CSCI-1200 Data Structures — Spring 2017
Homework 1 — Airline Seating

Before starting this homework, make sure you have read the Collaboration Policy & Academic Integrity.

Please read this entire document before attempting to implement this assignment.

Acme Airlines is a budget airline that flies several different types of planes. Each plane has a different seating arrangement. Some planes have first class and/or business class sections. All planes have a coach section. The diagram below shows the seating arrangement for a typical Acme plane:

```
F . ! . . ! . .
F . ! . . ! . .
B . . ! . ! . .
B . . ! . ! . .
C . . . ! . . .
C . . . ! . . .
C . . . ! . . .
C . . . ! . . .
C . . . ! . . .
C . . . ! . . .
```

The first letter, F, B, or C, indicates the seat class: **F**irst Class, **B**usiness Class, or **C**oach Class. A '.' indicates a seat; a '!' indicates an open aisle. Seat rows are numbered starting at 1. Aisles, including open aisles (non-seats), are identified by letters starting with 'A'. For example, seat 5A is the first seat in the coach class in the seating arrangement shown above..

Passenger Seating

Passengers choose their seats either by selecting a specific seat such as 2C, or by selecting a seating section, such as section C at a kiosk. Frequent fliers in coach or business class may be bumped up to the next highest class if a seat in their requested class cannot be found.

Command Line Arguments

Your program will read 2 command line arguments. The 1st argument is the name of the file containing the seating arrangement as described above. The 2nd argument is the name of a file containing passenger seat requests and commands. Your program will process the command file and assign passengers to seats, cancel seat requests, process possible upgrades, and print upgrades and the current seat assignments.

Command File Format

The command file is a text file containing passenger requests and commands. The first line of the file contains an integer flight number. The remaining lines may contain zero or more copies of each of the following commands. Each command occupies a single line in the file. The individual elements on the line are separated by whitespace.

SRN Passenger_ID Seat_class Frequent_flier - Passenger_ID is a unique 4 character code identifying the passenger; seat_class is one of F, B, or C; frequent_flier is either Y or N to indicate whether or not the passenger is a frequent flier on the airline. If the passenger is a frequent flier requesting business or coach class and no seat is available in the requested class, the passenger becomes available to be bumped to the next higher class. Acme Airlines prides itself on passenger comfort. Passengers are assigned to window seats (seats in the first or last position in a row) or aisle seats (seats next to a position marked '!') before

filling in middle seats, processing seats left to right in the rows. If no window or aisle seat is available in the section, the passenger is assigned the next available seat.

SSR Passenger_ID Seat_ID Frequent_flier - Passenger_ID is a unique 4 character code identifying the passenger; seat_ID identifies a specific seat by row and column, for example, 5C indicates a seat in row 5 column C; frequent_flier is either Y or N to indicate whether or not the passenger is a frequent flier on the airline. If the requested seat is not available, an attempt is made to find another seat in the class. In this case, seats are assigned in the same manner as with the SRN command. If the passenger is a frequent flier requesting business or coach class and no seat is available in the requested class, the passenger may be bumped to the next higher class. If the passenger requests an invalid seat, for example for a row or column that doesn't exist, an error message is printed (see below).

CXL Passenger_ID - cancel passenger Passenger_ID's seat request. Remove her/him from the seating chart and the collection of passengers scheduled for an upgrade.

UPL - print the Passenger_IDs of passengers scheduled for possible upgrades.

UPP - Upgrade the passengers scheduled for upgrades to a higher seating class.

PRT - Print the seating chart.

Command Processing and Program Output

Program output will be written to `std::cout`. Error messages will be written to `std::cout`. The format of these messages is important. On Submitty, the output produced by your program, will be compared against standard output files

SRN - If a seat was found, the program should print a message to std::cout:

```
Passenger xxxx was given seat yy.
```

where xxxx is the 4 character passenger ID and yy is the seat ID. For example,

```
Passenger O3JM was given seat 4G.
```

If a passenger requests a seating class other than F, B, or C, the program should print an error message to `std::cout`:

```
Invalid seating class - z.
```

where z is the invalid seating class. For example,

```
Invalid seating class - D.
```

If no seat can be found in the requested seating class, a message is printed to `std::cout` if the requested seating class was F, B, or C:

```
Could not find a seat in the z section for passenger xxxx.
```

where z is seating class.

SSR - If the requested seat is found, the program should print a message to std::cout:

```
Passenger xxxx was given seat yy.
```

where xxxx is the 4 character passenger ID and yy is the seat ID.

If the passenger requests an invalid seat (non-existent or aisle seat), a message is printed to `std::cout`:

```
Passenger xxxx requested an invalid seat yy.
```

where xxxx is the 4 character passenger ID and yy is the seat ID.

If the passenger requests an occupied seat, a message is printed to `std::cout`:

```
Passenger xxxx requested an occupied seat yy.
```

where xxxx is the 4 character passenger ID and yy is the seat ID.

When a passenger has requested an invalid seat, an attempt is made to find a seat for the passenger in the same class. If the passenger requested an invalid seat that contains a row or column greater than number of rows or columns in the plane, no attempt is made to seat the passenger.

For **SRN** and **SSR** commands, a frequent flier may become available for an upgrade if a seat cannot be found in the seating class.

CXL - The seat occupied by the passenger ID is made available. If the passenger is scheduled for a possible upgrade, the passenger_ID is removed from the collection of possible upgrades. If the passenger can't be found, a message is printed to `std::cout`:

```
Passenger xxxx could not be found.
```

where xxxx is the 4 character passenger ID.

UPL - Passengers scheduled for a possible seating upgrade are listed to `std::cout` as in the following example:

```
The following passengers are scheduled to be upgraded to first class:
53NK
The following passengers are scheduled to be upgraded to business class:
VAMT
CFG3
```

UPP - Passengers scheduled for a possible upgrade are upgraded to the next higher seating class. If a seat was found for the passenger, a message is printed to `std::cout`:

```
Passenger xxxx was given seat yy.
```

where xxxx is the 4 character passenger ID and yy is the seat ID.

A message is also printed to `std::cout` indicating that the passenger has been upgraded:

```
Passenger xxxx was upgraded to the z section.
```

where xxxx is the 4 character passenger ID and z is the seating section.

If no seat can be found in the requested seating class, a massage is printed to `std::cout`:

```
Could not find a seat in the z section for passenger xxxx.
```

where xxxx is the 4 character passenger ID and z is the seating section.

PRT - Print the current seating chart to `std::cout`. The seating chart has is formatted as in the following example:

```
Seating Chart for Flight 302
   A    B    C    D    E    F    G    Seating Class
1  23FX !!!! RDQH .... !!!! MFOX .... F
```

```
 2  .... !!!! .... 5865 !!!! .... .... F
 3  A3HK PKAP !!!! I5X3 !!!! RRH6 .... B
 4  1OZO .... !!!! .... !!!! .... .... B
 5  O2ZC S114 W7J9 !!!! CZJO CWC1 5YC1 C
 6  1YX8 ZF4O 2D4N !!!! UOJ1 Z6SV ZZ6E C
 7  F31V B86X NO2F !!!! QUBO .... .... C
 8  M3SP V283 D379 !!!! 5GR6 .... .... C
 9  N4QW 4P6T R1DY !!!! S8C9 .... .... C
10 BHBL .... LXUU !!!! 8J9M .... D60B C
```

Available seats are indicated by "....". If a seat is occupied, the passenger ID is displayed. Aisles are indicated as "!!!!". Passenger IDs are 4 characters wide. There is a single space between seats, aisles, and the seating class. There is a tab character (\t) after the row number.

If an invalid command is found in the command file, a message is printed to `std::cout`:

```
Invalid command: www."
```

where www is the invalid command.

Implementation

To implement your code, you may not use any data structure not yet covered in lecture. This means that we expect your code to use strings, vectors, and/or arrays. You should not uses structures such as STL lists or other structures that we have not yet introduced.

2D Vectors

A convenient data structure for representing the the seating chart is a two dimensional vector, i.e. a vector of vectors. The following code shows how to read strings from std::cin and create and fill a 5 x 3 2D vector of strings:

```
std::vector<std::vector<std::string> > s;
for (unsigned int i = 0; i < 5; ++i) {
  std::vector<std::string> temp_vector;
  for(unsigned int j = 0; j < 3; ++j) {
     string temp_string;
     std::cin >> temp_string;
     temp_vector.push_back(temp_string);
  }
  s.push_back(temp_vector)
}
```

Provided Code

We have provided you with some helper code in main.cpp to parse seat IDs. You are not required to use it if it doesn't fit with your program design.

Examples and More Information

Name your program file main.cpp. Sample input and output files are posted on the course website, and the validation and automatic grading on the submission server will also help you check your work. More information on command line arguments and file I/O in C++ is available on the course web page under "Misc. C++ Programming Info".

On Submitty, your program will be compile and run from the command line. For example:

```
./a.out acme_small_seats.txt acme_flight100_small.txt
./a.out acme_small_seats.txt acme_flight119_small.txt
```

Submission Details

Do your work in a folder named `hw1` within your Data Structures homework directory. Organize your program into logical, reasonable-length functions – don't put all of the code in `main`! Be sure to read the "Homework Grading Criteria" as you put the finishing touches on your solution. Create new test cases to fully debug your program and don't forget to comment your code! Use the provided template `README.txt` file for notes you want the grader to read.

In the `README.txt` file, describe how you implemented passenger upgrades to a different seating class. Describe how you dealt with the case of a CXL command where the passenger was scheduled to be upgraded.

You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.