|  |  |
| --- | --- |
| Module Code | IY453 |
| Group | C |
| Module Title | Software Design and Development |
| Assessment Type | Implementation Model |
| Module Tutor Name | Jonathan Shore |
| ID Number | T0321003 |
| Date of Submission | 3rd July 2022 |
| Word Count | 9625 |

I confirm that this assignment is my own work.

Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.

**Contents:**

Introduction -------------------------------------------------------------------------------------------------------- 3

Analysis and Design ---------------------------------------------------------------------------------------- 3 - 7

Outline ------------------------------------------------------------------------------------------------ 3 - 4

Inputs, Processes and Outputs ---------------------------------------------------------------- 4 - 5

Design Approach -------------------------------------------------------------------------------------- 5

Flowchart ------------------------------------------------------------------------------------------------ 6

Technical Elements ----------------------------------------------------------------------------------- 7

Testing -------------------------------------------------------------------------------------------------------- 8 - 23

Code Written ---------------------------------------------------------------------------------------------- 23 - 53

Evaluation -------------------------------------------------------------------------------------------------- 53 - 54

References ------------------------------------------------------------------------------------------------ 54 - 55

**Introduction:**

The aim of this coursework is to design a booking system for an independent cinema as well as a system for adding new movies. Several functionalities are outlined and detailed, this includes capturing booking information such as the ticket amounts, movie choice, date time, total price, payment type and card details which are not stored, tickets also cannot be booked if it exceeds the seating capacity of the screen. Specific information for screens, movies and the schedule need to be captured. There needs to be 4 weeks’ worth of movies, with 5 screens. For simplicity’s sake, these screens will display the same movies every day and only change every week.

This report will detail the design and implementation of the code written to tackle these problems and capture the correct information, as closely possible.

**Analysis and Design:**

* **Outline**

Firstly, a menu will be introduced in the beginning that shows the two main functions of the movie booking system, booking a movie and adding a movie. Users will be able to choose between these two options and each will branch out to multiple functions separately. To start, booking a movie will open up with users selecting the specific screen, each will have its own set of movies that change weekly. The screens planned are detailed below:

* IMAX for screen 1, capacity of 200 seats
* IMAX 3D for screen 2, capacity of 150 seats
* Standard for screen 3, capacity of 300 seats
* Standard for screen 4, capacity of 300 seats
* VIP for screen 5, capacity of 100 seats

The movies are set within each screen in weekly format, so screen 1 will have 4 movies available every week. Each of these movies will have set schedules that abide the conditions detailed in the assignment requirements, the cinema will open at 10:00 am and close at 11:30 pm, the first movie of the day will show at 10:15 and 25 minutes of break must occur in between movie showings so the staff will have time to clean and prepare for the next movie. The break between the movies will vary so the time set for each showing will be more appealing, but the break will always be equal or exceed 25 minutes (e.g. if the next showing of a movie after the 25 minute break is 11:33, the time will be changed to 11:40).

Besides that, each movie will have the basic information captured, such as the name, genre, main star, release year and runtime. The information will be captured using classes and objects.

There are also multiple ticket types and prices. To simplify the process, every screen will have the same ticket price. However, these tickets will be purchased based on user, so the 4 ticket types will be adults, children, seniors and students, their prices are £9.50, £5.50, £6.50 and £7.00 respectively. These tickets can be paid with 2 methods, cash or card. Choosing the cash function will check if the tickets purchased surpass the seat capacity, if it does then the user will need to redo the whole booking process, while if it does not, users will be able to proceed with the purchase. Finally, the program will print our relevant information with the purchase of the ticket, like a receipt.

The second function of the program is adding the movies. The manager will be able to input specific information for a movie and it will be printed out so customers can anticipate future releases. This can be done by saving the information within a text file that will be read and printed out and shown in a separate function that the users can access at the start of the menu.

* **Inputs, processes and outputs**

This stage will detail the inputs for the program. To start, there will be 3 inputs for the user to choose. Primarily, booking a movie, adding a movie and viewing upcoming movies. Booking a movie will display screens to choose from, then each screen will have 4 movies that are displayed based on the week. Each movie has information of it’s own that have been explained above. Then they will have to choose a schedule for the movie. Generally, a movie will be shown 4 times a day after some calculations, 3 times if the movie’s runtime is particularly long. After this, the user will be shown the available seating capacities of each screen when purchasing the tickets. This will help with the user calculating the amount of tickets they wish to purchase, if the user has accidentally purchased passed the available seats, they will not be able to proceed and be asked to redo their booking. Successfully purchasing the ticket will give the user 2 options to purchase their ticket as stated earlier. With a final receipt printed if the purchase is successful.

As for the staff or manager side of things, several inputs for the movie information when adding a movie. This information will be stored in a text file with proper formatting. Said movie can be viewed through the upcoming movies function. Booking for these upcoming movies could be done too.

The inputs will be simple to understand by using numbers for choosing options instead of having the user type out specific phrases. Instead of using loops, if the user makes an invalid input, the function can call itself again to act as a loop for invalid inputs. The reasoning behind this is the unpredictability and problems that might arise from using while loops.

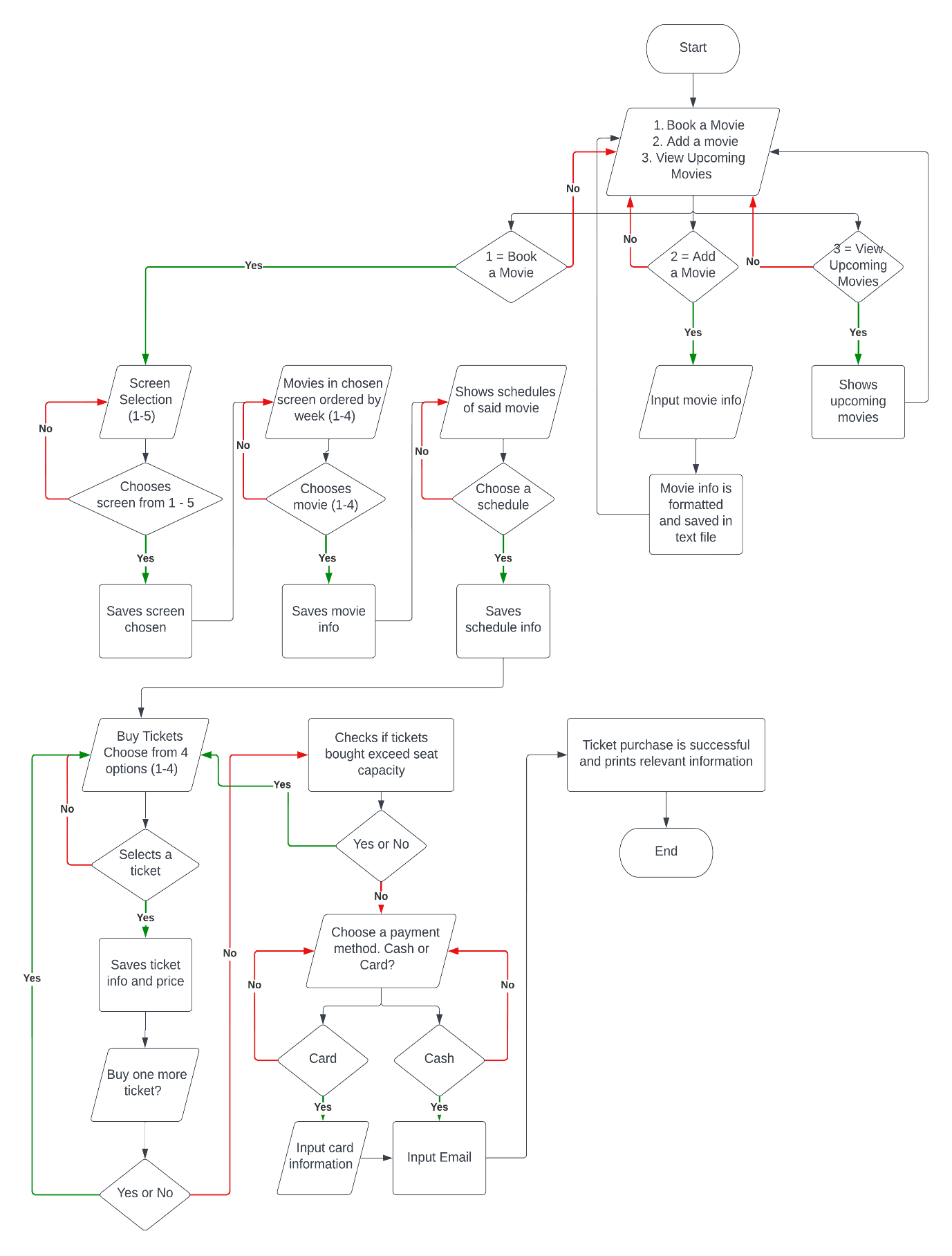
* **Design Approach**

The problem can be approached with 2 paradigms in mind, either procedural or object-oriented. Doing object-oriented program will prove to be more efficient in displaying the myriad of information that needs to be capture, shortening the needed code. Classes allow for much more flexibility in coding and give anyone reading said code easier understanding of how the code functions. There’s also the fact that the code can be modified and changed later on. Things such as encapsulation allows for objects to be self-contained, leaving other parts of the code unaffected, inheritance allows for reusing of classes that may have the same variables, but objects need unique and new variables added. OOP has a learning curve but gives a lot of flexibility and effective problem solving for larger projects, which is perfect for this project (Half, 2021).

Procedural on the other hand makes for easy coding as it is simple and straight forward to understand, it can be proved useful on some simpler aspects in the code. Besides that, larger projects will prove difficult to manage because you would have to manually define variables and functions several times, creating messy and hard to read code. As global data is always vulnerable to changes and might make for many unforeseen errors. Code is usually not reusable and real-world objects would be hard to relate to. (De Las Heras, 2022)

So, the final solution is to aim for a mix of both object-oriented programming and procedural to tackle this assignment depending on what is needed.

* **Flowchart**



* **Technical Elements**

With the outline detailed above, the libraries used would be <iostream>, <fstream>, <vector> and <string> libraries. <fstream> and <vector> libraries allow for file handling and use of vectors in the code respectively. Besides this, several classes can be made with encapsulation to store and get needed data, constructors will be written as well as get set methods will be used to get said data. Classes will be made for things such as the movies, schedule, tickets and screens, depending on how the code goes, this may be changed. Text files can be used to write up movie information and store whatever info is needed, most notably when adding new movies, it can also be used for writing up the 4 movies needed in 5 screens across 4 weeks, this will be read out with an <fstream> library function. Global variables can be used to store whichever screen or movie the user chooses, vectors will be used to store relevant information and iterated through with for loops, functions will be used to separate certain sections of the program, file handling can be used to append and store important information.

That will cover most of the object-oriented side of programming, basic functions, inputs and outputs can be used for the much simpler side of coding, making it more procedural than object-oriented. Specifics cannot be determined at this moment, but without proper understanding on object-oriented programming, making the whole program OOP will prove difficult, so certain aspects will be tweaked and adjusted to reach the requirements of the assignment.

**Testing:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Item to Test | Test Description | Expected Result | Actual Result | Comments/Actions |
| 1 | cin(“1. Book a Movie  2. Add a Movie  3. See Upcoming Movies  Please choose a number: “) | Typical valid data (1 - 3) | Value accepted | Value accepted – program continues as normal | No actions required |
| 2 | cin(“1. Book a Movie  2. Add a Movie  3. See Upcoming Movies  Please choose a number: “) | Invalid data  (4) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 3 | cin(“1. Book a Movie  2. Add a Movie  3. See Upcoming Movies  Please choose a number: “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 4 | cin(“1. Book a Movie  2. Add a Movie  3. See Upcoming Movies  Please choose a number: “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 5 | cin(“Please choose a number: “) | Typical valid data (1 – 5) | Value accepted | Value accepted – program continues as normal | No actions required |
| 6 | cin(“Please choose a number: “) | Invalid data  (6) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 7 | cin(“Please choose a number: “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 8 | cin(“Please choose a number: “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 9 | cin(“Pick a movie by number (1 - 4) / Return to screens (5): “) | Typical valid data (1 – 5) | Value accepted | Value accepted – program continues as normal | No actions required |
| 10 | cin(“Pick a movie by number (1 - 4) / Return to screens (5): “) | Invalid data  (6) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 11 | cin(“Pick a movie by number (1 - 4) / Return to screens (5): “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 12 | cin(“Pick a movie by number (1 - 4) / Return to screens (5): “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 13 | cin(“Pick a Time (1 - 4): “) | Typical valid data (1 – 4) | Value accepted | Value accepted – program continues as normal | No actions required |
| 14 | cin(“Pick a Time (1 - 4): “) | Invalid data  (5) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 15 | cin(“Pick a Time (1 - 4): “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 16 | cin(“Pick a Time (1 - 4): “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 17 | cin(“Pick your ticket (1 - 4): “) | Typical valid data (1 – 4) | Value accepted | Value accepted – program continues as normal | No actions required |
| 18 | cin(“Pick your ticket (1 - 4): “) | Invalid data  (5) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 19 | cin(“Pick your ticket (1 - 4): “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 20 | cin(“Pick your ticket (1 - 4): “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 21 | cin(“Buy more tickets?  1. Yes  2. No  Input: “) | Typical valid data ( 1 – 2) | Value accepted | Value accepted – program continues as normal | No actions required |
| 22 | cin(“Buy more tickets?  1. Yes  2. No  Input: “) | Invalid data  (3) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 23 | cin(“Buy more tickets?  1. Yes  2. No  Input: “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 24 | cin(“Buy more tickets?  1. Yes  2. No  Input: “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 25 | cin(“Choose your payment method:  1. Card  2. Cash  Pick a number (1 - 2): “) | Typical valid data ( 1– 2) | Value accepted | Value accepted – program continues as normal | No actions required |
| 26 | cin(“Choose your payment method:  1. Card  2. Cash  Pick a number (1 - 2): “) | Invalid data  (3) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 27 | cin(“Choose your payment method:  1. Card  2. Cash  Pick a number (1 - 2): “) | Erroneous data (“G”) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 28 | cin(“Choose your payment method:  1. Card  2. Cash  Pick a number (1 - 2): “) | Invalid data  (-1) | Error message: re-enter value | Value not accepted - program asks the user to type in another value | No actions required |
| 25 | getline(“Credit Card Information  Name: “) | Typical valid data (Rick Richard Harith) | Value accepted | Value accepted – program continues as normal | No actions required |
| 26 | getline(“Credit Card Information  Name: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 27 | getline(“Credit Card Information  Name: “) | Valid extreme data  (100000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 28 | getline(“Credit Card Information  Name: “) | Valid data  (“G”) | Value accepted | Value accepted – program continues as normal | No actions required |
| 29 | getline(“Credit Card Information  Name: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 30 | getline(“Credit Card Information  Name: “) | Valid extreme data  (-100000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 31 | cin(“Card Number: “) | Typical valid data (3461498731304051) | Value accepted | Value accepted – program continues as normal | No actions required |
| 32 | cin(“Card Number: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 33 | cin(“Card Number: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 34 | cin(“Card Number: “) | Valid data  (“G”) | Value accepted | Value accepted – program continues as normal | No actions required |
| 35 | cin(“Card Number: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 36 | cin(“Card Number: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 37 | cin(“CVV: “) | Typical valid data (207) | Value accepted | Value accepted – program continues as normal | No actions required |
| 38 | cin(“CVV: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 39 | cin(“CVV: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 40 | cin(“CVV: “) | Valid data  (“G”) | Value accepted | Value accepted – program continues as normal | No actions required |
| 41 | cin(“CVV: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 42 | cin(“CVV: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 43 | cin(“Expiry Date: “) | Valid data  (4/27) | Value accepted | Value accepted – program continues as normal | No actions required |
| 44 | cin(“Expiry Date: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 45 | cin(“Expiry Date: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 46 | cin(“Expiry Date: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 46 | cin(“Expiry Date: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 47 | cin(“Expiry Date: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 48 | cin(“Email: “) | Valid data  (rickharith@gmail.com) | Value accepted | Value accepted – program continues as normal | No actions required |
| 49 | cin(“Email: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 50 | cin(“Email: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 51 | cin(“Email: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 52 | cin(“Email: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 53 | cin(“Email: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 54 | getline(“Type in the movie name: “) | Valid data  (The Batman) | Value accepted | Value accepted – program continues as normal | No actions required |
| 55 | getline(“Type in the movie name: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 56 | getline(“Type in the movie name: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 57 | getline(“Type in the movie name: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 58 | getline(“Type in the movie name: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 59 | getline(“Type in the movie name: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 60 | cin(“Type in the release year: “) | Valid data  (2022) | Value accepted | Value accepted – program continues as normal | No actions required |
| 61 | cin(“Type in the release year: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 62 | cin(“Type in the release year: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 63 | cin(“Type in the release year: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 64 | cin(“Type in the release year: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 65 | cin(“Type in the release year: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 66 | cin(“Type in the genre: “) | Valid data  (Action) | Value accepted | Value accepted – program continues as normal | No actions required |
| 67 | cin(“Type in the genre: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 68 | cin(“Type in the genre: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 69 | cin(“Type in the genre: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 70 | cin(“Type in the genre: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 71 | cin(“Type in the genre: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 72 | getline(“Type in the main star: “) | Valid data  (Robert Pattinson) | Value accepted | Value accepted – program continues as normal | No actions required |
| 73 | getline(“Type in the main star: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 74 | getline(“Type in the main star: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 75 | getline(“Type in the main star: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 76 | getline(“Type in the main star: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 77 | getline(“Type in the main star: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 78 | cin(“Type in the runtime in minutes: “) | Valid data  (125) | Value accepted | Value accepted – program continues as normal | No actions required |
| 79 | cin(“Type in the runtime in minutes: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 80 | cin(“Type in the runtime in minutes: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 81 | cin(“Type in the runtime in minutes: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 82 | cin(“Type in the runtime in minutes: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 83 | cin(“Type in the runtime in minutes: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 84 | cin(“Type in the screen type: “) | Valid data  (IMAX) | 13 1222Q2IO94 | Value accepted – program continues as normal | No actions required |
| 85 | cin(“Type in the screen type: “) | Valid data  (1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 80 | cin(“Type in the screen type: “) | Valid extreme data  (100000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 81 | cin(“Type in the screen type: “) | Valid data  (“G”) | Value accepted | Value accepted – program7continues as normal | No actions required |
| 82 | cin(“Type in the screen type: “) | Valid data  (-1) | Value accepted | Value accepted – program continues as normal | No actions required |
| 83 | cin(“Type in the screen type: “) | Valid extreme data  (-1000000000) | Value accepted | Value accepted – program continues as normal | No actions required |
| 84 | .open(“Screen-1.txt) | Text file is opened | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 85 | .getline(“Screen-1.txt) | Text file has contents extracted successfully | Function executes successfully | Contents are extracted –  Program prints contents and continues as normal | No actions required |
| 86 | .close(“Screen-1.txt) | Text file is closed | Function executes successfully | File is closed –  Program prints contents and continues as normal | No actions required |

Text

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 84 | File is renamed (“Screen-1.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 85 | File is deleted (“Screen-1.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 86 | File is moved (“Screen-1.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 87 | .open(“Screen-2.txt) | Text file is opened | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 88 | .getline(“Screen-2.txt) | Text file has contents extracted successfully | Function executes successfully | Contents are extracted –  Program prints contents and continues as normal | No actions required |
| 89 | .close(“Screen-2.txt) | Text file is closed | Function executes successfully | File is closed –  Program prints contents and continues as normal | No actions required |

Text

Description automatically generated with low confidence

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 90 | File is renamed (“Screen-2.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 91 | File is deleted (“Screen-2.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 92 | File is moved (“Screen-2.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 93 | .open(“Screen-3.txt) | Text file is opened | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 94 | .getline(“Screen-3.txt) | Text file has contents extracted successfully | Function executes successfully | Contents are extracted –  Program prints contents and continues as normal | No actions required |
| 95 | .close(“Screen-3.txt) | Text file is closed | Function executes successfully | File is closed –  Program prints contents and continues as normal | No actions required |

Text

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 96 | File is renamed (“Screen-3.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 97 | File is deleted (“Screen-3.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 98 | File is moved (“Screen-3.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 99 | .open(“Screen-4.txt) | Text file is opened | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 100 | .getline(“Screen-4.txt) | Text file has contents extracted successfully | Function executes successfully | Contents are extracted –  Program prints contents and continues as normal | No actions required |
| 101 | .close(“Screen-4.txt) | Text file is closed | Function executes successfully | File is closed –  Program prints contents and continues as normal | No actions required |

Text

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 102 | File is renamed (“Screen-4.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 103 | File is deleted (“Screen-4.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 104 | File is moved (“Screen-4.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 105 | .open(“Screen-5.txt) | Text file is opened | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 106 | .getline(“Screen-5.txt) | Text file has contents extracted successfully | Function executes successfully | Contents are extracted –  Program prints contents and continues as normal | No actions required |
| 107 | .close(“Screen-5.txt) | Text file is closed | Function executes successfully | File is closed –  Program prints contents and continues as normal | No actions required |

Text

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 108 | File is renamed (“Screen-5.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 109 | File is deleted (“Screen-5.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 110 | File is moved (“Screen-5.txt) | Text file is not opened | Function fails to execute | File cannot be found –  Program exits | No actions required |
| 111 | .open  (“UpcomingMovie.txt) | Text file is opened | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 112 | .getline  (“UpcomingMovie.txt) | Text file has contents extracted successfully | Function executes successfully | Contents are extracted –  Program prints contents and continues as normal | No actions required |
| 113 | .close  (“UpcomingMovie.txt) | Text file is closed | Function executes successfully | File is closed –  Program prints contents and continues as normal | No actions required |
| 114 | .write  (“UpcomingMovie.txt) | Text file can be written into | Function executes successfully | File is written into –  Program prints contents and continues as normal | No actions required |
| 115 | File is renamed (“UpcomingMovie.txt) | New text file is created | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 116 | File is deleted (“UpcomingMovie.txt) | New text file is created | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |
| 117 | File is moved (“UpcomingMovie.txt) | New text file is created | Function executes successfully | File is opened –  Program prints contents and continues as normal | No actions required |

**Code Written:**

#include <iostream>

#include <fstream> //Allows for file handling

#include <string>

#include <vector>

using namespace std;

string FinalMovie, FinalScreen; //Globally declared strings to determine the movie the user chooses, as well as the screen they chose

int SeatCapacity;

class Movies { //Class for Movies, below are the private variables

string Name;

string Genre;

string MainStar;

int Year;

int Runtime;

public:

Movies(string name, string genre, string mainstar, int year, int runtime) { //Construstor

Name = name;

Genre = genre;

MainStar = mainstar;

Year = year;

Runtime = runtime;

}

void setName(string name) { Name = name; } //Get method

string getName() { return Name; } //Set method

void setGenre(string genre) { Genre = genre; }

string getGenre() { return Genre; }

void setMainStar(string mainstar) { MainStar = mainstar; }

string getMainStar() { return MainStar; }

void setYear(int year) { Year = year; }

int getYear() { return Year; }

void getRuntime(int runtime) { Runtime = runtime; }

int getRuntime() { return Runtime; }

};

class Schedule { //Schedule Class, this parent class is for movies that have only 3 SHOWINGS per day

protected:

string Show1;

string Show2;

string Show3;

public:

Schedule(string s1, string s2, string s3) { //Constructor

Show1 = s1;

Show2 = s2;

Show3 = s3;

}

Schedule() { //No argument constructor for inheritance

Show1 = "";

Show2 = "";

Show3 = "";

}

void setShow1(string s1) { Show1 = s1; } //Get method

string getShow1() { return Show1; } //Set method

void setShow2(string s2) { Show2 = s2; }

string getShow2() { return Show2; }

void setShow3(string s3) { Show3 = s3; }

string getShow3() { return Show3; }

};

class Schedule2 : public Schedule { //Inheritance used, child class

string Show4; //One new variable is added, this class is used for movies with 4 SHOWINGS per day

public:

Schedule2(string s1, string s2, string s3, string s4) { //Constuctor

Show1 = s1;

Show2 = s2;

Show3 = s3;

Show4 = s4;

}

Schedule2() { //No argument constructor

Show1 = "";

Show2 = "";

Show3 = "";

Show4 = "";

}

void setShow4(string s4) { Show4 = s4; } //Get method

string getShow4() { return Show4; } //Set method

};

vector<string> TicketTypes; //Vector for storing all the ticket types the user chooses, e.g. Adult ticket, senior ticket etc

vector<float> TotalPrices; //Vector for storing the prices of all the tickets chosen by the user, it is then added up and place in the PriceSum variable

vector<int> TicketAmount; //Vector for amount of tickets bought by user, so it can subtraced from the seat capacity, the sum is then placed into the TicketSum variable

float PriceSum = 0; //Empty global variable to store the final price sum of all tickets bought

int TicketSum = 0; //Empty global variable to store the final amount of tickets bought, used to be subtracted from seat capacity

void BuyingTickets(); //Function prototypes

string ScreenChoice();

void FinalPrint() { //Function that prints the information of the purchase, such as what tickets were bought, total price, which movie and screen

string Email;

for (auto& n : TicketAmount) //This sums up the amount of tickets the user bought

TicketSum += n;

int NewSeatCap = SeatCapacity - TicketSum; //NewSeatCap is the new variable created after the user has bought tickets

if (NewSeatCap <= 0) { //If the tickets bought exceed the available seats, the program clears all values and goes back to previous function

cout << "\nError: We're out of seats for this screen, please redo your booking.\n"<<endl;

TicketTypes.clear(); //Clears vectors

TotalPrices.clear();

TicketAmount.clear();

PriceSum = 0; //Resets the values in variables

TicketSum = 0;

ScreenChoice();

}

else { //If it does not exceed, program continues here

cout << "\nTickets will be sent to the customer's email\nEmail: ";

cin >> Email;

cout << "\n[-----Seats remaining: " << NewSeatCap << "-----]" << endl;

cout << "\nMovie Chosen: " << FinalMovie;

cout << " " << "(" << FinalScreen << ")" << endl;

for (int i = 0; i < TicketTypes.size(); i++) { //Iterates through the vector and prints the stored selected tickets

cout << "\nSelected Ticket: " << TicketTypes[i];

}

cout << "\nTotal price paid: " << PriceSum << " pounds\n";

cout << "\n[-----Payment was successful-----]\n";

}

}

void CardPayment() { //Funtion for card info input

string CardNumber;

string ExpiryDate;

string SecCode;

string CardOwner;

cout << "\nCredit Card Infomation\n";

cout << "Name: ";

getline (cin>> ws, CardOwner); //getline used to ignore spaces since cin

cout << "Card Number: ";

cin >> CardNumber;

cout << "CVV: ";

cin >> SecCode;

cout << "Expiry Date: ";

cin >> ExpiryDate;

FinalPrint();

}

void PaymentMethod() { //Function for choosing payment method

string PaymentChoice;

cout << "\nChoose your payment method:\n1. Card\n2. Cash\nPick a number (1 - 2): ";

cin >> PaymentChoice;

if (PaymentChoice == "1") {

CardPayment();

}

else if (PaymentChoice == "2") {

FinalPrint();

}

else {

cout << "\nWrong input, please try again" << endl;

PaymentMethod(); //If input is invalid, the function is called again to loop back to the beginning of it

}

}

void FinalisePrice() { //Function to get all the chosen information

cout << "Movie Chosen: " << FinalMovie;

cout << " " << "(" << FinalScreen << ")" << endl;

for (int i = 0; i < TicketTypes.size(); i++) { //Iterates through the vector and prints the stored selected tickets

cout << "\nSelected Ticket: " << TicketTypes[i];

}

for (auto& n : TotalPrices) { //Sums up the total price from the tickets bought

PriceSum += n;

}

cout << "\nYour total price is: " << PriceSum << " pounds\n";

PaymentMethod();

}

void BuyMoreTickets() { //Function that asks if the user wishes to buy more tickets;

string MoreTickets;

cout << "\nBuy more tickets?\n1. Yes\n2. No" << endl << "Input: ";

cin >> MoreTickets;

if (MoreTickets == "1") {

BuyingTickets();

}

else if (MoreTickets == "2") {

FinalisePrice();

}

else{

cout << "\nWrong input, please try again\n" << endl;

BuyMoreTickets();

}

}

void BuyingTickets() { //Function for choosing the different types of tickets

string TicketChoices;

cout << "\n[-----Seats left for booking: " << SeatCapacity << "-----]" << endl;

cout << "\n!!!Make sure you do not overbook!!!\n";

cout << "\nBuy Tickets\n" << endl << "Prices:" << endl << "1. Adults\n2. Seniors\n3. Children\n4. Students\n\nPick your ticket (1 - 4): ";

cin >> TicketChoices;

if (TicketChoices == "1") { //Each ticket appends different values to the vectors

TicketAmount.push\_back(1); //Amount of tickets

TicketTypes.push\_back("Adult Ticket"); //Name of ticket

TotalPrices.push\_back(9.50); //Price of ticket

BuyMoreTickets();

}

else if (TicketChoices == "2") {

TicketAmount.push\_back(1);

TicketTypes.push\_back("Senior Ticket");

TotalPrices.push\_back(6.50);

BuyMoreTickets();

}

else if (TicketChoices == "3") {

TicketAmount.push\_back(1);

TicketTypes.push\_back("Children Ticket");

TotalPrices.push\_back(5.50);

BuyMoreTickets();

}

else if (TicketChoices == "4") {

TicketAmount.push\_back(1);

TicketTypes.push\_back("Student Ticket");

TotalPrices.push\_back(7.00);

BuyMoreTickets();

}

else {

cout << "\nWrong input, please try again\n" << endl;

BuyingTickets();

}

}

void SchedulePicker() { //Function for picking the schedule

string ScheduleChoice;

cout << "Pick a Time (1 - 4): ";

cin >> ScheduleChoice;

if (ScheduleChoice == "1" || ScheduleChoice == "2" || ScheduleChoice == "3" || ScheduleChoice == "4") {

BuyingTickets();

}

else {

cout << "\nWrong input, please try again\n" << endl;

SchedulePicker();

}

}

void SchedulePicker2() { //Function for picking schedule

string ScheduleChoice;

cout << "Pick a Time (1 - 3): ";

cin >> ScheduleChoice;

if (ScheduleChoice == "1" || ScheduleChoice == "2" || ScheduleChoice == "3") {

BuyingTickets();

}

else {

cout << "\nWrong input, please try again\n" << endl;

SchedulePicker2();

}

}

string ScreenChoice();//Function prototype

string BookingMovieS1() { //Function for first screen movies

string movieline;

fstream movie;

movie.open("Screen-1.txt"); //Reading the text file and printing it out

if (movie.is\_open()) {

while (!movie.eof()) {

getline(movie, movieline);

cout << movieline << endl;

}

movie.close(); //Closes the text files

string MovieChoice;

cout << endl << "Pick a movie by number (1 - 4) / Return to screens (5): ";

cin >> MovieChoice;

if (MovieChoice == "1") {

FinalMovie = "Doctor Strange in the Multiverse of Madness"; //Variable to store the final movie thats printed at the end of the program

Movies m1("Doctor Strange in the Multiverse of Madness", "Action", "Benedict Cumberbatch", 2022, 126); //Object for setting information with the class

cout << endl <<"Name: " << m1.getName() << endl << "Genre: " << m1.getGenre() << endl << "Main Star: " << m1.getMainStar() << endl << "Release Year: " << m1.getYear() << endl << "Runtime: " << m1.getRuntime()<<"m"<< endl;

// ^ Prints the info with get method from class

Schedule2 ss1("10:15", "13:00", "15:40", "18:30"); //Object for setting the specific schedule function

cout << "\nAvailable Times:" << endl << "1. " << ss1.getShow1() << endl << "2. " << ss1.getShow2() << endl << "3. " << ss1.getShow3() << endl << "4. " << ss1.getShow4() << endl;

// ^ Prints the info with the get method from class

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "2") {

FinalMovie = "The Black Phone";

Movies m2 = Movies("The Black Phone", "Horror", "Ethan Hawke", 2022, 107);

cout << endl << "Name: " << m2.getName() << endl <<"Genre: " << m2.getGenre() << endl << "Main Star: " << m2.getMainStar() << endl <<"Release Year: " << m2.getYear() << endl <<"Runtime: " << m2.getRuntime() << "m" << endl;

Schedule2 ss2("10:15", "12:30", "14:45", "17:15");

cout << "\nAvailable Times:" << endl << "1. " << ss2.getShow1() << endl << "2. " << ss2.getShow2() << endl << "3. " << ss2.getShow3() << endl << "4. " << ss2.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "3") {

FinalMovie = "Elvis";

Movies m3 = Movies("Elvis", "Musical", "Austin Butler", 2022, 160);

cout << endl << "Name: " << m3.getName() << endl << "Genre: " << m3.getGenre() << endl << "Main Star: " << m3.getMainStar() << endl << "Release Year: " << m3.getYear() << endl << "Runtime: " << m3.getRuntime() << "m" << endl;

Schedule ss3("10:15", "13:20", "16:30");

cout << "\nAvailable Times:" << endl << "1. " << ss3.getShow1()<< endl << "2. " << ss3.getShow2() << endl << "3. " << ss3.getShow3() << endl;

SchedulePicker2();

return FinalMovie;

}

else if (MovieChoice == "4") {

FinalMovie = "Jurassic World Dominion";

Movies m4 = Movies("Jurassic World Dominion", "Chris Pratt", "Action", 2022, 150);

cout << endl << "Name: " << m4.getName() << endl << "Genre: " << m4.getGenre() << endl << "Main Star: " << m4.getMainStar() << endl << "Release Year: " << m4.getYear() << endl << "Runtime: " << m4.getRuntime() << "m" << endl;

Schedule2 ss4("10:15", "13:10", "16:10", "19:05");

cout << "\nAvailable Times:" << endl << "1. " << ss4.getShow1() << endl << "2. " << ss4.getShow2() << endl << "3. " << ss4.getShow3() << endl << "4. " << ss4.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "5") {

ScreenChoice(); //Goes back to ScreenChoice function

}

else {

cout << "\nWrong input, please try again\n" << endl; //Invalid input loops the user back to the beginning of the function

BookingMovieS1();

}

}

else {

cout << "Failed to open file.";

}

}

string BookingMovieS2() {

string movieline;

fstream movie;

movie.open("Screen-2.txt");

if (movie.is\_open()) {

while (!movie.eof()) {

getline(movie, movieline);

cout << movieline << endl;

}

movie.close();

string MovieChoice;

cout << endl << "Pick a movie by number (1 - 4) / Return to screens (5): ";

cin >> MovieChoice;

if (MovieChoice == "1") {

FinalMovie = "Star Wars: A New Hope";

Movies m1 = Movies("Star Wars: A New Hope", "Sci-Fi", "Mark Hamil", 1977, 121);

cout << endl << "Name: " << m1.getName() << endl << "Genre: " << m1.getGenre() << endl << "Main Star: " << m1.getMainStar() << endl << "Release Year: " << m1.getYear() << endl << "Runtime: " << m1.getRuntime() << "m" << endl;

Schedule2 ss1("10:15", "12:45", "15:15", "17:45");

cout << "\nAvailable Times:" << endl << "1. " << ss1.getShow1() << endl << "2. " << ss1.getShow2() << endl << "3. " << ss1.getShow3() << endl << "4. " << ss1.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "2") {

FinalMovie = "Star Wars: The Empire Strikes Back";

Movies m2 = Movies("Star Wars: The Empire Strikes Back", "Sci-Fi", "Mark Hamil", 1980, 107);

cout << endl << "Name: " << m2.getName() << endl << "Genre: " << m2.getGenre() << endl << "Main Star: " << m2.getMainStar() << endl << "Release Year: " << m2.getYear() << endl << "Runtime: " << m2.getRuntime() << "m" << endl;

Schedule2 ss2("10:15", "12:45", "15:15", "17:45");

cout << "\nAvailable Times:" << endl << "1. " << ss2.getShow1() << endl << "2. " << ss2.getShow2() << endl << "3. " << ss2.getShow3() << endl << "4. " << ss2.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "3") {

FinalMovie = "Star Wars: The Phantom Menace";

Movies m3 = Movies("Star Wars: The Phantom Menace", "Sci-Fi", "Jake Lloyd", 1991, 136);

cout << endl << "Name: " << m3.getName() << endl << "Genre: " << m3.getGenre() << endl << "Main Star: " << m3.getMainStar() << endl << "Release Year: " << m3.getYear() << endl << "Runtime: " << m3.getRuntime() << "m" << endl;

Schedule2 ss3("10:15", "13:00", "15:45","18:30");

cout << "\nAvailable Times:" << endl << "1. " << ss3.getShow1() << endl << "2. " << ss3.getShow2() << endl << "3. " << ss3.getShow3() << endl << "4. " << ss3.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "4") {

FinalMovie = "Star Wars: Revenge of the Sith";

Movies m4 = Movies("Star Wars: Revenge of the Sith", "Sci-Fi", "Hayden Christensen", 2005, 140);

cout << endl << "Name: " << m4.getName() << endl << "Genre: " << m4.getGenre() << endl << "Main Star: " << m4.getMainStar() << endl << "Release Year: " << m4.getYear() << endl << "Runtime: " << m4.getRuntime() << "m" << endl;

Schedule2 ss4("10:15", "13:00", "15:45", "18:30");

cout << "\nAvailable Times:" << endl << "1. " << ss4.getShow1() << endl << "2. " << ss4.getShow2() << endl << "3. " << ss4.getShow3() << endl << "4. " << ss4.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "5") {

ScreenChoice();

}

else {

cout << "\nWrong input, please try again\n" << endl;

BookingMovieS2();

}

}

else {

cout << "Failed to open file.";

}

}

string BookingMovieS3() {

string movieline;

fstream movie;

movie.open("Screen-3.txt");

if (movie.is\_open()) {

while (!movie.eof()) {

getline(movie, movieline);

cout << movieline << endl;

}

movie.close();

string MovieChoice;

cout << endl << "Pick a movie by number (1 - 4) / Return to screens (5): ";

cin >> MovieChoice;

if (MovieChoice == "1") {

FinalMovie = "Donnie Darko";

Movies m1 = Movies("Donnie Darko", "Fantasy", "Jake Gyllenhaal", 2001, 113);

cout << endl << "Name: " << m1.getName() << endl << "Genre: " << m1.getGenre() << endl << "Main Star: " << m1.getMainStar() << endl << "Release Year: " << m1.getYear() << endl << "Runtime: " << m1.getRuntime() << "m" << endl;

Schedule2 ss1("10:15", "12:45", "15:15", "17:45");

cout << "\nAvailable Times:" << endl << "1. " << ss1.getShow1() << endl << "2. " << ss1.getShow2() << endl << "3. " << ss1.getShow3() << endl << "4. " << ss1.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "2") {

FinalMovie = "Mulholland Drive";

Movies m2 = Movies("Mulholland Drive", "Mystery", "Naomi Watts", 2001, 147);

cout << endl << "Name: " << m2.getName() << endl << "Genre: " << m2.getGenre() << endl << "Main Star: " << m2.getMainStar() << endl << "Release Year: " << m2.getYear() << endl << "Runtime: " << m2.getRuntime() << "m" << endl;

Schedule2 ss2("10:15", "12:45", "15:10", "17:30");

cout << "\nAvailable Times:" << endl << "1. " << ss2.getShow1() << endl << "2. " << ss2.getShow2() << endl << "3. " << ss2.getShow3() << endl << "4. " << ss2.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "3") {

FinalMovie = "Taxi Driver";

Movies m3 = Movies("Taxi Driver", "Drama", "Robert De Niro", 1976, 114);

cout << endl << "Name: " << m3.getName() << endl << "Genre: " << m3.getGenre() << endl << "Main Star: " << m3.getMainStar() << endl << "Release Year: " << m3.getYear() << endl << "Runtime: " << m3.getRuntime() << "m" << endl;

Schedule2 ss3("10:15", "12:40", "15:00", "17:20");

cout << "\nAvailable Times:" << endl << "1. " << ss3.getShow1() << endl << "2. " << ss3.getShow2() << endl << "3. " << ss3.getShow3() << endl << "4. " << ss3.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "4") {

FinalMovie = "Drive";

Movies m4 = Movies("Drive", "Drama", "Ryan Gosling", 2011, 160);

cout << endl << "Name: " << m4.getName() << endl << "Genre: " << m4.getGenre() << endl << "Main Star: " << m4.getMainStar() << endl << "Release Year: " << m4.getYear() << endl << "Runtime: " << m4.getRuntime() << "m" << endl;

Schedule2 ss4("10:15", "12:20", "14:30", "17:35");

cout << "\nAvailable Times:" << endl << "1. " << ss4.getShow1() << endl << "2. " << ss4.getShow2() << endl << "3. " << ss4.getShow3() << endl << "4. " << ss4.getShow4() << endl;

return FinalMovie;

}

else if (MovieChoice == "5") {

ScreenChoice();

}

else {

cout << "\nWrong input, please try again\n" << endl;

BookingMovieS3();

}

}

else {

cout << "Failed to open file.";

}

}

string BookingMovieS4() {

string movieline;

fstream movie;

movie.open("Screen-4.txt");

if (movie.is\_open()) {

while (!movie.eof()) {

getline(movie, movieline);

cout << movieline << endl;

}

movie.close();

string MovieChoice;

cout << endl << "Pick a movie by number (1 - 4) / Return to screens (5): ";

cin >> MovieChoice;

if (MovieChoice == "1") {

FinalMovie = "Fight Club";

Movies m1 = Movies("Fight Club", "Drama", "Brad Pitt", 1999, 139);

cout << endl << "Name: " << m1.getName() << endl << "Genre: " << m1.getGenre() << endl << "Main Star: " << m1.getMainStar() << endl << "Release Year: " << m1.getYear() << endl << "Runtime: " << m1.getRuntime() << "m" << endl;

Schedule2 ss1("10:15", "13:00", "15:45", "18:30");

cout << "\nAvailable Times:" << endl << "1. " << ss1.getShow1() << endl << "2. " << ss1.getShow2() << endl << "3. " << ss1.getShow3() << endl << "4. " << ss1.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "2") {

FinalMovie = "Se7en";

Movies m2 = Movies("Se7en", "Crime", "Brad Pitt", 1995, 127);

cout << endl << "Name: " << m2.getName() << endl << "Genre: " << m2.getGenre() << endl << "Main Star: " << m2.getMainStar() << endl << "Release Year: " << m2.getYear() << endl << "Runtime: " << m2.getRuntime() << "m" << endl;

Schedule2 ss2("10:15", "13:00", "15:45", "18:20");

cout << "\nAvailable Times:" << endl << "1. " << ss2.getShow1() << endl << "2. " << ss2.getShow2() << endl << "3. " << ss2.getShow3() << endl << "4. " << ss2.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "3") {

FinalMovie = "The Silence of the Lambs";

Movies m3 = Movies("The Silence of the Lambs", "Horror", "Anthony Hopkins", 1991, 118);

cout << endl << "Name: " << m3.getName() << endl << "Genre: " << m3.getGenre() << endl << "Main Star: " << m3.getMainStar() << endl << "Release Year: " << m3.getYear() << endl << "Runtime: " << m3.getRuntime() << "m" << endl;

Schedule2 ss3("10:15", "12:40", "15:10", "17:40");

cout << "\nAvailable Times:" << endl << "1. " << ss3.getShow1() << endl << "2. " << ss3.getShow2() << endl << "3. " << ss3.getShow3() << endl << "4. " << ss3.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "4") {

FinalMovie = "Climax";

Movies m4 = Movies("Climax", "Musical", "Sofia Boutella", 2018, 97);

cout << endl << "Name: " << m4.getName() << endl << "Genre: " << m4.getGenre() << endl << "Main Star: " << m4.getMainStar() << endl << "Release Year: " << m4.getYear() << endl << "Runtime: " << m4.getRuntime() << "m" << endl;

Schedule2 ss4("10:15", "12:20", "14:30", "16:40");

cout << "\nAvailable Times:" << endl << "1. " << ss4.getShow1() << endl << "2. " << ss4.getShow2() << endl << "3. " << ss4.getShow3() << endl << "4. " << ss4.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "5") {

ScreenChoice();

}

else {

cout << "\nWrong input, please try again\n" << endl;

BookingMovieS4();

}

}

else {

cout << "Failed to open file.";

}

}

string BookingMovieS5() {

string movieline;

fstream movie;

movie.open("Screen-5.txt");

if (movie.is\_open()) {

while (!movie.eof()) {

getline(movie, movieline);

cout << movieline << endl;

}

movie.close();

string MovieChoice;

cout << endl << "Pick a movie by number (1 - 4) / Return to screens (5): ";

cin >> MovieChoice;

if (MovieChoice == "1") {

FinalMovie = "Oldboy";

Movies m1 = Movies("Oldboy", "Mystery", "Choi Min-sik", 2003, 120);

cout << endl << "Name: " << m1.getName() << endl << "Genre: " << m1.getGenre() << endl << "Main Star: " << m1.getMainStar() << endl << "Release Year: " << m1.getYear() << endl << "Runtime: " << m1.getRuntime() << "m" << endl;

Schedule2 ss1("10:15", "12:40", "15:05", "17:30");

cout << "\nAvailable Times:" << endl << "1. " << ss1.getShow1() << endl << "2. " << ss1.getShow2() << endl << "3. " << ss1.getShow3() << endl << "4. " << ss1.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "2") {

FinalMovie = "Parasite";

Movies m2 = Movies("Parasite", "Thriller", "Choi Woo-shik", 2019, 137);

cout << endl << "Name: " << m2.getName() << endl << "Genre: " << m2.getGenre() << endl << "Main Star: " << m2.getMainStar() << endl << "Release Year: " << m2.getYear() << endl << "Runtime: " << m2.getRuntime() << "m" << endl;

Schedule2 ss2("10:15", "13:00", "15:45", "18:30");

cout << "\nAvailable Times:" << endl << "1. " << ss2.getShow1() << endl << "2. " << ss2.getShow2() << endl << "3. " << ss2.getShow3() << endl << "4. " << ss2.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "3") {

FinalMovie = "Shoplifters";

Movies m3 = Movies("Shoplifters", "Drama", "Sakuya Ando", 2018, 121);

cout << endl << "Name: " << m3.getName() << endl << "Genre: " << m3.getGenre() << endl << "Main Star: " << m3.getMainStar() << endl << "Release Year: " << m3.getYear() << endl << "Runtime: " << m3.getRuntime() << "m" << endl;

Schedule2 ss3("10:15", "12:50", "15:20", "17:45");

cout << "\nAvailable Times:" << endl << "1. " << ss3.getShow1() << endl << "2. " << ss3.getShow2() << endl << "3. " << ss3.getShow3() << endl << "4. " << ss3.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "4") {

FinalMovie = "Kung Fu Hustle";

Movies m4 = Movies("Kung Fu Hustle", "Comedy", "Stephen Chow", 2004, 99);

cout << endl << "Name: " << m4.getName() << endl << "Genre: " << m4.getGenre() << endl << "Main Star: " << m4.getMainStar() << endl << "Release Year: " << m4.getYear() << endl << "Runtime: " << m4.getRuntime() << "m" << endl;

Schedule2 ss4("10:15", "12:30", "14:40", "17:00");

cout << "\nAvailable Times:" << endl << "1. " << ss4.getShow1() << endl << "2. " << ss4.getShow2() << endl << "3. " << ss4.getShow3() << endl << "4. " << ss4.getShow4() << endl;

SchedulePicker();

return FinalMovie;

}

else if (MovieChoice == "5") {

ScreenChoice();

}

else {

cout << "\nWrong input, please try again\n" << endl;

BookingMovieS5();

}

}

else {

cout << "Failed to open file.";

}

}

string ScreenChoice() { //Function for user to choose screens

int Choice1;

cout << "Choose a screen" << endl;

cout << "1. Screen 1 (IMAX)\n2. Screen 2 (IMAX 3D)\n3. Screen 3 (Standard)\n4. Screen 4 (Standard)\n5. Screen 5 (VIP)\nPlease choose a number: ";

cin >> Choice1;

if (Choice1 == 1) {

FinalScreen = "Screen 1 IMAX"; //Specific value of screen

SeatCapacity = 5; //Each screen has it's own capacity

BookingMovieS1();

return FinalScreen; //Returns the value of which screen the user has chosen

}

else if (Choice1 == 2) {

FinalScreen = "Screen 2 IMAX 3D";

SeatCapacity = 82;

BookingMovieS2();

return FinalScreen;

}

else if (Choice1 == 3) {

FinalScreen = "Screen 3 Standard";

SeatCapacity = 24;

BookingMovieS3();

return FinalScreen;

}

else if (Choice1 == 4) {

FinalScreen = "Screen 4 Standard";

SeatCapacity = 56;

BookingMovieS4();

return FinalScreen;

}

else if (Choice1 == 5) {

FinalScreen = "Screen 5 VIP";

SeatCapacity = 100;

BookingMovieS5();

return FinalScreen;

}

else {

cout << "\nWrong input, please try again\n" << endl;

ScreenChoice();

}

}

void StartOptions();

void AddMovies() { //Adding movies to the program

string AName;

string ARelease;

string AGenre;

string AStar;

string ARuntime;

string AScreen;

cout << "Type in the movie name: ";

getline(cin >> ws, AName);

cout << "Type in the release year: ";

cin >> ARelease;

cout << "Type in the genre: ";

cin >> AGenre;

cout << "Type in the main star: ";

getline(cin >> ws, AStar);

cout << "Type in the runtime in minutes: ";

cin >> ARuntime;

cout << "Type in the screen type: ";

cin >> AScreen;

fstream UpcomingMovie;

UpcomingMovie.open("UpcomingMovie.txt", ios::app); //Opens upcoming movie text file

if (UpcomingMovie.is\_open()) {

UpcomingMovie << AName << " (" << ARelease << ")" << ", " << AGenre << ", " << AStar << ", " << ARuntime << "m, " << "[" << AScreen << "]."<<endl; //<--Adds movies in file, formatted like this

UpcomingMovie.close();

cout << "\n[------Movie has been added successfully-----]\n";

StartOptions();

}

else {

cout << "File cannot be opened.";

}

}

void SeeUpcomingMovie() { //Function to see upcoming movies or newly added movies

cout << "\n[-----Upcoming Movies-----]\n";

string MovieLine;

fstream UMovie;

UMovie.open("UpcomingMovie.txt");

if (UMovie.is\_open()) {

while (!UMovie.eof()) { //While its not end of file, the code below will be executed

getline(UMovie, MovieLine);

cout << MovieLine << endl;

}

UMovie.close();

StartOptions();

}

else {

cout << "File cannot be opened.";

}

}

void StartOptions() { //Beginning function to let user choose the first few options

string Choice1;

cout << "Welcome to the Movie Booking System"<<endl;

cout << "1. Book a Movie\n2. Add a Movie\n3. See Upcoming Movies\nPlease choose a number: ";

cin >> Choice1;

if (Choice1 == "1") {

ScreenChoice();

}

else if (Choice1 == "2") {

AddMovies();

}

else if (Choice1 == "3") {

SeeUpcomingMovie();

}

else {

cout << "\nWrong input, please try again\n" << endl;

StartOptions();

}

}

int main() {

StartOptions();

}

**Evaluation:**

The three strengths of the program are the simplicity in the inputs, inputs are done with typing out numbers to select the options present in each menu. This allows for the user to make little mistakes and shortens the code needed to be written as there are only several inputs allowed, instead of doing while loops, the original functions are called when the wrong input is entered to loop it back. File handling was also used efficiently to print large amounts of information such as the movie information, so it does not clutter up the code too much, it is also used to store new movies and act as a save file of sorts, this is done with the <fstream> append function, vectors were also used to store certain information and iterated through with for loops. Finally, classes with encapsulation were used to store information and not disrupt the original variables which are private. Constructors and get set methods were used as well as no argument constructors to do inheritance, the inheritance was used for setting up the movie schedules, while most of the movies had 4 showings per day, one movie had 3. Instead of creating a new class to accommodate this movie, inheritance was used to make a parent class with 3 showings and a child class with 4 showings. Besides that, the classes used generally helped in making the code more efficient.

Despite the positives, the program has many negatives. The main negative is that the classes made were not used to their full potential, they could have been used far more than printing out information as I had difficulty finding a way to store objects in vectors, my lack of competence and knowledge in object-oriented programming cause this issue, overloading and pointers would have helped resolve some issues I was facing. The second weakness is the fact that the schedules are purely printed information and are not actually selected, the reason behind this was because of the abundance of movies present, with each needing unique schedules that abide the rules set by the assignment, many if statements would have to be used to store the user selected schedule. However, this issue would also be resolved by having more knowledge in how object-oriented programming works, if more classes and objects were used efficiently, the code written would have been much more efficient than writing if statements. Finally, the adding a movie function was a simple open, read and write into a file, the new movies were not tied to the schedule as the schedule is pre coded instead of being flexible, it is also not possible to book these movies, which is the programs largest problem.

There are areas of improvement such as utilising classes and do more object-oriented programming, and get the benefits of said way of programming, most of the code is procedural and lacks flexibility and reusability. The newly added movies should also be able to be booked by the user and tie itself to a proper schedule, as well as making up a newly added week. At the initial stages of writing the code, I made several classes that are not utilised in the final code, as I was not able to properly use them, with more time, I could have been able to do much more with those classes and make my code much more efficient and complete.

If I was able to start over with the assignment with my current knowledge, or was given much more time to do said work, I would resolve most of the issues such as having globally accessible and uniquely named objects for the 20 movies to print the final chose movie information, as well as doing so for the schedules and screens followed with their own respective classes, forming a receipt of sorts at the end of the booking purchase, making it a much more complete assignment.

**References:**

* De Las Heras, J. (2022). *Object Oriented vs Procedural Programming paradigms*. [online] Computingatschool.org.uk. Retrieved from: https://www.computingatschool.org.uk/news-and-blogs/2022/may/object-oriented-vs-procedural-programming-paradigms [Accessed 1 July 2022].
* freeCodeCamp.org (2021). *Object Oriented Programming (OOP) in C++ Course*. *YouTube*. Retrieved from: https://www.youtube.com/watch?v=wN0x9eZLix4&ab\_channel=freeCodeCamp.org [Accessed 1 July 2022].
* Half, R. (2021). *4 Advantages of Object-Oriented Programming*. [online] Roberthalf.com. Retrieved from: https://www.roberthalf.com/blog/salaries-and-skills/4-advantages-of-object-oriented-programming [Accessed 1 July 2022].
* Learn C (2022). *Learn C++ – Skill up with our free tutorials*. [online] Learncpp.com. Retrieved from: https://www.learncpp.com/ [Accessed 1 July 2022].
* ‌ W3schools.com. (2022). *C++ Tutorial*. [online] Retrieved from: https://www.w3schools.com/CPP/default.asp [Accessed 1 July 2022].

‌

‌