CSCE 221 Assignment 5 Cover Page

First Name Last Name UIN

User Name E-mail address

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: `http://aggiehonor.tamu.edu/`

Type of sources				
People				
Web pages (provide URL)				
Printed material				
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.
On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Date

CSCE 221 – Prelab for the Final Project, Fall 2013 (100 points)

Due: November 8, 2013 at 11:59 pm

Hardcopy Report: Return to your TA in lab

Problem description (80 points)

1. Consider the following interface for minimum priority queue:

   - `insert(k,x,&loc)` inserts a key `k` and element `x`; The address of a locator `loc` is passed and updated after insertion.
   - `min()` returns the locator to an item (pair - key and element) with the smallest key (but it does not remove it)
   - `remove(loc)` removes the item with locator `loc` and updates the minimum priority queue
   - `isEmpty()` tests whether the queue is empty
   - `decreaseKey(loc, k)` updates the minimum priority queue after the key `k` with locator `loc` was replaced

   - `createPriorityQueue(file, locArray)` reads (key, element) pairs from the `file` and initializes the input array of locators `locArray`

2. Implement minimum priority queue, once as an unsorted array or linked list and the other time as a minimum binary heap.

3. Implement both data structures with locator pattern to support all the above operations, see the lecture notes and Chap. 8 for more details about Priority Queues and Adaptable Priority Queues section 8.4 p. 357. An illustrated example of heap and locators also can be found at file “PQ-Example.pdf” on the course website.

   NOTE: It is important that locators are stored "outside" the priority queue (PQ), so later you can locate any item in the PQ from outside the PQ. Therefore, you have to pass the locator in to or out of the PQ functions, particularly, the functions that insert items to the PQ, so the locators outside are associated with the items/pairs in the PQ.

   Without locators, to find a particular element in the PQ, you can only go through all the elements, which takes O(n) time. The locators can be stored in any efficient structure outside the PQ: random-access array, hash table or search tree, which costs O(1), O(1) and O(log n) to find a specific locator using its id (say, city name or item No.). The sample code gives you an example of hashing different cities to an array of locators.

4. Get the number of comparisons for the `remove(loc)` and `decreaseKey(loc, k)` operations for both the implementation. For example, for a heap you count comparisons used to remove the smallest element and then correct the heap structure.

5. You may define your locator class as follows, which is consistent with the code in the lecture slides.

```
template<typename ElemType>
class Item {
private:
  int key;
  ElemType elem;
  Locator<ElemType>* loc;  // a pointer to the corresonding locator
public:
```

```
  Item(const int k=0, const ElemType& e=ElemType(), Locator<ElemType>* l=NULL):
    key(k), elem(e), loc(l) { } //constructor
};
template<typename ElemType>
class Locator {
private:
  Item<ElemType>* pItem; // A pointer to the corresonding item in the PQ
                         // if the PQ is implemented with an array or list
  // int indexItem;      // Alternative way: the index of the item in the PQ
                         // if the PQ is implemented with an array or vector
  Locator( Item<ElemType>* i = NULL ): pItem(i) { }
};
```

6. You are provided with a sample test program `Main.cpp`. In the `main()`, a minimum priority queue and an array of locators are created.

   ```
   PriorityQueue<int> pq;
   Locator<int> locArray[26]; // Assume there are at most 26 (key,element) pairs
   ```

   A test file is provided which contains key-element integer pairs. They represent distances and number of stops from the BWI airport to the nine airports, from the textbook figure 13.15(g) at p.642.

   ```
   //key (distance), element (stops) to the city
   2467 -1 // SFO
   3288 -1 // LAX
   621 0 // ORD
   1423 -1 // DFW
   84 0 // JFK
   371 1 // BOS
   328 1 // PVD
   0 0 // BWI
   946 0 // MIA
   ```

   The `main()` creates a minimum priority queue from the data and performs a sort based on distance by removing minimum. Next, the `main()` fills the minimum priority queue again with the same data by insertion. Locators are specially stored, i.e. the initial letter of a city `c` is hashed to a spot in the locator array using `arrayIndex = c-'A'`. Thus, a city can be located in constant O(1) time in the locator array and in the priority queue. Then, its key can be replaced in the priority queue.

   NOTE: You will have to adapt the Main.cpp to your program.

Report (20 points)

In addition to the regular programming assignment, your report should include answers to the following problems:

- Describe your algorithms for `remove(loc)`, `decreaseKey(loc, k)` and `createPriorityQueue(file, locArray)`
- Discuss about number of comparisons for `remove(loc)` and `decreaseKey(loc, k)`
- Provide running time of Priority-Queue ADT for both the representations (unsorted array/linked list and binary heap) express in terms of the big-O notation.
- Compare the running time for the same operations in both the implementations of Priority-Queue ADT.

- Does using the locator pattern have an impact on the complexity of the priority queue operations for the both representations?
- Write about three real-life applications where you can use a minimum priority queue.