CSCI 335
Fourth assignment

Total: 100 points

Due 4/27/2017
Please, follow the blackboard instructions on writing and submitting programming assignments
We will not debug your assignment. It should run correctly to receive credit.

Mutable priority queues

Your task is to combine the implementation of a priority queue with that of hashing. In order to achieve this, one of the private members of your priority queue class should be a hash-table. Use the Binomial Queue implementation. The goal is to be able to search for a specific key in the priority queue (i.e. not only the minimum key), and to be able to remove a specific key (not only the minimum key).
So you have to add the following functionality:

```
PriorityQueue<int> a_priority_queue;
  // Insert a few elements into the queue…
  // Search for an element.
  const int a_value = ... a given integer..;
  if (a_priority_queue.Find(a_value))
      cout << "Found " <<  a_value << endl;
  else cout << "Not found " << a_value << endl;
  // Delete an element.
  const int b_value = … a given integer..;
  while (a_priority_queue.Remove(b_value)) {
    // This code deletes all b_value items.
    // Remove() returns true if item deleted and false otherwise
    // (i.e. item not in the queue).
    cout << b_value << " deleted from queue." << endl;
  }
  // Check if deletion was successful.
  if (a_priority_queue.Find(b_value)) {
     cout << "Serious problem with deletion routine… Need to debug…";
     cout << endl;
  } else {
     cout << "Deletion successful" << endl;
  }
```

Implementation hints: Keep a hash-table of the keys that already in the priority queue (use a binomeal queue). Use any hash table implementation. For any pair `<key,p>`, where p is the position of the key in the priority queue (i.e. the pointer of the node that holds the `key`) , hash on the key, and store the pair `<key,p>` in your hash table. Any time you insert key into the priority queue you should also insert the pair `<key,p>` into the hash table. Similarly, every time you delete a key from the priority queue, you should also delete it from the hash table. In

order to implement the Delete of a given element, you have to add to each node in your Binomial Queue implementation a pointer to its parent node.

Part 1 (40 points)

Implement the mutable priority queue as described and test the search functionality.

Write a program that runs as follows:

./TestInsertAndSearch <input_file_to_create_queue> <input_file_to_check_search>

The program should read all the integers in <input_file_to_create_queue> and insert them into the mutable priority queue. After it is done it should print the message:

"Success inserting N elements into the queue. The minimum element is X"

Then the test program should search for every key in file <input_file_to_check_search>. For each key (i.e. integer) should print the message "Found" or "Not Found" based on whether the key was found in the queue or not. For example:

Key1 "Found"
Key2 "Not found"
…

where Key1, Key2, etc. are the integers in file <input_file_to_check_search>.

Part 2 (40 points)

Now test the remove functionality.

Write a program that runs as follows:

./TestInsertAndDelete <input_file_to_create_queue> <input_file_to_check_delete>

The program should read all the integers in <input_file_to_create_queue> and insert them into the mutable priority queue. After it is done it should print the message:

"Success inserting N elements into the queue. The minimum element is X", where N and X have the correct values according to the input.

The test program should delete every key in file <input_file_to_check_delete>. For each key (i.e. integer) should print the message "Deletion successful", or "Serious problem with deletion". You should also print the new minimum (it will not change if you don't delete the minimum). In order to check whether the deletion is successful, in your code after you delete, search for the item. If the item is still there, then you have a problem. I will check your code to verify that you are doing this test. So the output will look like that:

Key1 "Deletion successful" – "New minimum is " X

Key2 “Serious problem with deletion” – “New minimum is “ X

…

where Key1, Key2, etc. are the integers in file <input_file_to_check_deletion>. Of course, if your implementation is correct, deletion will be successful for all keys. See code at the beginning of the assignment to see code you have to write in order to check whether the deletion was successful.

Part 3 (20 points)

Write a faster insert(key) function for the binomial queue. In order to achieve that you have to modify the Merge() function and make it specific to the merging of one element only.
Test it using the program:

./TestNewInsert <input_file_to_create_queue> <input_file_to_check_search>

This program will behave in the exact same way as the one in Part1, but it will use the new Insert function. Make sure to specify in your **README.txt** the algorithm for your modified insert.