AGH

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

Object oriented programming and software engineering

Cinema Tickets Vending Machine

project report

by

Marcin Walaszek

Nikodem Weltrowski

1. Introduction:

The Cinema Tickets Vending Machine project is designed to simulate an automated ticket vending machine for a cinema. The main objective of this project was to create an interactive, user-friendly system that allows customers to view available movie screenings, select seats, purchase tickets, and complete transactions. This project demonstrates the use of object-oriented programming concepts such as encapsulation, inheritance, and file handling in C++.

2. Code:

Below are key parts of the code used in the project, divided into sections with explanations of the thought process behind each part.

main.cpp

This file contains the main logic for interacting with the user. It includes displaying the list of movies, handling user input for selecting movies and purchasing tickets, and managing the payment process.

```
#include "other.h"

#include <iostream>

#include <string>

#include <vector>

#include <cctype>

#include <limits>

#include <cstdlib>


using namespace std;


// Function to validate seat format

// Ensures that the seat is in the format [A-E][1-6]

bool isValidSeat(const string& seat) {

    if (seat.length() != 2) return false;

    if (!isalpha(seat[0]) || !isdigit(seat[1])) return false;

    char col = toupper(seat[0]);

    int row = seat[1] - '0';

    return (col >= 'A' && col <= 'E' && row >= 1 && row <= 6);
```

```cpp
}

int main() {
    // Load movie screenings from file
    Theater theater = getMovies();

    // Set genre and minimum age for movies
    theater.setGenreForMovie("GWTW", "Drama");
    theater.setMinAgeForMovie("GWTW", 12);

    theater.setGenreForMovie("GBaU", "Comedy");
    theater.setMinAgeForMovie("GBaU", 15);

    theater.setGenreForMovie("AVATAR", "Sci-Fi");
    theater.setMinAgeForMovie("AVATAR", 13);

    theater.setGenreForMovie("AVATAR2", "Sci-Fi");
    theater.setMinAgeForMovie("AVATAR2", 13);

    while (true) {
        // List movies available in the theater
        for (size_t i = 0; i < theater.getScreenings().size(); ++i) {
            cout << i + 1 << ". " << theater.getScreenings()[i].getMovieName() << endl;
        }
        cout << theater.getScreenings().size() + 1 << ". Exit" << endl;

        // Take user input to select a movie or exit
        cout << "Enter the number of the movie to see details or 'exit' to quit: ";
        string input;
        cin >> input;

        if (input == "exit" || input == to_string(theater.getScreenings().size() + 1)) {
            break; // Exit the program
        }
```

```
try {
    size_t choice = stoi(input); // Convert user input to a number
    if (choice > 0 && choice <= theater.getScreenings().size()) {
        while (true) {
            // Display selected movie details
            theater.getScreenings()[choice - 1].display();

            cout << "1. Purchase ticket(s)" << endl;
            cout << "2. Return" << endl;

            // Take user input to purchase tickets or return to the movie list
            cout << "Enter your choice: ";
            string subInput;
            cin >> subInput;

            if (subInput == "2") {
                break; // Return to the movie list
            } else if (subInput == "1") {
                // Check available seats
                int availableSeats = 30 - theater.getScreenings()[choice - 1].getTotalOccupiedSeats();
                if (availableSeats == 0) {
                    // No seats available
                    cout << "No seats available. Please return to the previous menu." << endl;
                    cout << "1. Return" << endl;
                    while (true) {
                        cout << "Enter your choice: ";
                        string returnInput;
                        cin >> returnInput;
                        if (returnInput == "1") {
                            break;
                        } else {
                            cout << "Invalid choice. Please try again." << endl;
                        }
                    }
                    break;
                }

                // Ask for the number of reduced tickets
                int numReducedTickets;
                while (true) {
                    cout << "Enter number of reduced tickets (50% off): ";
                    cin >> numReducedTickets;
                    if (cin.fail() || numReducedTickets < 0 || numReducedTickets > availableSeats) {
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        cout << "Invalid number. Please enter a number between 0 and " <<
        availableSeats << "." << endl;
                    } else {
                        break;
                    }
                }

                // Ask for the number of normal tickets
                int remainingSeats = availableSeats - numReducedTickets;
                int numNormalTickets;
                while (true) {
                    cout << "Enter number of normal tickets: ";
                    cin >> numNormalTickets;
                    if (cin.fail() || numNormalTickets < 0 || numNormalTickets > remainingSeats) {
                        cin.clear();
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        cout << "Invalid number. Please enter a number between 0 and " <<
        remainingSeats << "." << endl;
                    } else {
                        break;
                    }
                }
```

```
int numTickets = numReducedTickets + numNormalTickets;

// Ask for seat selection for each ticket
vector<string> seats;
for (int i = 0; i < numTickets; ++i) {
    while (true) {
        cout << "Choose seat (" << i + 1 << " of " << numTickets << "): ";
        string seat;
        cin >> seat;
        if (seat == "exit") {
            goto endPurchase; // Exit seat selection
        }
        if (!isValidSeat(seat)) {
            cout << "Invalid seat. Please enter in the format [A-E][1-6] (e.g., B2)." << endl;
        } else {
            int row = seat[1] - '0';
            char col = toupper(seat[0]);
            if (theater.getScreenings()[choice - 1].isSeatAvailable(row, col)) {
                seats.push_back(seat);
                theater.getScreenings()[choice - 1].occupySeat(row, col);
                break;
            } else {
                cout << "Seat " << seat << " is already occupied. Choose another seat." <<
endl;
            }
        }
    }
}

// Calculate the total price based on the number of reduced and normal tickets
theater.getScreenings()[choice - 1].calculateAndSetTotalPrice(numReducedTickets,
numNormalTickets);
double totalPrice = theater.getScreenings()[choice - 1].getPrice();

// Payment process
```

```
                        while (true) {
                            cout << "1. Pay " << totalPrice << " PLN" << endl;
                            cout << "2. Exit" << endl;

                            cout << "Enter your choice: ";
                            string payInput;
                            cin >> payInput;

                            if (payInput == "2") {
                                goto endPurchase; // Exit payment process
                            } else if (payInput == "1") {
                                cout << "Transaction complete" << endl;
                                theater.saveScreeningsToFile("movies"); // Save updated screenings to file
                                exit(0); // Exit the program after transaction is complete
                            } else {
                                cout << "Invalid input. Please try again." << endl;
                            }
                        }
                    } else {
                        cout << "Invalid choice. Please try again." << endl;
                    }
                }
            } else {
                cout << "Invalid choice. Please try again." << endl;
            }
        } catch (invalid_argument&) {
            cout << "Invalid input. Please enter a number or 'exit'." << endl;
        }

endPurchase:
        continue; // Return to the main loop
    }

    return 0;}
```

getMovies.cpp

This file handles the implementation of the Movie and Screening classes, including their methods for getting and setting attributes, checking seat availability, and saving screening details to a file. It also includes the Theater class for managing a collection of screenings.

```
#include "other.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <cctype>

using namespace std;

// Default constructor for the Movie class
// Initializes the movie with default values
Movie::Movie() : movieName("Not specified"), genre("None"), minAge(0) {}

// Parameterized constructor for the Movie class
// Initializes the movie with provided values
Movie::Movie(const string& name, const string& genre, int minAge)
    : movieName(name), genre(genre), minAge(minAge) {}

// Getter for movieName
string Movie::getMovieName() const {
    return movieName;
}

// Setter for movieName
void Movie::setMovieName(const string& name) {
    movieName = name;
}

// Getter for genre
```

```
string Movie::getGenre() const {
    return genre;
}

// Setter for genre
void Movie::setGenre(const string& genre) {
    this->genre = genre;
}

// Getter for minAge
int Movie::getMinAge() const {
    return minAge;
}

// Setter for minAge with validation
void Movie::setMinAge(int age) {
    if (age >= 0 && age <= 120) {
        minAge = age;
    } else {
        cout << "Invalid age value assigned." << endl;
    }
}

// Parameterized constructor for the Screening class
// Initializes the screening with provided values and counts occupied seats
Screening::Screening(const string& movieName, const string& date, const string& hour, double price,
const vector<bool>& occupancy)
    : Movie(movieName, "None", 0), date(date), hour(hour), price(price), occupancy(occupancy),
totalOccupiedSeats(0), totalPrice(0.0) {
    totalOccupiedSeats = count(occupancy.begin(), occupancy.end(), true);
}

// Default constructor for the Screening class
// Initializes the screening with default values
Screening::Screening()
```

```
    : Movie("Unknown", "None", 0), date("Unknown"), hour("Unknown"), price(0.0), occupancy(30,
false), totalOccupiedSeats(0), totalPrice(0.0) {}


// Display method with default option to show price

// Displays screening details including movie information

void Screening::display() const {

    display(true); // Default to showing price

}


// Overloaded display method

// Optionally displays the price along with screening details

void Screening::display(bool showPrice) const {

    cout << "Movie: " << movieName << endl;

    cout << "Genre: " << getGenre() << endl;

    cout << "Min Age: " << getMinAge() << endl;

    cout << "Date: " << date << endl;

    cout << "Hour: " << hour << endl;

    if (showPrice) {

        cout.precision(2);

        cout << fixed << "Price: " << price << " PLN" << endl;

    }

    int availableSeats = 30 - totalOccupiedSeats;

    cout << "Available Seats: " << availableSeats << endl;

    cout << "Occupancy:" << endl;


    // Display column headers

    cout << "  A B C D E" << endl;


    // Display grid with numbered rows

    for (int row = 0; row < 6; ++row) {

        cout << row + 1 << " ";

        for (int col = 0; col < 5; ++col) {

            int index = row * 5 + col;

            if (index < occupancy.size()) {

                cout << (occupancy[index] ? "X" : "O") << " "; // 'X' for occupied, 'O' for available
            } else {
                cout << "0 "; // Default to 0 if index is out of bounds
            }
        }
        cout << endl;
    }
}

// Getter for date
string Screening::getDate() const {
    return date;
}

// Setter for date
void Screening::setDate(const string& date) {
    this->date = date;
}

// Getter for hour
string Screening::getHour() const {
    return hour;
}

// Setter for hour
void Screening::setHour(const string& hour) {
    this->hour = hour;
}

// Getter for price
double Screening::getPrice() const {
    return price;
}

// Setter for price with discount logic
// Applies a 30% discount if the screening is before 15:00
void Screening::setPrice(double price) {
    int movieHour = stoi(hour.substr(0, 2));
    if (movieHour < 15) {
        price *= 0.7; // Apply 30% discount
    }
    if (price >= 0) { // Example condition
        this->price = price;
    }
}

// Getter for occupancy
vector<bool> Screening::getOccupancy() const {
    return occupancy;
}

// Setter for occupancy
// Updates the occupancy status and recalculates the total occupied seats
void Screening::setOccupancy(const vector<bool>& occupancy) {
    this->occupancy = occupancy;
    totalOccupiedSeats = count(occupancy.begin(), occupancy.end(), true); // Recalculate total
occupied seats
}

// Getter for totalOccupiedSeats
int Screening::getTotalOccupiedSeats() const {
    return totalOccupiedSeats;
}

// Setter for totalOccupiedSeats
void Screening::setTotalOccupiedSeats(int totalOccupiedSeats) {
    if (totalOccupiedSeats >= 0) {
        this->totalOccupiedSeats = totalOccupiedSeats;
    }
}
```

```
// Check if a seat is available
// Returns true if the seat at the specified row and column is available
bool Screening::isSeatAvailable(int row, char col) const {
    int colIndex = toupper(col) - 'A';
    int index = (row - 1) * 5 + colIndex;
    return (index >= 0 && index < occupancy.size() && !occupancy[index]);
}

// Mark a seat as occupied
void Screening::occupySeat(int row, char col) {
    int colIndex = toupper(col) - 'A';
    int index = (row - 1) * 5 + colIndex;
    if (index >= 0 && index < occupancy.size() && !occupancy[index]) {
        occupancy[index] = true;
        totalOccupiedSeats++;
    }
}

// Save screening details to file
// Writes the screening details and occupancy status to a file
void Screening::saveToFile(ofstream& file) const {
    file << movieName << "," << date << "," << hour << "," << price << ",";
    for (size_t i = 0; i < occupancy.size(); ++i) {
        file << (occupancy[i] ? "1" : "0");
        if (i < occupancy.size() - 1) {
            file << ",";
        }
    }
    file << endl;
}

// Calculate and set the total price based on the number of reduced and normal tickets
void Screening::calculateAndSetTotalPrice(int numReducedTickets, int numNormalTickets) {
    double reducedPrice = price * 0.5;
    totalPrice = (numReducedTickets * reducedPrice) + (numNormalTickets * price);
    setPrice(totalPrice);
}

// Adds a screening to the theater
void Theater::addScreening(const Screening& screening) {
    screenings.push_back(screening);
}

// Loads screenings from a file
// Reads screening details from a file and adds them to the theater
void Theater::loadScreeningsFromFile(const string& filename) {
    ifstream file(filename);

    if (file.is_open()) {
        string line;
        bool isFirstLine = true;

        while (getline(file, line)) {
            if (isFirstLine) {
                isFirstLine = false;
                continue; // Skip header line
            }

            stringstream lineStream(line);
            string movieName, date, hour, cell;
            double price;
            vector<bool> occupancy;

            // Read movie name
            getline(lineStream, movieName, ',');

            // Read date
            getline(lineStream, date, ',');

            // Read hour
            getline(lineStream, hour, ',');

            // Read price
            getline(lineStream, cell, ',');
            price = stod(cell);

            // Read occupancy
            while (getline(lineStream, cell, ',')) {
                occupancy.push_back(cell == "1");
            }

            Screening screening;
            screening.setMovieName(movieName);
            screening.setDate(date);
            screening.setHour(hour);
            screening.setPrice(price);
            screening.setOccupancy(occupancy);

            addScreening(screening);
        }

        file.close();
    } else {
        cerr << "Unable to open file: " << filename << endl;
    }
}

// Saves screenings to a file
// Writes the details of all screenings in the theater to a file
void Theater::saveScreeningsToFile(const string& filename) const {
    ofstream file(filename);
    if (file.is_open()) {
        // Write header line
        file << "Name,Date,Hour,Price,Occupancy" << endl;

        // Write each screening to file
        for (const auto& screening : screenings) {
            screening.saveToFile(file);
        }

        file.close();
    } else {
        cerr << "Unable to open file: " << filename << endl;
    }
}

// Getter for the list of screenings
vector<Screening>& Theater::getScreenings() {
    return screenings;
}

// Sets the genre for a specific movie
// Updates the genre for all screenings of the specified movie
void Theater::setGenreForMovie(const string& movieName, const string& genre) {
    for (auto& screening : screenings) {
        if (screening.getMovieName() == movieName) {
            screening.setGenre(genre);
        }
    }
}

// Sets the minimum age for a specific movie
// Updates the minimum age for all screenings of the specified movie
void Theater::setMinAgeForMovie(const string& movieName, int minAge) {
    for (auto& screening : screenings) {
        if (screening.getMovieName() == movieName) {
            screening.setMinAge(minAge);
        }
    }
}

// Function to load movies from the file
Theater getMovies() {
    string filename = "movies";
    Theater theater;
    theater.loadScreeningsFromFile(filename);
    return theater;
}
```

other.h

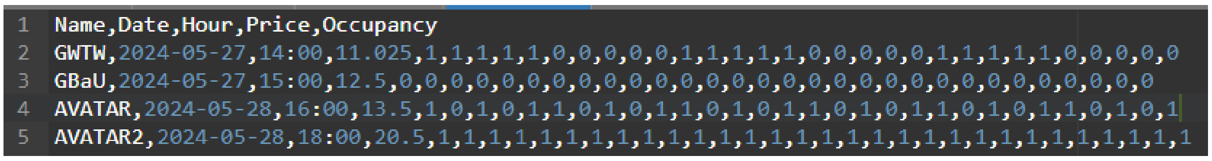
This file contains the declarations of the Movie, Screening, and Theater classes. The Movie class encapsulates basic movie details, the Screening class inherits from Movie and adds screening-specific details, and the Theater class manages a collection of screenings.

```
#ifndef OTHER_H
#define OTHER_H

#include <vector>
#include <string>

using namespace std;

// The Movie class represents basic information about a movie.
class Movie {
protected:
    string movieName; // Name of the movie, accessible by derived classes

private:
    string genre;     // Genre of the movie
    int minAge;       // Minimum age to watch the movie

public:
    // Constructors
    Movie();
    Movie(const string& name, const string& genre, int minAge);

    // Getters and Setters
    string getMovieName() const; // Returns the name of the movie
    void setMovieName(const string& name); // Sets the name of the movie

    string getGenre() const; // Returns the genre of the movie
    void setGenre(const string& genre); // Sets the genre of the movie

    int getMinAge() const; // Returns the minimum age required to watch the movie
    void setMinAge(int age); // Sets the minimum age with validation
};

// The Screening class represents a movie screening, inheriting from the Movie class.
class Screening : public Movie {
private:
    string date; // Date of the screening
    string hour; // Hour of the screening
    double price; // Price of a single ticket
    vector<bool> occupancy; // Seat occupancy status (true if occupied, false if available)
    int totalOccupiedSeats; // Total number of occupied seats
    double totalPrice; // Total price for the purchased tickets

public:
    // Constructors
    Screening();
    Screening(const string& movieName, const string& date, const string& hour, double price, const
vector<bool>& occupancy);
```

```
    // Getters and Setters for Screening attributes
    string getDate() const; // Returns the date of the screening
    void setDate(const string& date); // Sets the date of the screening

    string getHour() const; // Returns the hour of the screening
    void setHour(const string& hour); // Sets the hour of the screening

    double getPrice() const; // Returns the price of a single ticket
    void setPrice(double price); // Sets the price of a ticket, applies discount if before 15:00

    vector<bool> getOccupancy() const; // Returns the occupancy status of the seats
    void setOccupancy(const vector<bool>& occupancy); // Sets the occupancy status of the seats

    int getTotalOccupiedSeats() const; // Returns the total number of occupied seats
    void setTotalOccupiedSeats(int totalOccupiedSeats); // Sets the total number of occupied seats

    // Other methods
    bool isSeatAvailable(int row, char col) const; // Checks if a seat is available
    void occupySeat(int row, char col); // Marks a seat as occupied
    void saveToFile(ofstream& file) const; // Saves the screening details to a file
    void calculateAndSetTotalPrice(int numReducedTickets, int numNormalTickets); // Calculates and
sets the total price for tickets

    // Display methods
    void display() const; // Displays screening details with price
    void display(bool showPrice) const; // Displays screening details, optionally with price
};

// The Theater class manages a collection of movie screenings.
class Theater {
public:
    void addScreening(const Screening& screening); // Adds a screening to the theater
    void loadScreeningsFromFile(const string& filename); // Loads screenings from a file
    void saveScreeningsToFile(const string& filename) const; // Saves screenings to a file
    vector<Screening>& getScreenings(); // Returns a list of screenings
    // Methods to set genre and minimum age for movies

    void setGenreForMovie(const string& movieName, const string& genre); // Sets the genre for a specific movie

    void setMinAgeForMovie(const string& movieName, int minAge); // Sets the minimum age for a specific movie


private:

    vector<Screening> screenings; // List of screenings in the theater

};


// Function to load movies from the file

Theater getMovies();


#endif // OTHER_H
```

‘movies’ file

The movies file is a crucial part of the Cinema Tickets Vending Machine project, storing all the necessary information about the movie screenings available in the theater. The file uses a CSV (Comma-Separated Values) format to organize data in a structured manner.



```
Name,Date,Hour,Price,Occupancy
GWTW,2024-05-27,14:00,11.025,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0
GBaU,2024-05-27,15:00,12.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
AVATAR,2024-05-28,16:00,13.5,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1
AVATAR2,2024-05-28,18:00,20.5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

The movies file contains multiple lines, where each line represents a movie screening with specific details. The first line is a header that labels each column for easy identification. Each subsequent line corresponds to a screening with the following fields:

Name: The name of the movie.

Date: The date of the screening.

Hour: The time of the screening.

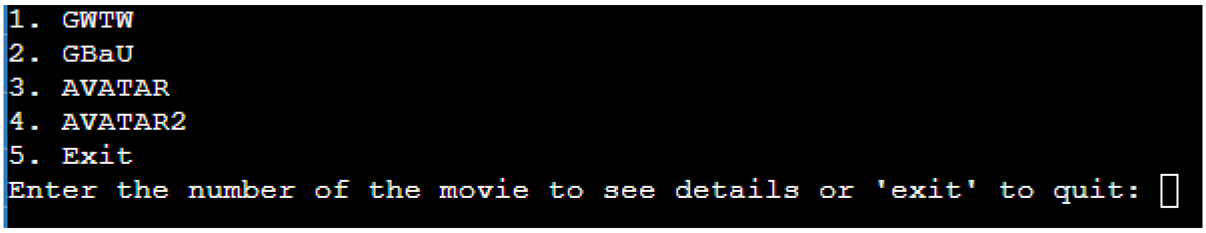
Price: The price of a single normal ticket for the screening.

Occupancy: A series of binary values indicating the occupancy status of each seat (1 for occupied, 0 for available).

3. Program operation

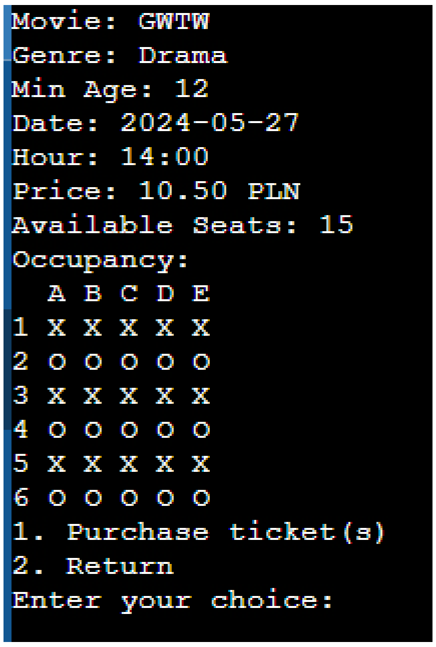
Here are a few screenshots from the operation of the program in order to demonstrate how it works.

First, the main menu presents the user with options to select a movie or exit the program:



```
1. GWTW
2. GBaU
3. AVATAR
4. AVATAR2
5. Exit
Enter the number of the movie to see details or 'exit' to quit:
```

After selecting a movie, details about the movie are displayed, and the user is given the options to purchase tickets for the movie or return to the main menu.

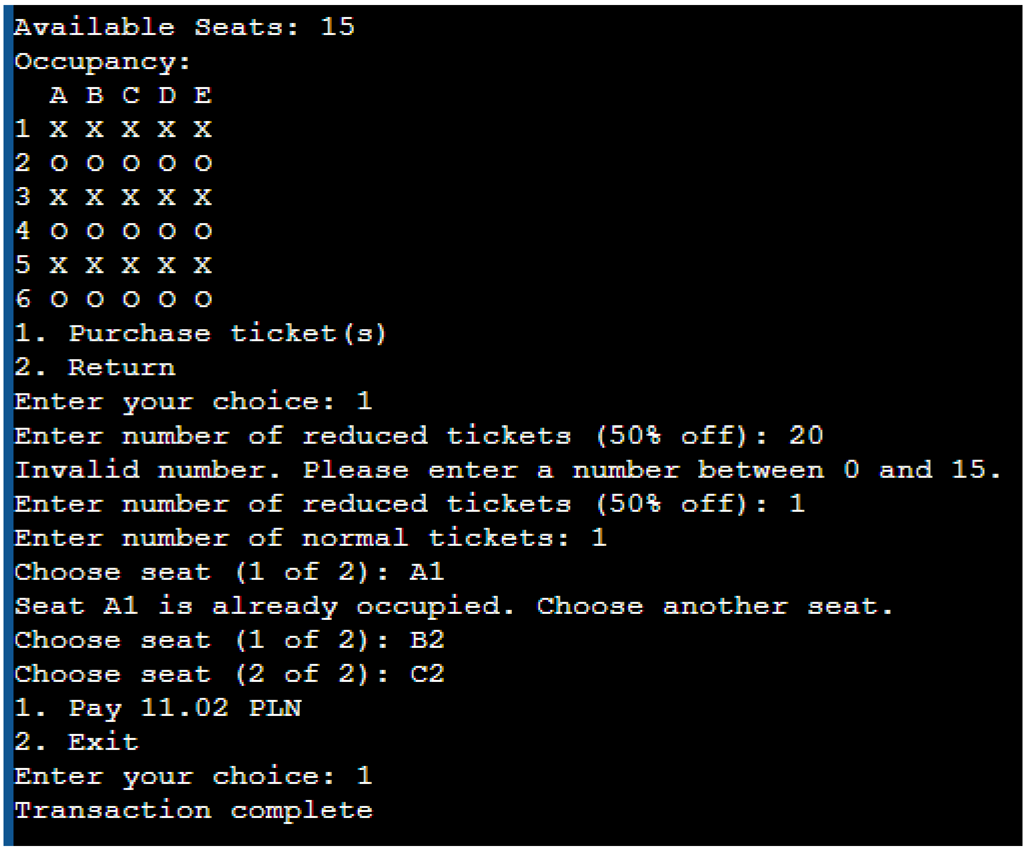


```
Movie: GWTW
Genre: Drama
Min Age: 12
Date: 2024-05-27
Hour: 14:00
Price: 10.50 PLN
Available Seats: 15
Occupancy:
  A B C D E
1 X X X X X
2 O O O O O
3 X X X X X
4 O O O O O
5 X X X X X
6 O O O O O
1. Purchase ticket(s)
2. Return
Enter your choice:
```

Once the user decides to purchase tickets, they will be asked to specify the number of reduced and normal tickets respectively. Then, they will be able to choose from the available seats in the theater.

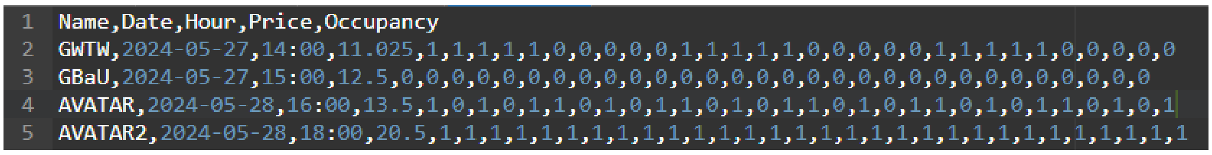
If the user tries to buy more tickets than there are available seats or selects an already occupied seat, they will be prompted to change their choice.

After all selections are made, the total price for the tickets is calculated and the user is given the options to pay or cancel the purchase and return to the main menu. If they choose to pay, the program terminates and the information about the newly occupied seats is written into the 'movies' file.



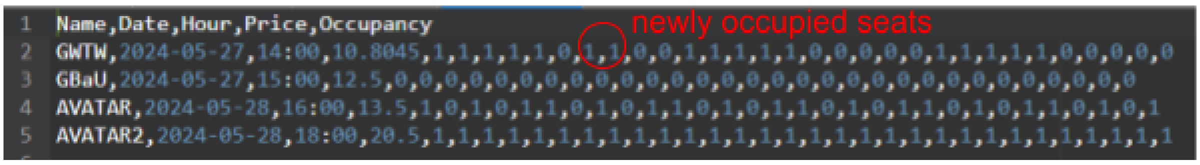
```
Available Seats: 15
Occupancy:
  A B C D E
1 X X X X X
2 O O O O O
3 X X X X X
4 O O O O O
5 X X X X X
6 O O O O O
1. Purchase ticket(s)
2. Return
Enter your choice: 1
Enter number of reduced tickets (50% off): 20
Invalid number. Please enter a number between 0 and 15.
Enter number of reduced tickets (50% off): 1
Enter number of normal tickets: 1
Choose seat (1 of 2): A1
Seat A1 is already occupied. Choose another seat.
Choose seat (1 of 2): B2
Choose seat (2 of 2): C2
1. Pay 11.02 PLN
2. Exit
Enter your choice: 1
Transaction complete
```

'movies' file before the purchase:



```
Name,Date,Hour,Price,Occupancy
GWTW,2024-05-27,14:00,11.025,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0
GBaU,2024-05-27,15:00,12.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
AVATAR,2024-05-28,16:00,13.5,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1
AVATAR2,2024-05-28,18:00,20.5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

'movies' file after the purchase:



```
Name,Date,Hour,Price,Occupancy
GWTW,2024-05-27,14:00,10.8045,1,1,1,1,1,0,1,1,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0
GBaU,2024-05-27,15:00,12.5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
AVATAR,2024-05-28,16:00,13.5,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1,1,0,1,0,1
AVATAR2,2024-05-28,18:00,20.5,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
```

4. Conclusions:

The Cinema Tickets Vending Machine project was successfully implemented, providing users with the ability to view available movie screenings, select seats, and purchase tickets. The project demonstrated the use of object-oriented programming principles such as encapsulation and inheritance. Additionally, file handling was utilized to store and retrieve movie screening data.

Functionality Implemented Successfully:

- Viewing available movie screenings
- Displaying movie details including visualization of seats occupancy
- Selecting seats for a chosen screening
- Calculating and displaying the total price for tickets
- Completing the purchase transaction and saving data to a file

Potential Improvements:

- Implementing user authentication to manage different user roles (e.g., admin, customer)
- Adding more detailed error handling and validation
- Enhancing the user interface for a more intuitive experience
- Integrating with a payment gateway for transactions
- Providing language selection

This project provided a comprehensive learning experience in C++ programming and demonstrated the practical application of object-oriented design principles.