Homework 4, CSCE 240, Spring 2014

RULES

1. Your C++ code **must** contain files named as required by the `makefile` and be in a directory named `{yourloginid}_hw04`, that has been tarred and gzipped to a gzip file `tar{yourloginid}_hw04.gz`. My loginid is `buell`, so I would be submitting a file `tarbuell_hw04.gz` that would unzip and un-tar to a directory `buell_hw04`.

   You have been provided with a complete directory in which you should be developing your code. If you can run the three `zz*` scripts on a reference machine in the lab and get the output you expect, then I should be able to get the same output and be able to assign positive marks for your submission, because what I will be doing is running the same three scripts on one of the reference machines in the lab.

2. Your program should take its input from standard input and write to standard output, because that's what the scripts will do.

3. Your program must run correctly on the reference machines. The list of reference machines is on the main web page for this course. I am assuming that all the reference machines are identical, so you need only check on one machine. If it happens to work on one reference machine but not another, then you will get the benefit of the doubt.

Assignment

You are to write a program to do tally votes baseed on a ranked choice voting system. This is used in Australia and has been experimented with in thd United States.

RCV works as follows. If there are five candidates for an election, then each voter has to rank all five candidates in order, similar to basketball or football rankings. Candidates ranked first then get a weighted vote of 1, candidates ranked second get a weighted vote of 1, and so forth. This is the way that the college sports rankings are done, but the sports rankings quit there.

Read RCV is somewhat different, as mentioned below.

Your code should run through the votes and compute a weighted vote. The candidate (or candidates) with the least number of weighted votes is then eliminated from contention. All candidates receiving rankings below the removed candidates are shifted up in the ranking, and the weighted total is recomputed. This process continues until there is one candidate left, who is the winner, or until all candidates remaining in contention have the same number of weighted votes. (In real RCV, there would then be some sort of tiebreaking decision.)

The heart of this program is not the computation or the actual data structure that is used. The heart of this program is the logic of how to do this efficiently and cleanly without too much repeated code.

You might also note that this isn't just a voting scheme for elections. There's a great book by Langville and Meyer, *Who's Number One?*, that goes through the math and the computation for ranking. Any sport without a playoff system is the obvious example of when ranking takes place, but this kind of logic is applicable in any situation in which there are more than two "candidates". In a ranking

```
A B C
B C A
C A B
```

each of the three candidates gets six weighted votes, and we have a tie.

Complications

You are encouraged to think about how the program could become more complicated if the rules were made looser.

For example, if it was not required that all candidates be ranked, how would that change things? Certainly the input of data would be different, but that's a small point. More important would be how to score the non-votes. Would all candidates not ranked get tied for a worst-possible ranking? Or would they tie for the next possible ranking? This would affect the tallying.

Also, if this were not ranked choice but at-large, the vote tallying would have to be done differently and this would affect the logic of the program. Traditional at-large voting is that one chooses up to some maximum number of candidates. There are professional organizations, however, whose elections permit one to vote multiple times for the same person, thus weighting an individual's vote.

Classes, Variables, and Functions

You will need a `Candidate` class that is a holder for the candidate name, the weighted vote count, and boolean that says whether or not the candidate is still in contention, and a boolean that says whether or not the candidate is a winner. (I think it's easier to store a value in a variable for these rather than to compute all of them on the fly, because there could be multiple winners and because you don't really want to delete candidates from the list of candidates.)

You will need a `TallyVotes` class that does the real work of the computation.

You may assume that you have been given the number of votes (i.e., number of lines of later input), and that you have the number of candidates (i.e., number of entries per line), and that all votes contain the same number of names (i.e., no one is allowed to abstain). This is done simply to make it easier to read the data.

The initialization function should read the data into `theVotes` and in so doing should create a `map` of `theCandidates`.

To tally the votes, you walk through `theVotes`, each entry of which is a list of names. Candidates get a vote depending on the position in the list (i.e., the first name in a given line of input data is the first choice of that voter. This allows you to compute a weighted vote total.

When you have done the weighted total, you will need to determine the best vote and the worst vote. If these are equal, then everyone is tied and is a winner. If these are not equal, then the worst vote candidate or candidates gets eliminated.

This means that in the next round of voting, that candidate gets skipped. If there was a vote "buell, smith, jones", then in the initial round, buell would get a 1, smith a 2, and jones a 3. If smith were eliminated, then in the next round buell would get a 1 and jones would get a 2 (not a 3).

You can choose to "shift" the candidates in one of two ways. Either keep track of how many names are skipped and adjust your weighted vote accordingly, or else create a new `vector` of candidate names by adding in only those names still in contention, and then processing that `vector` instead of the original vote.

I do not recommend deleting names from the original vote list or deleting candidates from the original `map` of candidates. I think those are bad ideas.

Note also that I have put in two functions that will produce formatted versions of the main data structures. I did that because I needed at least one of then (the candidates and their votes) output in two places in the program, so it was a natural to write a separate function.

Real RCV

In round zero of real RCV, the first scan is to see if any candidate received a majority of the first-place votes. If so, that candidate is elected.

You don't have to do that.