



FINAL PROJECT COVER LETTER

(Group Assignment)

Student Names :

Ardelia Shaula Araminta	2440065163
Sri Kalyan Rohan	2440090266
Raphael Reynaldi	2440071973

Course Code : COMP 6571

Course Name :
Data Structures and Algorithm

Class : L2BC

Name of Lecturer :
Maria Seraphina

Major : Computer Science

Title of Assignment :
Cinema Ticketing

Type of Assignment : Final Project

Due Date : 28/06/2021

Submission Date : 28/06/2021

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Problem Description

When our group first got together, we looked at trying to tackle a project that was something relatable so that it would be easier for us to build upon it. After some discussions, one of our group members shared an anecdote where they faced problems when trying to buy a ticket at the cinema. There was a conflicting situation where two people shared the same seat and ultimately, our group member had to end up sitting further away from the cinema screen which was not what she had hoped for after booking the original seat. We wanted to try to solve this issue by creating our own ticketing system which will aim to solve this issue using a variety of data structures.

Proposed Data Structure

To overcome our major challenge, we decided to use a Linked List and queue approach. Based on our research, we decided to use the Doubly Circular Linked List since it is the most effective in movie ticketing systems, including queue and seat reservations. This linked list is the ideal data structure for seating arrangements since it allows customers to book seats at the far end, in the front, and in the middle of the hall. In addition, because it is a first come, first served method, the line will be used for ticket booking.

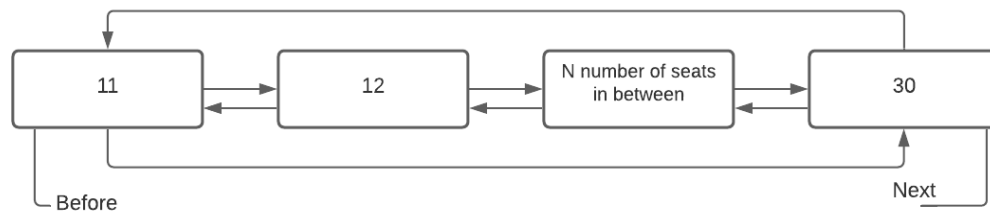
How the Data Structure Solved the Problem

The circular linked list will be used to represent the different seats within the cinema. While the queue will be used to determine which user is able to book the seat first. After some discussions, and talk amongst each of the groupmates, we felt that a doubly circular linked list best suits the solution to the problem due to the characteristics it carries which can be effectively applied to the problem that we're tackling. Furthermore, we will pair with the queue data structure to allow a system which will ensure that users won't collide with each other when booking their tickets.

Theoretical Analysis:

i) Doubly Circular Linked List

In a doubly circular linked list every node links to the next node and the last node will link to the first node and allows reversal and it doesn't contain NULL in any of the nodes. Circular doubly linked list structure is made up of three components and it requires more space per node but it provides easy manipulation of the pointers and the searching becomes more efficient. However, the algorithm will find the shortest path to the selected seat that the customer wants to book, since it can access either the previous pointer or the next pointer based on the seat number. It enables customers to book a seat from any seat while avoiding multiple bookings.



On the other hand, a singular linked list, every node links to its next node in the sequence and the last node will be NULL. Thus, a singly linked list is not suitable for the booking system since the program will exit on the first iteration when the first customer booked his/her seat as it can't access the head of the node. Since the seats are connected, we use a Doubly Circular Linked List so that we can go directly to the next row seats and the client can reserve seats anywhere in the cinema by using the prior pointer, as the previous pointer of the head node address of seat 30 is stored (based on the image above).

ii) Queue

In queue, it uses First in First-Out methodology. The one end is always used to input data (enqueue), whereas the other end is always used to delete data (dequeue) (dequeue). The queue is really useful in our situation because the booking lane is one-way, with the client in front of the ticketing desk able to book the seat first and complete it.

For our final project, we implement the combination of queue and doubly circular linked list. Our code will start from the register and login, followed by choosing the movie and booking the seat.

Code Commenting

1. loginregister.cpp

Login and Register

```
#include<iostream>
#include<iostream>
#include<fstream>
#include<stdlib.h>
#include<string.h>

using namespace std;

class Account {
private:
    // create variables of user, password, user authentication
    int count;
    // login
    string user,password,usr,pwd;
    //register
    string regUser,regPass,ru,rp;

public:
    void accRegister();
    bool accLogin();
};
```

In the class Account, we make 4 variables particular for the login which includes; user, password, usr and pwd. Later, the user can login with the registered username and password they entered before. Whereas for the register, we make different variables to store the input username and password to be written in the existing text file.

bool Account::accLogin()

```

bool Account::accLogin(){
    // user name and password
    system("cls");
    cout<<"Please enter"<<endl;
    cout<<"Username :";
    cin>>user;
    cout<<"Password :";
    cin>>password;

    //read the text file and compare the user entry and the existing text file
    // for both username and password
    ifstream input("customerData.txt");
    while(input>>usr>>pwd)
    {
        if(usr==user && pwd==password)
        {
            count=1;
            system("cls");
        }
    }

    input.close();

    // then it notifies the customer if the login is unsuccessful or success
    if(count==1)
    {
        cout<<"Hi there "<<user<<"\n Login Successful.\n";
        return true;
    }
    else
    {
        cout<<"\n Login Unsuccessful. ";
        return false;
    }
}

```

Next, we create `bool Account::accLogin()` which is a method for user login.

First, it takes the username and password from the user entry and compares the data with the registered name in the text file, and returns a boolean validation. If it matched, it would return true, proceeding to the customer main menu. On the other hand, if either the username and password don't match with the existing data, it will return false and ask for the customer entry again. This will come useful for data verification.

`void Account::accRegister()`

```

void Account::accRegister(){
    system("cls");

    cout<<"Enter the username :";
    cin>>regUser;

    cout<<"\nEnter the password :";
    cin>>regPass;

    // write the data into the text file
    ofstream reg("customerData.txt",ios::app);
    reg<<regUser<<" "<<regPass<<endl;
    system("cls");
    cout<<"\nRegistration Sucessful\n";
}

```

Then, we create void Account::accRegister() which is a method for user registration. It asks for the user username and password to be registered and we use ofstream to write in the text file (customerData.txt).

2. movie.cpp

```

#include<iostream>
#include<stdlib.h>
#include<bits/stdc++.h>
#include<list>
#include "indoXXI.cpp"

using namespace std;

//class for Movie Object
class Movie:public indoXXI{
public:
    string title;
    string time;
    string date;
    //main constructor
    Movie(string title_entry,string time_entry,string date_entry){
        title=title_entry;
        this->time=time_entry;
        this->date=date_entry;
    };
    //null constructor
    Movie(){

    };

    string getTitle();
    string getTime();
    string getDate();
    void setTitle(string title);
    void setTime(string time);
    void setDate(string date);
}

```

In this class Movie, we have inheritance from indoXXI and create variables of movie details. This includes the movie title, time, and date of show and creates the movie constructor that later be useful to showcase during the movie booking, and the program will automatically return movie, title, and date without calling it one by one.

```

string getTitle();
string getTime();
string getDate();
void setTitle(string title);
void setTime(string time);
void setDate(string date);

```

This is the setter and getter for the movie's title, time, date.

```

//getters
string Movie::getTitle(){
    return title;
}

string Movie::getTime(){
    return time;
}

string Movie::getDate(){
    return date;
}

//setters
void Movie::setTitle(string title){
    this->title=title;
}

void Movie::setTime(string time){
    this->time=time;
}

void Movie::setDate(string date){
    this->date=date;
}

```

3. indoXXI.cpp

```

#include<iostream>
#include<stdlib.h>
#define MAX 50

using namespace std;

//creating node class
class Node{
public:

    // initialize node, seat number, seat status, and the customer name
    Node* before;
    Node* link;
    // seat number, seat status initially 0 == null
    int seatNum;
    int seatStat;
    // customer initial
    string cosName;
};

```

We define the max as 50 because this will be our total seat numbers. First, we create a class Node to initialize the pointers before and link to traverse through the nodes of the seat number for the searched item and create a variable of seat number and seat status, and customer name that is initially null. These pointers will be useful to create new temporary nodes that are able to

traverse the number 1-50 for each different customer and store temporary data.

Class Queue

```
//creating queue class
class Queue
{
private:
    //using array for queue
    Node queue[MAX];
    //initializing front and end indices
    int front = -1, rear = -1;

public:
    //queue operations
    void insert(Node);
    void delete_element();
    void display();
};

//push operation
void Queue::insert(Node num){

    //if the alst index is MAX-1, then queue is already full
    if (rear == MAX-1 ){
        cout << "OVERFLOW" << endl;
    }
    //if queue is empty
    }else if(front == -1 && rear == -1){
        front = rear = 0;
        queue[rear] = num;
    }
    //if queue has other elements in it
    }else {
        rear += 1;
        queue[rear] = num;
    }
}
```

This class data structure was used to process the user's request according to whoever connects to the ticking system first. This way, the customers won't experience collisions when trying to book a ticket for a movie. This data structure answers the problem that was mentioned in the beginning where if there were two people trying to book the same seat, that one would be notified if they were successful or unsuccessful in the operation. The insert method is used to place the customers into the queue. The logic we used were if statements to check if the queue is already full, and then checks if there are other elements in the queue or not and if there isn't, then it just inserts the input into the queue. However, if the queue is empty, it will compensate the front and rear and make them -1 to make sure that the element gets pushed into the queue correctly.

```

//pop operation
void Queue::delete_element()
{
    Node val;
    // if queue is empty, we cannot perform the delete or pop operation
    if ( front == -1 || front > rear ) {
        cout << "UNDERFLOW" << endl;
        // if not empty then remove the item at the front index
    } else {
        val = queue[front];
        front+=1;
        cout<<"Node popped: "<<endl;
        cout<<val.cosName<<endl;
        cout<<val.seatNum<<endl;
    }
}
}

```

The pop operation first checks if the queue is empty. If the queue is empty, then the method will not run and will return UNDERFLOW. However, if the queue is not empty it will find the element at the front of the index within the queue and then pop it out. Furthermore, the front increments by one whenever the operation successfully happens to be able to indicate that there is space within the queue being made. We also printed out the seat number and customer name to showcase that it has been successfully popped out of the queue.

Class indoXXI

```

//creating the seating arrangement class
class indoXXI{
public:
    //initializing pointers
    Node* first, * last, * temp;
    //initializing queue
    Queue q;
    indoXXI()
    {
        //setting the first pointer to null to create an empty linked list
        first == NULL;
    }
    //operations
    void addList();
    void displayAll();
    void seatBook();
    void cancelBook();
    void seatAvailable();
};

```

Next, we created a class called indoXXI that contains the public pointers of head, tail and temporary pointer and queue that will be useful for creating a doubly circular linked list later. Then, we set the first pointer to null to create an empty link list. Lastly, we created an empty constructor for different methods that will be used in the ticketing system which includes displaying the seats, seat booking, and cancel book.

void indoXXI::addList()

```

void indoXXI::addList(){
// initialize the seat starts from 1
    int i = 1;
// total seats available
    int numSeat = 50;

    //initializing new node
    temp = new Node;
    temp->seatNum = 1;
    temp->seatStat = 0;
    temp->cosName = "NULL";
    last = first = temp;
}

```

This method initializes the seat starting from 1 and the number of seats which is 50 in total. This initializes a temporary new node, sets the temp seat number starting from 1, as well as setting the seat status which is initially 0 because the customer hasn't booked the seat yet, and sets the head and the tail as temporary nodes. This will be use for storing the temporary node for the first data where the first customer book the seating arrangement.

```

// for the number of seats ( 50 with 10 each rows )

// for loop to iterate the number of seat available
for(int i = 2; i <= numSeat; i++)
{
    // create a node pointer
    // and initialize a new node
    Node *a;
    a = new Node;

    // set a as the new node
    // seat status initially 0 since its null / not booked yet
    // "cosName" or the customer name/initial's value is initially null
    a->seatNum = i;
    a->seatStat = 0;
    a->cosName = "NULL";
}

```

This is a for loop to iterate through the seat number available in the theatre.

Then we set a as a new node and pointer. Then, set the seat status of a node as 0 since it's not booked yet as well as the customer name of a value as null.

```

//create the doubly circular linked list which are nodes that are created using self referential structures.
//previous node -> next pointer ( find last )

//find the last node and set it as a
last->link = a;
// setting up previous and next of new node
a->before = last;
//set the next node
last = a;
// make new node next of old last
last->link = first;
//make last previous of new node or make new node previous of start
first->before = last;
}

```

This is where we create the doubly circular linked list with the nodes that we initialize before as self referential structures. It will find the last node and set it as a and continue to set the previous node as the last node and the next of the nodes. Next, we set the next node as the new head, and the previous head (first-> before) as the tail. This allows the program to do multiple bookings

one after another as it stores the temporary nodes, and also it is able to find the seat with the link pointer (forward) or with the before pointer (previous) and it will be efficient.

void indoXXI::displayAll()

```
void indoXXI::displayAll()
{
    {   int i = 1;
        Node* temp;
        temp = first;
        int count = 0;
```

This method initializes seat number 1 because it starts from 1, temporary node pointer, and sets the temp node used before as the head, and initializes the count = 0. This method is to display and visualize the seat in the console, and it will be updated if the seat status changed from 0-> 1 or 1->, meaning it changed when the customer booked or canceled any specific seat.

```
// display the cinema seating
cout<<"\n -----\n";
cout<<" |          CINEMA SCREEN          | \n";
cout<<"\n -----\n";

// display the seats arrangement if its less than 10 it will add 0 in front of a
// single digit number = 1,2,3,5,6,7,8,9,10

while(temp->link!= first){

    if(temp->seatNum/10 == 0)
        cout<<"0"<<temp->seatNum<<" ";
    else
        cout<<" "<<temp->seatNum<<" ";

    // if the seat status is 0 or empty
    // it will display blank seat
    // if the seat status is 1
    if(temp->seatStat ==0)
        cout<<"|_| ";
    else
        cout<<"|_TAKEN_| ";
        count++;

    if(count%5==0){
        cout<<endl;
        i++;
    }

    //find node having searched value and next node of it
    temp = temp->link;
}

cout<<" "<<temp->seatNum<<" ";
if(temp->seatStat==0)
    cout<<"|_| ";
else
    cout<<"|_TAKEN_| ";
```

In this method, we create a while loop that basically checks the seat arrangement number. If it is less than 10 (single digit numbers) it will add an additional 0 in front of the number so it will be aligned with the other two digits numbers. This will also print out the current state of the seats booking, if the seats are booked it will be displayed with "TAKEN" keyword else, it will just be blank.

void indoXXI::seatBook()

```
void indoXXI::seatBook()
{
    int numSeat;
    string initial;
    // label allows the user to be redirected the label
    label:

    cout << " Please choose your seat/s: " << endl;
    cin >> numSeat;

    cout << " Initial: " << endl;
    cin >> initial;
```

This method is used to book a seat for a customer, and we set the number of seats and the customer's initial as a string. Following that, we construct a label with the program requesting the customer to select a seat number and initial in order to book the seat.

```
    // if the chosen no of seat is not available,
    // less than 1 and more than 50 it will return to the label

    if (numSeat<1||numSeat>50)
    {
        cout<<" The seats exceeded the availability :D, please choose other seat: ";
        goto label;
    }

    //for checking the status of the seats
```

In the next line, we created an exception handling if the chosen seat number is not available, in this case less than 1 and more than 50, it will return to the label and ask the customer to enter the valid seat number. Then we initialize a temporary pointer and create a new node as the head.

```
    //for checking the status of the seats
    Node *temp;
    temp = new Node;
    temp = first;

    // if the search seat number not equal to the current seat number,
    // it will keep traverse through the seat numbers

    while(temp->seatNum!= numSeat)
    {
        temp=temp->link;
    }

    //booked
    if(temp->seatStat == 1){
        cout<<"Selected seats are booked";
    }

    // not booked
    else{
        // change the seat status == 1
        // change the cosName into the initial entered by the user
        temp->seatStat = 1;
        temp->cosName = initial;
    }
```

The temporary node then traverses through the seat numbers to the next node in the while loop until it reaches the chosen seatNumber, and if the chosen seat number is booked or the status is 1, it will not be able to book for the current customer. Otherwise, the temporary seat status will be

changed to 1 or “TAKEN” and the null cosName (customer initial) will be replaced with the initial. This means the seat is booked.

void indoXXI::cancelBook():

```
void indoXXI::cancelBook()
{
    //initialize the seat number and initial to be cancelled
    int n;
    string in;

    label1:
    cout<<"Enter your seat number to cancel seat\n";
    cin>>n;
    cout<<"Enter your initial: ";
    cin>>in;

    // exception handling
    // if seat number is out of range of the available seat, it will ask the user to input the correct seat number
    // between 1-50

    if(n<1||n>50)
    {
        cout<<"Invalid. Enter seat number to cancel (1-50)\n";
        goto label1;
    }
    //initialize new temp node which is the head
    Node *temp;
    temp = new Node;
    temp = first;
```

This method is used to cancel a seat booking and initialize the seat number and initial to be cancelled. In the next line, we created an exception handling if the chosen seat number is not available, in this case less than 1 and more than 50, it will return to the label and ask the customer to enter the valid seat number to be cancelled. Then we initialize a temporary pointer and create a new node as the head.

```
// while the pointer haven't found the search seat,
// it will keep traversing until its found the right node of seat Number
while(temp->seatNum!=n){
    temp=temp->link;
}

// if the seat is null or not booked
if(temp->seatStat==0){
    cout<<"seat not booked yet!!\n";
}
else{

    // if the cosName == initial entered ( matched between the two data, previous and the new entry )
    // set the seat status = 0

    if(temp->cosName==in){
        temp->seatStat = 0;
        cout<<"seat cancelled!\n";
    }
    // wrong data / seat number entered , unable to cancel
    else
        cout<<"Seat cannot be cancelled\n";
}
```

Then, the temporary node then traverses through the seat numbers to the next node in the while loop until it reaches the chosen seatNumber that wants to be canceled , and if the chosen seat number is booked or the status is 0 1, it will not be able to cancel the seat for the current customer since the seat isn't even booked in the first place . Otherwise, the temporary seat status will be changed to 0 or “____” and the cosName (customer initial) will be replaced with null. This means the seat is cancelled.

How does the program know which pointer to use when booking a seat ?

if seatNum - targetSeatNum < totalSeat / 2 = reverse traverse

Case 1:

Let's say a first customer wants to book a seat with the seat number of 49, it will traverse backward and since **50-49 < 25** means 1 is way smaller than the value of 25 and it is closer from the head of the node. It will be quicker and efficient for the pointer to traverse the searched seat and the booking to proceed.

if seatNum - targetSeatNum > totalSeat / 2 = forward traverse

Case 2:

Let's say the second customer wants to book the second seat with the seat number 22, it will traverse forward because even though the head has moved to 49 it will still use the same logic. Where we can declare that **49 -22 = 27 > 25** will traverse forward and find the selected seat and keep shifting the value of the pointer and find the selected node of seat number.

4. main.cpp

```
#include<iostream>
#include<stdlib.h>
#include "movie.cpp"
#include "loginregister.cpp"

using namespace std;
int main(){
    //initialize int variable to store the user's choice for navigation
    int choice;

    //creating list for Movie objects
    Movie movielist[5];
    //creating movie objects
    Movie avengers=Movie("Avengers","12:10","12 november 2019");
    Movie interstellar=Movie("Interstellar","13:30","12 november 2019");
    Movie kungfu_panda=Movie("Kung Fu Panda","12:30","12 november 2019");
    Movie pokemon=Movie("Pokemon","14:00","12 november 2019");
    Movie cars=Movie("Cars","13:30","12 november 2019");

    //creating seating arrangements for each movie
    avengers.addList();
    interstellar.addList();
    kungfu_panda.addList();
    pokemon.addList();
    cars.addList();

    //adding movie objects to the list
    movielist[0]=avengers;
    movielist[1]=interstellar;
    movielist[2]=kungfu_panda;
    movielist[3]=pokemon;
    movielist[4]=cars;
```

The main file is where the objects are created and where the main menu is. Before creating the menu, an array will be created to store the movie objects. The movie objects will be instantiated and the addList() function will be called to create the seating arrangements for each movie. Then the movie objects are appended to the movie list.

```

//storing size of movie array
int size=sizeof(movielist)/sizeof(movielist[0]);
//for choosing the movie
int moviechoice;
//to store the wanted movie chosen by the user
Movie movie;
// initiaze new instance to call methods from loginregister.cpp
Account newAcc;

    label3:
    //login main menu
    do {
        cout<<"Account Registration and Login"<<endl;
        cout<<"1. Log in"<<endl;
        cout<<"2. Register "<<endl;
        cout<<"3. Exit "<<endl;
        cout<<"\nEnter your choice :";

        cin>>choice;
        cout<<endl;

        // for login validation
        bool validation = false;

```

We create an instance of Account to call the methods of accLogin() and accRegister that is used for the user register and login. Then we create a bool validation that is first set to false and is later used for login verification.

```

switch(choice){
case 1:
    // check if the login successful or else it will go to the break label
    validation = newAcc.accLogin();
    break;
case 2:
    //register new account
    newAcc.accRegister();
    break;
case 3:
    return 0;
default:
    system("cls");
    cout<<"You've made a mistake , give it a try again\n"<<endl;
    break;
    // if login successful proceed to the booking
} if (validation == true){
    break;
}

} while (choice!=4);

```

In the first main menu, we create a switch case and a do while loop. The user has three options: login, register, and leave the program. In the first case, the program will check if the login data matched with the text file and if the login is successful it will break the current menu and proceed to the next main menu. Otherwise, it will return to the first menu. In case 2, the program

will proceed to the register menu where the user enters the username and password and it will return to the main menu to login with their new registered account.

```
do
{
    //MAIN MENU
    cout<<"Welcome to IndoXXI"<<endl;
    cout<<"1. Book and Cancel a Ticket"<<endl;
    cout<<"2. View schedule"<<endl;
    cout<<"3. Exit"<<endl;
    cin>>choice;

    switch (choice)
    {
        //Booking a ticket
        case 1:
            //setting the movie variable into an object without any attributes
            movie=Movie();
            //displaying the movies
            for(int i=0;i<size;i++){
                cout<<i+1<<" "<<endl;
                cout<<"Movie Name: "<<movielist[i].getTitle()<<endl;
                cout<<"Time: "<<movielist[i].getTime()<<endl;
                cout<<"Date: "<<movielist[i].getDate()<<endl;
            }
            //choosing movie
            cout<<"Enter the number displayed of the movie: "<<endl;
            cin>>moviechoice;
            //setting the movie variable to the movie object chosen by the user
            //subtract 1 because indices start at 0
            if(moviechoice<size){
                movie=movielist[moviechoice-1];
            }
            else{
                cout<<"Please enter a valid number!"<<endl;
                break;
            }
        }
    }
```

The main menu will be printed out and it is split to 4 functions. The first is the booking and cancellation of tickets, the second is to view the schedule of the movie and the third one is to terminate the program. For case 1, the available movies are printed out first and a movie object with no attributes is instantiated using the default constructor. The user will then enter their desired movie and then the program will check whether the movie is in the array. If it is the movie object that was instantiated will be equal to the movie object that the user wanted.

```
    }
    //showing the seating arrangement of the movie
    cout<<endl;
    cout<<"Movie Name: "<<movie.getTitle()<<endl;
    cout<<"Time: "<<movie.getTime()<<endl;
    cout<<"Date: "<<movie.getDate()<<endl;
    movie.displayAll();
    cout<<endl;
    //booking the seat
    cout << "1) Book seat" << endl;
    cout << "2) Cancel Book"<<endl;
    cout << "3) Main Menu"<<endl;
    int subchoice;
    cin>>subchoice;
    switch(subchoice){
        case 1:
            //calling the seatBook function to book seat for the user
            movie.seatBook();
            break;
        case 2:
            movie.cancelBook();
            break;
    }
    break;
    //displaying the schedule of movies
    case 2:
        for(int i=0;i<size;i++){
            cout<<i+1<<" "<<endl;
            cout<<"Movie Name: "<<movielist[i].getTitle()<<endl;
            cout<<"Time: "<<movielist[i].getTime()<<endl;
            cout<<"Date: "<<movielist[i].getDate()<<endl;
        }
        break;
    }
} while (choice!=4);
```

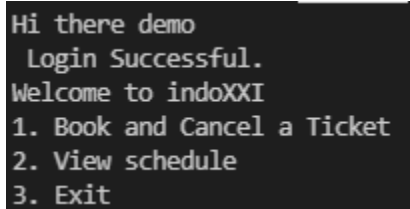
The movie details will be printed and then the “display all” function will be called to show the seating arrangement of the cinema. The user can then choose to cancel or book a ticket. If they choose to book a seat then the book seat function will be called and to cancel the tickets that they have bought, the “cancel book” function will be called. The 2nd case will iterate through the

movie array and print out their details.

Program Manual

1. The user is greeted with a menu that allows them to either login or register an account. They would need to login into an account to be able to access the ticketing system.
2. After logging in, they can enter one into the console to be able to book a ticket or enter two to be able to see the different movies currently playing.
3. After entering one into the console, a list of all the movies currently playing will appear and they will be able to choose a movie they want to book a ticket for.
4. They can enter one again to book a ticket, followed by the seat number of their choice, and then their name to successfully book a seat under their name.
5. They can also choose to cancel their booking by redirecting the same way as mentioned before but instead of entering one into the console, they can enter two, and then enter the initial they inputted when they booked the ticket. The user is also able to book tickets for other movies too.

Results of the Execution



```
Hi there demo
Login Successful.
Welcome to indoXXI
1. Book and Cancel a Ticket
2. View schedule
3. Exit
```

Successfully logging into an account inside the movie ticketing system

```

Movie Name: Avengers
Time: 12:10
Date: 12 november 2019
2.
Movie Name: Interstellar
Time: 13:30
Date: 12 november 2019
3.
Movie Name: Kung Fu Panda
Time: 12:30
Date: 12 november 2019
4.
Movie Name: Pokemon
Time: 14:00
Date: 12 november 2019
5.
Movie Name: Cars
Time: 13:30
Date: 12 november 2019
Welcome to indoXXI
1. Book and Cancel a Ticket
2. View schedule
3. Exit

```

The list of all the movies available after entering two into the menu

```

-----
|          CINEMA SCREEN          |
-----
01 : |  |  | 02 : |  |  | 03 : |  |  | 04 : |  |  | 05 : |  |  |
06 : |  |  | 07 : |  |  | 08 : |  |  | 09 : |  |  | 10 : |  |  |
11 : |  |  | 12 : |  |  | 13 : |  |  | 14 : |  |  | 15 : |  |  |
16 : |  |  | 17 : |  |  | 18 : |  |  | 19 : |  |  | 20 : |  |  |
21 : |  |  | 22 : |  |  | 23 : |  |  | 24 : |  |  | 25 : |  |  |
26 : |  |  | 27 : |  |  | 28 : |  |  | 29 : |  |  | 30 : |  |  |
31 : |  |  | 32 : |  |  | 33 : |  |  | 34 : |  |  | 35 : |  |  |
36 : |  |  | 37 : |  |  | 38 : |  |  | 39 : |  |  | 40 : |  |  |
41 : |  |  | 42 : |  |  | 43 : |  |  | 44 : |  |  | 45 : |  |  |
46 : |  |  | 47 : |  |  | 48 : |  |  | 49 : |  |  | 50 : |  |  |
1) Book seat
2) Cancel Book
3) Main Menu

```

The menu shown after choosing from the list of possible movies

```

Please choose your seat/s:
27
Initial:
demo

Nodes currently in queue:

Selected seats for demo successfully booked!
Your seat number is: 27
Node popped:
demo
27

```

What the terminal looks like when a user is able to successfully book an empty seat

```
-----
|          CINEMA SCREEN          |
|-----|
01 : |  |  | 02 : |  |  | 03 : |  |  | 04 : |  |  | 05 : |  |  |
06 : |  |  | 07 : |  |  | 08 : |  |  | 09 : |  |  | 10 : |  |  |
11 : |  |  | 12 : |  |  | 13 : |  |  | 14 : |  |  | 15 : |  |  |
16 : |  |  | 17 : |  |  | 18 : |  |  | 19 : |  |  | 20 : |  |  |
21 : |  |  | 22 : |  |  | 23 : |  |  | 24 : |  |  | 25 : |  |  |
26 : |  |  | 27 : | TAKEN | 28 : |  |  | 29 : |  |  | 30 : |  |  |
31 : |  |  | 32 : |  |  | 33 : |  |  | 34 : |  |  | 35 : |  |  |
36 : |  |  | 37 : |  |  | 38 : |  |  | 39 : |  |  | 40 : |  |  |
41 : |  |  | 42 : |  |  | 43 : |  |  | 44 : |  |  | 45 : |  |  |
46 : |  |  | 47 : |  |  | 48 : |  |  | 49 : |  |  | 50 : |  |  |
1) Book seat
2) Cancel Book
3) Main Menu
```

What the terminal looks like when there is a seat taken for a movie

```
-----
|          CINEMA SCREEN          |
|-----|
01 : |  |  | 02 : |  |  | 03 : |  |  | 04 : |  |  | 05 : |  |  |
06 : |  |  | 07 : |  |  | 08 : |  |  | 09 : |  |  | 10 : |  |  |
11 : |  |  | 12 : |  |  | 13 : |  |  | 14 : |  |  | 15 : |  |  |
16 : |  |  | 17 : |  |  | 18 : |  |  | 19 : |  |  | 20 : |  |  |
21 : |  |  | 22 : |  |  | 23 : |  |  | 24 : |  |  | 25 : |  |  |
26 : |  |  | 27 : | TAKEN | 28 : |  |  | 29 : |  |  | 30 : |  |  |
31 : |  |  | 32 : |  |  | 33 : |  |  | 34 : |  |  | 35 : |  |  |
36 : |  |  | 37 : |  |  | 38 : |  |  | 39 : |  |  | 40 : |  |  |
41 : |  |  | 42 : |  |  | 43 : |  |  | 44 : |  |  | 45 : |  |  |
46 : |  |  | 47 : |  |  | 48 : |  |  | 49 : |  |  | 50 : |  |  |
1) Book seat
2) Cancel Book
3) Main Menu
2
Enter your seat number to cancel seat
27
Enter your initial: test
Seat cannot be cancelled
```

What the terminal looks like when a user tries to cancel a booking that isn't theirs

```
-----
|          CINEMA SCREEN          |
|-----|
01 : |  |  | 02 : |  |  | 03 : |  |  | 04 : |  |  | 05 : |  |  |
06 : |  |  | 07 : |  |  | 08 : |  |  | 09 : |  |  | 10 : |  |  |
11 : |  |  | 12 : |  |  | 13 : |  |  | 14 : |  |  | 15 : |  |  |
16 : |  |  | 17 : |  |  | 18 : |  |  | 19 : |  |  | 20 : |  |  |
21 : |  |  | 22 : |  |  | 23 : |  |  | 24 : |  |  | 25 : |  |  |
26 : |  |  | 27 : | TAKEN | 28 : |  |  | 29 : |  |  | 30 : |  |  |
31 : |  |  | 32 : |  |  | 33 : |  |  | 34 : |  |  | 35 : |  |  |
36 : |  |  | 37 : |  |  | 38 : |  |  | 39 : |  |  | 40 : |  |  |
41 : |  |  | 42 : |  |  | 43 : |  |  | 44 : |  |  | 45 : |  |  |
46 : |  |  | 47 : |  |  | 48 : |  |  | 49 : |  |  | 50 : |  |  |
1) Book seat
2) Cancel Book
3) Main Menu
2
Enter your seat number to cancel seat
27
Enter your initial: demo
seat cancelled!
```

What the terminal looks like when a user successfully cancels a booked seat.

Demo Video:

<https://www.youtube.com/watch?v=3rTYM4i5eMY>

Github Link:

<https://github.com/ardeliaraminta/Cinema-Ticketing>