies from a file and populate the inventory
void loadMovies(const std::string& filename, Inventory& inventory) {
std::cout << "Opening movie file: " << filename << std::endl;
std::ifstream file(filename);
if (!file.is_open()) {
std::cerr << "Error: Unable to open movie file: " << filename << std::endl;
return;
}
std::string line;
while (std::getline(file, line)) {
std::istringstream ss(line);
char genre;
ss >> genre;
if (genre == 'F') {
std::cout << "Parsing comedy movie: " << line << std::endl;
// Rest of the parsing logic for comedy movies...
}
else if (genre == 'D') {
std::cout << "Parsing drama movie: " << line << std::endl;
// Rest of the parsing logic for drama movies...
}
else if (genre == 'C') {
std::cout << "Parsing classics movie: " << line << std::endl;
// Rest of the parsing logic for classics movies...
}
else {
std::cerr << "Invalid genre code '" << genre << "' found. Line discarded:
}
}
std::cout << "Finished loading movies." << std::endl;
}

```cpp

// Function to load customers from a file and populate the customer hash table
void loadCustomers(const std::string& filename, HashTable& customers) {
    std::cout << "Opening customer file: " << filename << std::endl;
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Error: Unable to open customer file: " << filename << std::endl;
        return;
    }

    std::string line;
    while (std::getline(file, line)) {
        std::istringstream ss(line);
        int customerID;
        std::string firstName, lastName;
        ss >> customerID >> lastName >> firstName;

        std::cout << "Adding customer: " << customerID << ", " << firstName << " " <<
        Customer* customer = new Customer(customerID, firstName, lastName);
        customers.insert(std::to_string(customerID), customer);
    }
    std::cout << "Finished loading customers." << std::endl;
}


// Function to process commands from a file
void processCommands(const std::string& filename, Inventory& inventory, HashTable& cus
    std::cout << "Opening command file: " << filename << std::endl;
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Error: Unable to open command file: " << filename << std::endl;
        return;
    }

    std::string line;
    while (std::getline(file, line)) {
        std::istringstream ss(line);
        char command;
        ss >> command;

        std::cout << "Processing command: " << command << " | " << line << std::endl;

        if (command == 'B') {
            std::cout << "Borrow command detected." << std::endl;
            // Borrow command logic...

        }
        else if (command == 'R') {
            std::cout << "Return command detected." << std::endl;
```

```cpp

int main() {
    std::cout << "Starting program..." << std::endl;

    // Initialize the inventory and customer hash table
    Inventory inventory;
    HashTable customers(100); // Assuming a hash table size of 100 for simplicity

    // Output for debugging
    std::cout << "Loading movie data from file..." << std::endl;
    loadMovies("data4movies.txt", inventory);
    std::cout << "Movie data loaded successfully." << std::endl;

    std::cout << "Loading customer data from file..." << std::endl;
    loadCustomers("data4customers.txt", customers);
    std::cout << "Customer data loaded successfully." << std::endl;

    std::cout << "Processing commands from file..." << std::endl;
    processCommands("data4commands.txt", inventory, customers);
    std::cout << "Commands processed successfully." << std::endl;

    std::cout << "Program finished successfully." << std::endl;

    return 0;
}
```