CS 2337 PROJECT 4 – Toy Story 25th Anniversary Ticket Reservation System

Pseudocode Due: 11/4 by 11:59 PM (No late submission)

Final Code Due: 11/20 by 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- All project source code will be submitted in zyLabs.
  - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Each submitted program will be graded with the rubric provided in eLearning as well as a set of test cases. These test cases will be posted in eLearning after the due date.
  - zyLabs will provide you with an opportunity to see how well your project works against the test cases. Although you cannot see the actual test cases, a description will be provided for each test case that should help you understand where your program fails.
- Programs must compile using gcc 7.3.0 or higher with the following flags enabled
  - -Wall
  - -Wextra
  - -Wuninitialized
  - -pedantic-errors
  - -Wconversion
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objectives:

- Use a hashmap (hash table) to access and store various pieces of data.
- Create a comprehensive professional application.

Problem: This year marks the 25th anniversary of Toy Story. The owner of a small theater in an isolated town in Montana has hired you to develop the backend for an online ticket reservation system. Your program will allow attendees to select seats and display the current seating arrangement. At the end of the program, a report will be generated for the owner indicating how many seats were sold/unsold and how much money was earned.

The theater owner also wants to provide expanded customer service by building a reservation system to track customer orders. The customers need the ability to login to the system and access their orders. The customers should have the ability to add orders, change orders or cancel orders as well as the ability to see the receipts for their orders. The reservation system should also have an administrator interface.

Pseudocode: Your pseudocode should describe the following items

- Main.cpp
  - List functions you plan to create
    - Determine the parameters
    - Determine the return type
    - Detail the step-by-step logic that the function will perform

    - Detail the step-by-step logic of the main function
- Describe how you intend to store orders in the hash map

Details

- The user account information will be stored in a HashMap (-15 points if not)
    - You may use the pre-built unordered map class or modify the hashmap you built for review homework 2.
    - If using your own hashmap class, name the filse `Hashmap.h` and `Hashmap.cpp`
    - If using the pre-built unordered map class, you must submit blank files in Zybooks for `Hashmap.h` and `Hashmap.cpp`
- Each entry in the HashMap should contain the customer username, password and all orders.
    - An order consists of the selected seats and number of tickets per ticket type
    - I suggest using the username as the key value
- Three auditoriums will be available to reserve seats
- The seating arrangement for each auditorium will be stored in individual files
    - Auditorium 1 – A1.txt
    - Auditorium 2 – A2.txt
    - Auditorium 3 – A3.txt
- Each line in the file will represent a row in the auditorium. The number of rows in the auditorium is unknown to you.
    - Do not assume that each auditorium has the same number of rows
- The number of seats in each row of the auditorium will be the same. For example, if the first line of the file has 15 seats, then every subsequent row in the theater will also have 15 seats.
    - Do not assume each auditorium has the same number of seats per row
- No auditorium will have more than 26 seats in a row.
- Empty seats are represented by a period (.).
- Reserved seats are represented by a letter (A, C or S) **in the file**
    - This will be used to create reports
    - A =adult
    - C = child
    - S = senior
- Reserved seats will be represented by a hashtag (#) **on the screen**
    - The user does not need to know what type of ticket was sold, just that a seat is reserved.
- You may use the reservation system from project 0 or 2.
    - Feel free to tweak the reservation system as you feel is necessary.
    - You must submit the following files in ZyBooks
        - `Auditorium.h and Auditorium.cpp`
        - `Node.h`
        - `Seat.h and Seat.cpp`
        - If you used the 2D array approach, these files will be empty
- Tickets can only be reserved the day of the screening and all screenings are at the same time.
- Ticket prices are as follows:
    - Adult - $10
    - Child - $5

    - Senior - $7.50
- When the program starts, no user will have any orders, although some seats may already be reserved in the auditorium.
    - There is not enough time remaining in the semester to build a fully functioning system.

Filling the HashMap:

- All user data will be in a file named `userdb.dat`.
- The program should read the file and fill in the hashmap before starting the user interface.
- The format of the file is as follows:
  `<username><space><password><newline>`
    - This should be encrypted, but you have enough to do without adding in decryption
- The last line in the file may not contain a newline character at the end of the line

Customer User Interface and Workflow: Present a user-friendly system for the user to reserve seats.

- **Starting Point:** Before the customer can use the system, a login is required. The starting point of the interface is to ask the user for a username. After the username is entered, prompt for the password. Verify that the password is correct. If the password is not valid, display `Invalid password` and prompt the user to enter the password again. If the password is entered incorrectly a total of 3 times, return to the starting point. Once the user has successfully logged in, display the main menu.

- **Main Menu**

```
1. Reserve Seats
2. View Orders
3. Update Order
4. Display Receipt
5. Log Out
```

- **Reserve Seats:** When the user reserves seats, display the auditorium submenu.

```
1. Auditorium 1
2. Auditorium 2
3. Auditorium 3
```

After the auditorium is selected, display that auditorium.

```
  ABCDEFGHIJKLMNOPQRST
1 ...##..#####........
2 ########....####..##
3 .........##.........
4 #.#.#.#.#.#.#.#.#.#.
5 ########.#####.#####
```

The rows are numbered and the seats are identified in each row by a letter of the alphabet

After the seating chart has been displayed, prompt the user for the following information in the order below:

- Row number
- Starting seat letter
- Number of adult tickets
- Number of child tickets
- Number of senior tickets

Assume that the user wants to reserve sequential seats to the right of the first seat entered. Adult tickets will be reserved first, followed by child and then senior. All seats must be open for a reservation to be processed.

If the desired seats are not available, offer the user the best available seats that meet their criteria **in the entire auditorium**. The best available seats are the seats closest to the middle of the auditorium.

- Think of the distance between 2 points
- Use the distance between the center of the selection and the center of the auditorium.
- In the event of a tie for distance, the row closest to the middle of the auditorium should be selected.
- In the event of a tie for closest row, use the row with the smaller number

Display the best available seats in the following format:
`<row><starting seat> - <row><ending seat>`
`Example:` `3D – 3F`

State to the user what the best available seats are and then ask if they would like those seats. Prompt the user to enter a **Y** to reserve the best available or **N** to refuse the best available. If the user declines the best available seats, return the user to the main menu. If the user accepts the best available seats, reserve them and display a confirmation to the screen. Once the selection has been processed, return to the main menu**.** If there are no alternate seats available, display **`no seats available`** to the user instead of a prompt and return to the main menu.

When prompting the user for input, expect anything. Do not assume any input will be valid. If you ask for a number, the user could put in a floating point number, symbols or maybe even a sentence or two. Make sure that your program handles proper validation. If invalid input is entered, loop until valid input is received.

- **View Orders:** Display all the orders associated with the user using the format below. The order consists of the auditorium, seats, and number of tickets per ticket type. You do not have to specify which type of person is sitting in each seat. If there are no orders, display `No orders` to the user. Return to the main menu once the orders are displayed. Seats will be displayed in row then seat order.

```
Auditorium <number>,<space><each seat separated by commas><newline>
<number><space>adult,<space><number><space>child,<space><number><space>senior
```

Example:
```
Auditorium 1, 2A,2B
1 adult, 1 child, 0 senior
```

- **Update Order:** Display the orders in a numerical menu system so that the user can enter the number of the order that should be updated. List each order on a separate line and be descriptive of the order. Do not just list Order 1, Order 2, etc.

  After the order menu has been displayed and the user has selected an order, display the menu below and prompt the user for an update action:

  ```
  1. Add tickets to order
  2. Delete tickets from order
  3. Cancel Order
  ```

  If the user wishes to add tickets to the order, step the user through the reservation process. **All seats reserved this way will be added to the current order.** The user may attempt to reserve seats anywhere in the current auditorium. Do not offer the user the best available seats if the selected seats are not available. Reserve the seats if available. If not, display an appropriate message and return to the update action menu (see above).

  If the user wishes to delete tickets from an order, ask the user for the row and seat number to remove from the order. Verify the seat entered by the user is valid for the order. If the seat selection is not valid, display an appropriate message and return to the Update Order submenu. After validating the seat information to remove, unreserve the seat and update the ticket totals for that order accordingly. It is possible for the user to remove all tickets from the order effectively cancelling it.

  If the user wishes to cancel the order, mark all seats in the order as available and remove the order from the user's account.

  Once the order update has been competed, return to the main menu.

- **Display Receipt:** Create a receipt for the user. Display each order in detail (auditorium, seats, and number of tickets per ticket type), the amount of each order and the overall amount of all orders using the format below. Once displayed, return to the main menu.

  ```
  Auditorium <number>,<space><each seat separated by commas>
  <number><space>adult,<space><number><space>child,<space><number><space>senior
  Order Total:<space>$<amount to two decimal places>
  <blank line>

  <repeat for each order>

  Customer Total:<space>$<total amount to two decimal places>
  ```

  Example:

  ```
  Auditorium 1, 2A,2B
  1 adult, 1 child, 0 senior
  Order Total: $15.00
  ```

```
Customer Total: $15.00
```

- **Log Out:** Return to the **Starting Point**

Admin Interface and Workflow:

- **Starting Point:** Like the customer, a login is required for the admin. Both the customer and admin will begin at the login prompt. The admin username is `admin` (*Please note this is a horrible idea professionally, but that is the network admin's problem*). Like all other users, the username and password information for the admin will be in `userdb.dat`. After the username is entered, prompt for the password. Verify that the password is correct. If the password is not valid, prompt the user to enter the password again. If the password is entered incorrectly a total of 3 times, return to the starting point. Once the user has successfully logged in, display the Admin main menu.

- **Admin Main Menu**

```
1. Print Report
2. Logout
3. Exit
```

- **Print Report:** Display a report to the console using the following format for each auditorium:

  For each auditorium, display the following information on 1 line separated by tabs:

```
Auditorium<space><number>
<Open #>
<Reserved #>
<Adult #>
<Child #>
<Senior #>
$<total amount to two decimal places>
```

  Overall Summary

```
Total<tab>
<Open seats in all auditoriums>
<Reserved seats in all auditoriums>
<Adult seats in all auditoriums>
<Child seats in all auditoriums>
<Senior seats in all auditoriums>
$<total amount of all auditoriums to two decimal places>
```

- **Logout:** Return to the **Starting Point**

- **Exit:** Store each auditorium in a new file named A**#**Final.txt, where the **#** represents the auditorium number. The auditorium files should only contain the proper character for each type of seat reserved. After the files have been updated, end the program.