CS3520 Programming in C++

Assignment 2

Objectives:

- Understand how to built-in arrays and vectors
- Working with functions
- Learn how to write to a file and read from a file.
- Work with strings and characters
- Working with pseudocode
- Code organization with multiple files (*.hpp, *.cpp)

Submission:

Create a separate folder (named "hw2" or "assign2") in your github repository and submit all of your source code (*.cpp and *.hpp files), other files (if needed), along with appropriate Makefile(s). Submit the link to your github commit on Canvas. Also, be sure to make the code readable.

Grading scheme: 90 pts

- Pseudocode: 10 pts, Source Code: 72 pts,
- Code Organization: 3 points, Code Readability: 5 pts

Coding Assignments: Airline Reservations System

A small airline has just purchased a computer for its new automated reservations system. The CTO has asked you to program the new system. You will write a program to assign seats on each flight of the airline's only plane (capacity: **72** seats, in 8 rows). Use a 2D array (or vector) for seat assignments. For the sake of simplicity, assume that each row has **9** seats labeled A, B, D, E, F, G, I, J and K. In this configuration, there are 2 seats on one side of the plane, 4 seats in the middle and 3 seats on the other side i.e., Seats A and K are window seats; and Seats C and H are not being used and can be thought of as Aisles.

At the start of the program, your program should give the following **main menu** options to the user:

(1) Display the seating map/chart showing all seats and indicating if each seat is filled and which seats are assigned.
(2) Print a passenger manifest showing names of all passengers and their seat numbers in a tabular format. The default view prints the manifest organized by seat numbers. Prompt the user to *choose one of the 4 available options: Print Manifest using seat numbers (Ascending order or Descending order), Alphabetical format (i.e., using name, A-Z or Z-A)*.
(3) Load Previous "Seat Assignment and Passenger Manifest"
(4) Save Current "Seat Assignment and Passenger Manifest"
(5) Choose a seat (with the help of automated system)
(6) Cancel seat assignment
(7) prints a boarding pass for a chosen seat passenger or seat number indicating the person's name, seat number and whether it's in the first class or business class or economy section of the plane (For example**: JOHN DOE,** SEAT: **B2,** Level: **Business Class**).
User will provide the name for this option and if no passenger is assigned to the selected seat, you should print a message, "Seat Unassigned - No Boarding pass available. Try Again!". If you have multiple travelers with same name, user must provide the seat number as well. Alternatively, user may provide seat number only to print the boarding pass. Allow user to enter the input no

more than 3 times. In case of 3 incorrect choices (for names or seat numbers), display a message, “Boarding pass is not available at this time. Try Again later!” and display the main menu again.

(8) Quit the program

Your program **must** show the above main menu again if user does not select {one of the nine} valid input options. Your program must not quit until “Quit” option (#8) is selected.

If user likes to choose a seat, the program should display the following menu of alternatives:

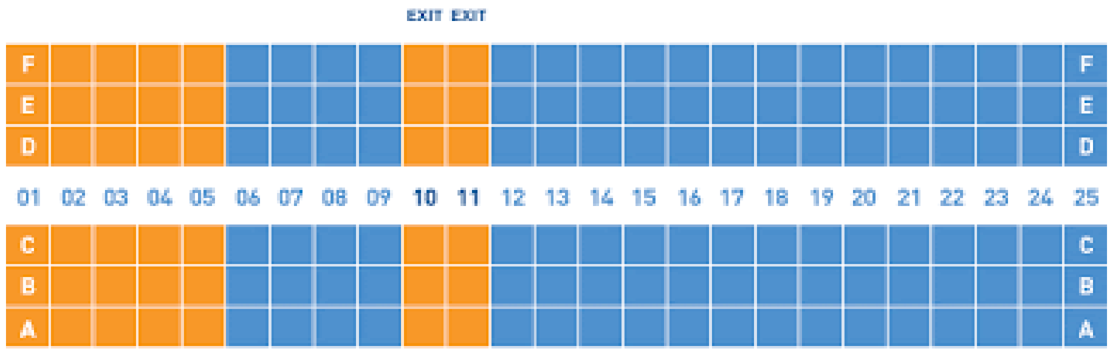
Please type ‘F’ for "first class"
Please type ‘B’ for "business class"
Others will be “economy class”.

If the person types ‘F’, then your program should assign a seat in the first-class section (row 1). If the person types ‘B’, then your program should assign a seat in the business class section (rows 2-3). For others, your program should assign a seat in the economy section (rows 4–8). Your automated system uses **random numbers generator for Seat Assignment** *(i.e., Use a random number generator to select the seat - row number and column - and then check if that seat is currently un-assigned. This should be repeated until no seat is left unassigned).* Let’s call this INITIAL SEAT SELECTION. Your program should, of course, never assign a seat that has already been assigned. When the first-class section is full, your program should ask the person if it’s acceptable to be placed in the business class. Similarly, if the business section is full, ask the user if economy section is acceptable. If user doesn’t like the seat or choices given, he/she can always choose the next flight! You may print the message "Next flight leaves in 3 hours.".

After the INITIAL SEAT SELECTION, User should have the choice of ***upgrading*** to the higher class (i.e., business class or first class) if a seat is available. To keep things simple, user should not be inquired about upgrade options if there is no seat available in the higher class or if no upgrade is possible (i.e., if the seat is already in first class). If user chooses to upgrade, then make the appropriate seat assignment and show the FINAL SEAT ASSIGNMENT. User may also ***downgrade*** from higher class (i.e., first class or business class). Other details of downgrade option will be similar to the details of upgrade option as described above. If user is not able to upgrade or downgrade then their INITIAL SEAT SELECTION becomes FINAL SEAT ASSIGNMENT.

If user selects the upgrade/downgrade option, he/she should be able to see the previous name, seat number and the new one. Also, if a seat has ***now*** become available in higher class, he/she should be inquired about the upgrade. Similarly, if a seat has ***now*** become available in lower class, he/she should be inquired about the downgrade.

Note: Airlines typically use a seating map/chart similar to the one shown below. Design your own version of seating chart (you don’t have to use graphics or colors)



If user does not want to travel any more, they can request to cancel their seat assignment. To cancel, they must provide the seat number and the name. Your system should not cancel a seat until both the seat number and name are matched.

You may need to think of a mechanism to store passenger names. This can be done in several ways including using 1-dimensional or 2-dimensional arrays or vectors. You will also have to think carefully about how to print the manifest in alphabetical order.

Your program should be able to save (and later load) the current seat assignments and passenger manifest from file(s). Use any format that you find appropriate. This approach is commonly known as ***state-persistence.***

***Here is one approach you could use*:** Use a double-subscripted (2D) array to represent the seating chart of the plane. Initialize all the elements of the array to 0 to indicate that all seats are empty. As each seat is assigned, set the corresponding element of the array to 1 to indicate that the seat is no longer available.

Also implement an auto-save capability where your manifest gets saved automatically whenever options 5,6,7 are used (i.e., seat chosen or cancelled). User can manually save the manifest too, as described above.

Write a modular program, a program which uses appropriate functions to complete different tasks. Don't write all the code in main function. Organize your code in files properly (by moving all function declarations in a header file, all function definitions in a cpp file and your main function in a separate cpp file)

To-do:

- **Part1:** Start by writing complete pseudocode or algorithm (describing the steps that your program will need to perform). These must include sufficient details that would allow you to write the code directly. For submission, a txt file or *.cpp file with comments will suffice.
- **Part2**: Implement and fully test the program as described above. While testing, make sure that you have enough passengers added to the flight to check if the users can successfully upgrade, as needed. Also verify that you are successfully able to save and then load seat assignments and passenger manifest. Test all test cases properly.
  Additional Requirements:
  - Your implementation must use an array **AND** a vector *somewhere*.
  - Please avoid using classes and structures.

Hints:

- *To help you visualize the seat arrangements, you can either think of seats labelled in first row as A1, B2, D4, ... K11 and second row labelled as A12, B13, D14 and so on. Another way is to think of seats in row 1 labelled as A1, B1, D1, …, K1 and then second row as A2, B2, D2, and so on. Use whatever helps.*
- *If you design your program in a way that relies on a new random number being generated for choosing each seat, you might observe that your program generates a number that may already be generated before (i.e., in some previous iteration). This could make your program computationally inefficient. Think of a way to improve this. Perhaps use a random number generator that doesn't provide repeat values.*
- *If you design your read/write operations carefully, you will be able to exit your program {and subsequently run it} repeatedly without having to add passengers from scratch again and again.*

- *Be sure to implement auto-save feature i.e., as soon as any changes are made, your program should save the required detail (passenger manifest, seat assignment, etc.) in the file. Similarly, if user decides to cancel travel plans, your file should be updated accordingly.*
- *Only accept valid options from Menu. If user chooses incorrect option, display the menu again with appropriate message!*
- *When you start your program, it may not have any passengers. Alternatively, you may pre-assign a handful of seats to passengers before you start using your program. The choice is yours.*

Change Log:

- September 28th (11pm): Initial Version
- September 29th (11pm): Fixed typo, Seat 2B is Business class" and Option 8 is for Quit.
- October 1st (5pm): Clarified Requirements regarding use of Classes and Structs