**Project 2 Specifications**

**Priority Queue**

Please review the general [*Program Guidelines and Evaluation Criteria*](https://aacc.instructure.com/courses/64613/pages/project-guidelines-and-evaluation-criteria) for information about projects, and download files needed for this project.

**Learning Objectives:**

1. Develop a high-quality C++ program, following a specified Style Guide and incorporating Javadoc comments.
2. Draw a complete UML class diagram for the program.
3. Create a makefile for a C++ program.
4. Incorporate priority queue data structure software written by the textbook author into your program.
5. Input data from multiple text files.
6. Save program results to multiple text files.
7. Apply secure programming practices when using text files
8. Create a program that simulates a real-world situation using a priority queue.
9. Create records for a priority queue based on data from a text file.
10. Insert records into multiple priority queues, setting an appropriate priority value for each.
11. Delete records from multiple priority queues.
12. Apply secure programming practices when using a priority queue.
13. Perform simulation calculations based on the order in which records are deleted from the heaps.
14. Create output files that are accurate, complete, easily identifiable, and user-friendly.

***Specifications:***

Develop a high-quality, object-oriented C++ program that performs a simulation using a heap implementation of a priority queue.

A simulation creates a model of a real-world situation, allowing us to introduce a variety of conditions and observe their effects. For instance, a flight simulator challenges a pilot to respond to varying conditions and measures how well the pilot responds. Simulation is frequently used to measure current business practices, such as the number of checkout lines in a grocery store or the number of tellers in a bank, so that management can determine the fewest number of employees required to meet customer needs.

Airlines have been experimenting with different boarding procedures to shorten the entire boarding time, keep the flights on-time, reduce aisle congestion, and make the experience more pleasant for passengers and crew. A late-departing flight can cause a domino effect:

* the departure gate is tied up and cannot be used by other landing or departing flights,
* passengers on board the late flight may miss connecting flights and require rebooking and possibly overnight arrangements (meals and lodging), etc., and
* passengers complain about being late, and/or about having to rearrange their plans.

Thus, late flights have a huge operational impact.

For this project, we will simulate boarding procedures for Airworthy Airlines.  The Airline's current procedure is as follows:

* **pre-board** in the **following order**:
  + families with young children or people who need help (e.g., wheelchair)
  + first class and/or business class passengers
  + elite passengers (frequent fliers) and those passengers seated in exit rows
* conduct **general boarding** in reverse, from the back of the plane to the front in the **following order**:
  + rows 23-26
  + rows 17-22
  + rows 11-16
  + rows 5-10

Airworthy is considering revising their boarding procedure such that general boarding is done randomly, meaning the first passenger in line for general boarding is the first passenger to board (i.e., general boarding passengers all have the same priority). Airworthy suspects this random general boarding method will improve the flow of passengers, getting them on board and seated more quickly. It is also less labor-intensive for Airworthy's customer service agents because it significantly reduces the number of boarding announcements required and eliminates confrontations with customers trying to board "out of turn."  Note that the revision is to general boarding only.  The pre-boarding procedures will not be changed.

Develop an object-oriented C++ program that simulates these two boarding procedures using a heap implementation of a priority queue. Assume that all passengers are already checked in and at the gate area when boarding begins for both scenarios. The simulation should model an Airbus A320-200 plane, which is configured as follows:

* Rows 1 through 4  are first class seats (4 seats in each row)
* Rows 5 through 26 are coach class seats (6 seats in each row)
* Rows 10 and 11 are exit rows

Your program must consist of the Heap\_PriorityQueue class from the textbook (which inherits from PriorityQueueInterface and uses the ArrayMaxHeap and HeapInterface classes, which are also provided), the PrecondViolatedExcep class from the textbook, the Passenger class and Airworthy class that you create, and a separate file named SimulationProject.cpp that contains the main() method. When designing and implementing the program, apply good software engineering principles. Create a makefile for the program. Be sure to follow the style guide and be sure to use Javadoc style comments appropriately.

Start the analysis and design process by drawing a complete UML class diagram for the program that includes the application and all the classes that are contained in the program, including the classes provided by the textbook and the classes that you create. After you have completed the program, update the UML class diagram to reflect the completed implementation.

The files for the Heap\_PriorityQueue, the PriorityQueueInterface, the ArrayMaxHeap, the HeapInterface, and the PrecondViolatedExcep classes (Heap\_PriorityQueue.h, Heap\_PriorityQueue.cpp, PriorityQueueInterface.h, ArrayMaxHeap.h, ArrayMaxHeap.cpp, HeapInterface.h, PrecondViolatedExcep.h, and PrecondViolatedExcep.cpp) are provided in a zipped folder, along with input files for the program. Note that the priority queue must store Passenger objects, and the > and < operators must be defined for that class because the ArrayMaxHeap class uses those overloaded operators.

Develop a **Passenger** **class** that stores the following data for an Airworthy Airlines passenger:

* key - the priority value for the PriorityQueue
* passenger's last name
* passenger type, a character, where 'H' is a child or passenger who needs help in boarding, 'E' is an elite passenger (frequent flyer), and 'G' is a general boarding passenger
* row where passenger is seated; must be a number between 1 and 26, where rows 1-4 are first class and rows 10 and 11 are exit rows

Include a constructor and accessors and mutators for each attribute. In addition, you **MUST** include methods that overload the < and > operators. You may include other methods, if needed.

Develop an **Airworthy** **class** that contains the following data members:

* exactly one priority queue where the priority is set using Airworthy's previous boarding procedure
* exactly one priority queue where the priority is set using Airworthy's new random boarding procedure
* the amount of time, in seconds, required to board a plane using Airworthy's previous boarding procedure
* the amount of time, in seconds, required to board a plane using Airworthy's new random boarding procedure

You may use additional attributes, as needed. The Airworthy class must include methods to support the following functions of the class:

* a constructor
* read the data from the input file
* load the two priority queues
* run the simulation

Be sure to incorporate adequate error checking for all files and for the priority queue, providing error messages as needed. When setting the priority (the key) for a passenger, keep the pre-boarding procedure described above in mind.

When running the simulation, assume that:

* A passenger who is not blocked by another passenger requires one second to board.
* A passenger is blocked when the **previous passenger** is sitting **in the same row** or **a row closer to the front of the plane**. A blocked passenger requires 25 seconds to board.

For example,

Passenger 1 seated in row 5 - not blocked (no one in front) - add 1  
Passenger 2 seated in row 3 - not blocked (previous passenger's row is 5) - add 1  
Passenger 3 seated in row 10 - blocked (previous passenger's row is 3) - add 25  
Passenger 4 seated in row 15 - blocked (previous passenger's row is 10) - add 25  
Passenger 5 seated in row 9 - not blocked (previous passenger's row is 15) - add 1  
Passenger 6 seated in row 7 - not blocked (previous passenger's row is 9) - add 1  
Passenger 7 seated in row 9 - blocked (previous passenger's row is 7) - add 25

The **SimulationProject.cpp** file contains a main() method and a method named description() that provides a detailed description for the user, explaining what the program is doing, how it works, and the location of all output files. Note that the description() method does **NOT** substitute for Javadoc comments. The audience for the description() method consists of non-technical users who have no information at all about the program or the assignment. The main() method must support the following:

* call the description() method
* call methods of the Airworthy class to load the priority queues and run the simulation using both the previous boarding procedure and the new random boarding procedure for each of 3 different input files.  Each input file contains, on each line, the last name of the passenger, the type of passenger, and the row number in which the passenger is seated. The input files are:
  + airworthy100.txt which loads the plane at 100% of capacity
  + airworthy85.txt, which loads the plane at 85% of capacity
  + airworthy70.txt, which loads the plane at 70% of capacity
* call methods of the Airworthy class to create 3 different output files named results100.txt, results85.txt, and results70.txt. Each output file should contain the data related to its similarly named input file. Be sure to title each part of the output and label all data so it is easy to identify. Each file should contain:
* a list of each passenger read from the input file, showing the passenger's name, type, and row
* a list of each passenger in order as they are removed from the priority queue when running the simulation using the current boarding procedure, showing the passenger's name, type, row, and priority value (the key)
* the total number of **minutes** (shown with 2 decimal places) required to board the plane using the current boarding procedure,
* a list of each passenger in order as they are removed from the priority queue when running the simulation for using the new random boarding procedure, showing the passenger's name, type, row, and priority value (the key), and
* the total number of **minutes** (shown with 2 decimal places) required to board the plane using the new random boarding procedure.

Be sure to follow the Project Guidelines & Evaluation Criteria, since the project will be evaluated using this criteria. In addition, be sure to use javadoc-style comments appropriately. When creating javadoc-style comments, keep in mind that the comment will eventually become part of an html file that will be used by other programmers on your programming team, and by maintenance programmers. Remember also that maintenance programmers have not seen the assignment (the specification), so the information you are providing here must provide all of the detailed information another programmer will need to completely understand the program and to maintain the code. This program is worth 10% of your overall course grade, and it will require a major time investment.

***Output Examples (Assume 30 passengers):***

*(When the program reads an external file and runs, it should generate the results as the following example, and the results are saved into another output file.)*

Original List in the file:   
  
Vang G 23  
Velazquez G 5  
Langley G 17  
Buckner G 14  
Barlow G 18  
Guthrie G 11  
Hyde G 21  
Delacruz H 22  
Foreman G 7  
Hewitt G 12  
Downs G 15  
Franks G 7  
Hendrix G 8  
Finch E 10  
Mcleod G 10  
Workman G 21  
Byers G 15  
Mcfadden E 13  
Goff G 14  
Madden G 5  
Burris G 2  
Talley G 24  
Tyson E 20  
Lancaster G 18  
Burks E 11  
Hahn G 8  
Clements E 24  
Holden G 23  
Witt G 6  
Snider E 22  
  
Boarding in Previous Procedure ...  
  
Last Name Type Row Key   
Delacruz H 22 30   
Burris G 2 29   
Guthrie G 11 28   
Finch E 10 27   
Mcleod G 10 26   
Mcfadden E 13 25   
Tyson E 20 24   
Burks E 11 23   
Clements E 24 22   
Snider E 22 21   
Vang G 23 20   
Talley G 24 19   
Holden G 23 18   
Langley G 17 17   
Barlow G 18 16   
Hyde G 21 15   
Workman G 21 14   
Lancaster G 18 13   
Buckner G 14 12   
Hewitt G 12 11   
Downs G 15 10   
Byers G 15 9   
Goff G 14 8   
Velazquez G 5 7   
Foreman G 7 6   
Franks G 7 5   
Hendrix G 8 4   
Madden G 5 3   
Hahn G 8 2   
Witt G 6 1   
  
Total boarding time: 6 minutes and 54 seconds.  
  
  
Boarding in Random Procedure ...  
  
Last Name Type Row Key   
Delacruz H 22 30   
Burris G 2 29   
Guthrie G 11 28   
Finch E 10 27   
Mcleod G 10 26   
Mcfadden E 13 25   
Tyson E 20 24   
Burks E 11 23   
Clements E 24 22   
Snider E 22 21   
Vang G 23 20   
Velazquez G 5 19   
Langley G 17 18   
Buckner G 14 17   
Barlow G 18 16   
Hyde G 21 15   
Foreman G 7 14   
Hewitt G 12 13   
Downs G 15 12   
Franks G 7 11   
Hendrix G 8 10   
Workman G 21 9   
Byers G 15 8   
Goff G 14 7   
Madden G 5 6   
Talley G 24 5   
Lancaster G 18 4   
Hahn G 8 3   
Holden G 23 2   
Witt G 6 1   
  
Total boarding time: 6 minutes and 30 seconds.

***Grading rubrics***

**Output (35 points)**

* The makefile is correct
* The program displays a description that is adequate for a non-technical user who is totally unfamiliar with the program and/or the assignment; the names of all output files are provided
* The output files contain appropriately titled and labeled output that includes the correct original list of passengers showing name, type, and row, a list of each passenger removed from the priority queue using the previous boarding procedure showing name, type, row, and priority value, the total number of minutes required for the previous boarding procedure showing 2 decimal places, a list of each passenger removed from the priority queue using the new random boarding procedure showing name, type, row, and priority value, and the total number of minutes required for the new boarding procedure showing 2 decimal places
* The output files list the passengers in an order that meets all the requirements of each boarding procedure
* The output files specify a reasonable value for the number of minutes required to board for each boarding procedure, based on the input data

**Algorithm (55 points)**

* Passenger class
  + Stores the priority value and other specified private attributes using appropriate data types
  + Contains accessors and mutators for each data member and methods that overload the < and > operators
  + All methods of the Passenger class support good software engineering principles, error handling and fail-safe programming, and modularity and use appropriate C++ constructs
* Airworthy class
  + Sets an appropriate priority for each boarding procedure and correctly determines the amount of boarding time for each blocked and unblocked passenger
  + Defines the specified private data members using appropriate data types
  + Defines a default constructor that initializes the attributes to something appropriate for a nonexistent airworthy object
  + Defines methods to read an input file, load the priority queue, and run the simulation
  + All methods of the Airworthy class support good software engineering principles, error handling and fail-safe programming, and modularity and use appropriate C++ constructs
* Simulationproject.cpp
  + Contains the main() method and the description() method
  + The main() method that opens and closes all files used for input and output in the program, calls appropriate methods in the Airworthy class to load the queues and run the simulation for each boarding procedure using each of the 3 files specified, and creates 3 different correctly named output files
  + The methods contained in the simulationproject.cpp file support good software engineering principles, error handling and fail-safe programming, and modularity and use appropriate C++ constructs
  + If the program is unable to open the input files or create any of the output files, the program provides an appropriate error message and behaves in a reasonable way

**Style (10 points)**

* The program adheres to all the style criteria for documentation
* The program adheres to all the style criteria for readability
* The program adheres to all the style criteria for modifiability

Top of Form