SEHH2042 Computer Programming

Group Project – Airplane Seating Assignment

(Due: 18:00 - 23 APR 2021, Friday)

Intended Learning Outcomes

- develop computer programs in one or more high level language programming environment
- design and develop structured and documented computer programs
- explain the fundamentals of object-oriented programming and apply it in computer program development
- integrate the computer programming techniques to solve practical problems

Case Background and System Requirements

In this assignment, you need to develop an Airplane Seating Assignment system. This program is used to assign seats for *a small commercial airplane*. The airplane has *thirteen* rows of seat, with six seats in each row. Below please find the details of rows according to classes.

Row	Class
1 – 2	First
3 – 7	Business
8 – 13	Economy

Your program must accept the user's ticket class type and the desired seat location based on the seating plan as follows:

```
            A    B    C    D    E    F
 1          *    *    *    *    *    *
 2          *    *    *    *    *    *
 3          *    *    *    *    *    *
 4          *    *    *    *    *    *
 5          *    *    *    *    *    *
 6          *    *    *    *    *    *
 7          *    *    *    *    *    *
 8          *    *    *    *    *    *
 9          *    *    *    *    *    *
10          *    *    *    *    *    *
11          *    *    *    *    *    *
12          *    *    *    *    *    *
13          *    *    *    *    *    *
```

* represents an empty seat available to be assigned. When a seat is assigned, an X will appear to replace the * in the location of that seat.

Each group (5-6 students per group) is required to write a Win32 Console Application program called **`JetAssign.cpp`**. The requirements (R) are elaborated as follows.

R0 When the program starts, the console should display a welcome message, followed by the main menu of the system.

```
Welcome message (designed by your group)
*** main menu ***
[1] Add an assignment
[2] Delete an assignment
[3] Add assignments in batch
[4] Show latest seating plan
[5] Show details
[6] Exit
*****************
Option (1-6):
```

This system runs continuously until the operator chooses to quit it. The corresponding requirement of each option shall be further elaborated as follows.

R1 **`[1] Add an assignment`**

When a user demands an assignment, the operator presses [1]. The operator helps enter the **user's name, passport ID and the desired seat location**. If the desired seat location has been assigned, the operator enquires the user for another one until a satisfactory assignment location is found. The user can choose aborting the assignment addition if *no* satisfactory location can be sorted out.

The table below shows the format of input values:

Entry	**Format**	**Example**
User's name	Surname Firstname	Chan Tai Man
Passport ID	A series of letters and digits without space	HK12345678A
Seat location	RowColumn, no space	10D

R2 **`[2] Delete an assignment`**

When a user requests to cancel an assignment, the operator presses [2]. The previously stored **user's name and passport ID** identical with the request user's name and passport ID will be deleted. If no previously stored record can be identified, the operator can either re-enter the details to match again, or abort the deletion entry operation. On successful match, the assigned seat location will be set available for addition assignment. However, before the assignment deletion is committed, the operator must **confirm** the deletion operation.

R3 **`[3] Add assignments in batch`**

When a group of users approaches for seat assignment, the operator presses [3]. The operator enters the users' details using the format as follows, with each user using one line to indicate, until 0 is entered.

Format: **`Name/PassportID/Seat`**

Example:

```
Chan Tai Man/HK12345678A/10D
Pan Peter/US2356AAD11/2E
Chu Mimi/ER273/1A
...
0
```

If the requested seat location of a particular user in the batch has already been assigned, the assignment addition request of that particular user will be *unsuccessful*. Only the successful assignment addition requests update the system. The system displays both a list of successful requests and a list of unsuccessful requests. Before committing the batch assignment of seats, the operator can confirm or abort the action. Upon knowing the unsuccessful cases from the system, the operator handles each of them individually instead.

R4 **`[4] Show latest seating plan`**

When a user wishes to know the latest overall seat assignment, the operator presses [4]. The most updated seating plan displays.

R5 **`[5] Show details`**

The operator presses [5] when some particular details are required. A "Details" menu is shown for the operator to choose.

```
*** Details ***
[1] Passenger
[2] Class
[3] Back
*****************
Option (1-3):
```

R5.1 **`[1] Passenger`**

The operator can search a particular passenger by entering the passport ID. If there is a match, the ticket details (i.e. name, passport ID, ticket class type and seat location) correspond to that passenger is displayed.

R5.2 **`[2] Class`**

The operator enters a particular class type to show the list of passengers of that class. The list should contain the seat locations and the passenger's name of a seat, in **ascending** order of seat location. The word "vacant" is shown if the seat is not occupied.

R5.3 **`[3] Back`**

The operator presses [3] to go back to the main menu.

R6 **`[6] Exit`**

The operator presses [6] on duty release. The system prompts the operator whether uploading to the central database is done. The operator has to affirm this before the system can end.

Reminder

1. Suitable checking on user's input is expected, except in situation with assumptions stated in the requirements above. Appropriate error messages should be printed whenever unexpected situation happens, e.g. input value out of range, incorrect date format, etc.

2. The use of functions (in addition to main function), arrays and classes are expected in your program. Appropriate comments should be added in your source code file as well.

Submission

Submit the files below via Moodle (class page) by 18:00 - 23 APR 2021 (Friday). **Late submission is not accepted**.

- Source File: Each group submits **one** source code file (`JetAssign.cpp`).
- Peer-to-peer Evaluation: Each student fills in and submits the peer-to-peer evaluation form (i.e., SEHH2042_P2P_Evaluation.docx).

Grading criteria

Your program will be executed with different test cases in Microsoft Visual Studio. Any deviation from the requirement is considered as incorrect and no mark is given for that case. Your program will also be marked based on its user-friendliness and creativity.

Criteria Aspect	Percentage
Program correctness (Follow ALL instructions, marks deduction on errors found)	70%
Program design (Appropriate use of functions, use of class, modularity, etc.)	10%
Program standard (Use of variable names, indentation, line spacing, clarity, comments, etc.)	5%
Algorithm design (Use of reasonable algorithms and data structures)	5%
User-friendliness (Clear guidelines to users, messages to users, etc.)	5%
Creativity and critical thinking (Additional useful features)	5%
Total (Group Mark)	**100% (max)**

Note: the length of your program does not affect the grading of the assignment. However, appropriate use of loops, functions, arrays and objects are expected to avoid too many repeated codes in your program, which contributes to the program design score of this assignment.

Individual mark is determined by both group mark (80%) and percentage of individual contribution (20%), where the percentage of individual contribution is directly proportion to the average marks given by group members in the peer-to-peer evaluation form.

Marks deduction

Late submission: ***100% deduction.*** ***No late submission is allowed.***

Syntax error: ***100% deduction***. **No syntax error is accepted.** You will get **0** mark if your program fails to be compiled.

Runtime error: **No mark** for the particular test case that triggers the runtime error.

Logic error (bug): **No mark** for the particular test case that deviates from the requirement. Note that a logic error may lead to *failure in ALL test cases of one particular set of requirements*.

Ensure the originality of your work.

Plagiarism in any form is prohibited.

- End -