CSCI 241, Fall 2012

Assignment #3: Heap-based Priority Queues and Hashing (30 points)

Summary

- **Goal**: This assignment is worth 10% of the grade for the course. In this assignment you will implement a heap-based priority queue and using hashing technique.
- **Collaboration policy for this homework**: For this homework, you may optionally work with **one** other student in the class. If you do so, you must indicate in your submission: (1) who your partner was (2) the detailed breakdown of the workload. And the division of the workload in percentage between you two.
- Due time: **end of day (11:59pm), Friday, Nov 16.**
- Hand in: submit a single compressed file via the **Assignment 3 Submission** link on the Blackboard under "Assignments". A Readme file is not required but you may include one if there's anything you want to tell me. Both of the team members need to submit even though the submissions are the same.

Task

In this assignment, we'll be dealing with a voting system using priority queue and hashing.

You are given a data file, which contains nearly 19,000 ballots. Download it. Each ballot is represented by a single line with the name of the selected candidate (case insensitive). We'll just count the votes in the usual way. There are 24 candidates for 9 seats in the selection. So you will end up with a heap of size 24. You will have to perform Insertion, Heapify, Sorting etc. on the data structure in order to get the desired output.

Input and Output (12 pts)

Input will be the ballot file.

Output format:

// You will print a news bulletin whenever the front-runner changes, something like this:

```
Born pulls into the lead with 1 of 1 votes
Sullivan pulls into the lead with 2 of 4 votes
Toomey pulls into the lead with 4 of 17 votes
   ...
```

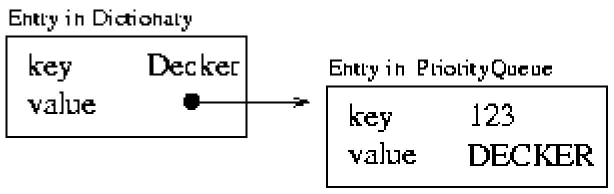
// You will print the final votes for all candidates in descending order:

```
(2730 votes) Galluccio
(1669 votes) Born
(1653 votes) DECKER
```

. . .

Implementations (18 pts):

1. You don't have a list of candidates ahead of time – you will become aware of new candidates only as you see votes from them. The first time a candidate is seen, an entry should be created for that candidate and then inserted into the priority queue. Each entry should at least contain the info of the candidate name and the votes.
2. The priority queue should maintain a heap after an entry is inserted.
3. At any moment, a heapsort is able to render us a winner list for us. We want to get the final votes list in our output.
4. So far, fairly easy. Let's try to improve the search speed. Even though insertion and deletion on a heap take log time, finding an entry's position on the heap takes linear time, which is too slow! Can we make it constant time?
5. After we inserted an entry into the heap, the entry should also be inserted into a look-up table (aka, dictionary, or hash table, or map), keyed by the candidate's name so that it can be found again when the candidate gets additional votes. Don't get confused between the entries in the dictionary and the entries on the priority queue.



6. Remember that you should ignore uppercase/lowercase distinction in candidate names. But when you print a candidate name, you should use the capitalization from the first ballot you saw for that candidate.
7. For the implementation of the dictionary,
    a. You may not make use of any existing dictionary or map package.
    b. You may use any hash function you like to get from the key to the location. You would expect that hash key mod N would give you a result from 0 to N-1 that you can use as an array index.
    c. To handle collision, use "chaining (array of linked lists)" that we will discuss in class.
    d. Here explains why you should probably have a prime number of locations. When constructing the hash table, you should round the user's requested capacity up to the nearest prime number. http://www.cs.unm.edu/~saia/numtheory.html
8. Resources you can use and modify:
    - Resources indicated on the syllabus.
    - Sample codes on http://users.cis.fiu.edu/~weiss/ada.html
9. In your submission, you need to have a hash_table package, a priority queue package and a main program.
10. If you want to reuse the priority queue package from the above link, please modify it to have percolate_up and percolate_down subroutines.
11. Comment on your submission properly.