CSE 6010
Assignment 2
Software Modules

Initial Submission Due Date: 11:59pm on Thursday, September 14
Final Submission Due Date: 11:59pm on Thursday, September 21
Peer review and code reflections assignments due September 28 will be posted separately
Submit files as described herein to Gradescope through Canvas
48-hour grace period applies to all deadlines

This assignment consists of writing a program to implement a program module for a priority queue (PQ) of flight information using a **singly-linked-list** data structure. The PQ code should be structured as if it were part of a software library to be used with other programs written by someone else. In fact, like in Assignment 1, you will write only the file that implements the functions below; you will be provided with a main.c file and a header file along with a template code for the rest. You should assume that the items to be stored in the PQ are structures that contain information about a given flight: specifically, the airline, flight number, and departure time.

The software should implement the following interface.

1. Use a `typedef` to define a data structure called `Flightinfo`, defined as a C `struct` representing the information for a flight to be stored in the priority queue. It should include three fields: an `airline` field consisting of a single character, an integer `flightnumber` (you may assume it to be positive), and an integer `time` representing the departure time (you may assume it to be positive). The time will be represented as military time: e.g., 1523 represents 3:23pm. You may assume no invalid times will be used (e.g., 5023, 488). The implementation of the Flightinfo data structure should not be visible outside the module you are creating.

2. Use a `typedef` to define a data structure called `Item`, defined as a C `struct` representing a PQ item. Its members should include a field `flight` of type `Flightinfo`; an integer pointer `priority`, which will point to the `time` field of the `flight` member; and a field `next` that is a pointer to the next item in the PQ.

   Note that although `typedef`'s typically are implemented in header (.h) files, because main.c will require access to the header file and we do not want main.c to know about the `Flightinfo` details, we will include the typedef definitions for `Flightinfo` and `Item` along with the function implementations in pq.c for simplicity. However, we must also include an incomplete definition of `Item` in pq.h, because it is needed for the function prototypes, but it includes a member of type `Flightinfo`. Don't worry; it will all work!

3. Write a function with the prototype

   ```
   Item *Initialize();
   ```

   Create a new node to set up the initial PQ and return a pointer to it. The PQ should be set up to include a single "dummy node" representing a "dummy flight" with the `time` field and `flightnumber` field both set to -1 and the `airline` field set to ' ' (a space character). The

`priority` field of the Item should be a pointer to the dummy flight's `time` field. Set the `next` field appropriately. Return `NULL` if the operation fails, e.g., due to a lack of memory.

4. Write a function with the prototype

```
int Add(Item* mypq, char itemAirline, int itemFlightnumber, int
itemTime);
```

Insert the item of type `Flightinfo` into the PQ that begins with `mypq`. Verify that the `itemFlightnumber` is positive and that the `itemTime` is nonnegative (you may assume they are both integers) before inserting into the PQ. Return 0 if the operation succeeds, or 1 if it fails (including if a non-positive value was entered for the `itemFlightnumber` or a negative value for the `itemTime`). You do not need to perform any other validation of the `itemFlightnumber` and `itemTime`. You may assume that flightnumbers are unique, and you do not need to test this case explicitly.

5. Write a function with the prototype

```
void Remove(Item* mypq);
```

Remove the node with the highest priority from the PQ, if the PQ is not empty.

6. Write a function with the prototype

```
void Print(Item* mypq);
```

Print the Flightinfo items of the items stored in the PQ in that begins with `mypq` in order from first to last, without changing the contents of the PQ. You can use the following formatted print statement before printing the contents:

```
printf("Priority queue contents:\n");
```

and the following formatted print statement for each item in the PQ, which you may copy and paste (note there is one space between items):

```
printf("%c %d %d\n",temp->flight.airline,
   temp->flight.flightnumber, temp->flight.time);
```

7. Write a function with the prototype

```
int Count(Item* mypq);
```

Return the number of items currently in the PQ that begins with `mypq` or 0 if the PQ is empty (that is, it contains only the "dummy" entry).

8. Write a function with the prototype

```
int CountAirline(Item* mypq, char myairline);
```

Return the number of flights currently in the PQ that begins with `mypq` whose `airline` member's value is the same as the specified `myairline` value, or 0 if the PQ is empty.

9. Write a function with the prototype

   ```
   int CountEarlier(Item* mypq, int mypriority);
   ```

   Return the number of flights currently in the PQ whose `priority` is smaller in value than the specified `mypriority` (not larger and not equal). Return 0 if the PQ is empty.

10. Write a function with the prototype

    ```
    int CountLater(Item* mypq, int mypriority);
    ```

    Return the number of flights currently in the PQ whose `priority` is larger in value than the specified `mypriority` (not smaller and not equal). Return 0 if the PQ is empty.

11. Write a function with the prototype

    ```
    int CountSmaller(Item* mypq, int itemflightnumber);
    ```

    Return the number of flights currently in the PQ whose `flightnumber` member is smaller in value than the specified `itemflightnumber` (not larger and not equal). Return 0 if the PQ is empty.

12. Write a function with the prototype

    ```
    int CountLarger(Item* mypq, int itemflightnumber);
    ```

    Return the number of flights currently in the PQ whose `flightnumber` member is larger in value than the specified `itemflightnumber` (not smaller and not equal). Return 0 if the PQ is empty.

13. Write a function with the prototype

    ```
    void Finalize(Item* mypq);
    ```

    Delete the PQ that begins with `mypq` by releasing all memory used by all the contents of the data structure. Return 0.

You should define all these items using pq.h (which is provided in full) and pq.c (which you will complete). Along with pq.h, we will also provide an example of a main.c file that uses pq.h and pq.c to implement your PQ. As mentioned above, **you are responsible for writing pq.c based on the provided template file** (which includes the typedefs). A Makefile is also provided.

There may be additional functionality to tested beyond what is included in the main.c file provided; you are responsible for testing the specified functionality fully by altering main.c file (but you will not submit your altered main.c). In particular, you should ensure that your program can properly handle the following cases.

- Adding a flight whose flightnumber is a non-positive number (error case).
- Adding a flight whose time is a negative number (error case).
- Removing from an empty PQ (error case).
- Printing from an empty PQ.
- Counting the number of items in an empty PQ.
- Counting the number of items with a specified airline when the PQ is empty.
- Counting the number of items with a flight number before or after a specified flight number when the PQ is empty.
- Counting the number of items with a priority before or after a specified flight number when the PQ is empty.

Although you should test that your program can handle many cases, you do not need to demonstrate all of them in the test output you will submit on your submission slides because you are limited to only 2 slides.

Please be sure your code does not have any memory leaks; you may use valgrind for this purpose. (See Workshop 1 for more information on valgrind.)

To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

Final submission: You should submit to Gradescope through Canvas the following three files.

Your submission should include the following files:

(1) your **pq.c** file. **Do not submit pq.h, main.c, or Makefile.**

(2) a text file (not formatted in a word processor, for example; no .docx, .rtf, .pdf formats) named README.txt that includes the compiler and operating system you used for compiling and running your code along with instructions on how to compile and run your program. (It is expected that this file will be quite short and you may just indicate you used the provided Makefile, if you used it.)

(3) a series of 3 slides **saved as a PDF** and named slides.pdf, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any if statements you used to change the flow of the program, the purpose of any functions you created, etc.). You are limited to one slide.
- Slides 2-3: a description of your most important tests and sample output produced by the test program. You are limited to two slides; we understand you will not be able to demonstrate all functionality here, so focus on what you think is most important.

Initial submission: Your initial submission does not need to include the slides or README. It will not be graded for correctness or even tested but rather will be graded based on the appearance of a good-faith effort to complete the majority of the assignment.

Some hints:

- Start early!
- Identify a logical sequence for implementing the modules. Then implement one at a time and verify each works properly before proceeding to implement the next module.