YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 427: Object-Oriented Programming *Handout #7*
Professor M. J. Fischer *March 22, 2016*

Problem Set 4

Due before midnight on Monday, April 11, 2016.

1 Assignment Goals

- Learn an interesting and non-trivial voting system.
- Apply basic OO-design principles to reduce complexity.
- Learn to modify and reuse existing bits of code.
- Learn how to compute with rational numbers using the GMP rational arithmetic routines.
- Learn how to carefully test a program where the correct answers are not obvious.

2 Problem

This assignment is to implement a version of the Single Transferrable Vote voting system. (See https://en.wikipedia.org/wiki/Single_transferable_vote.) Many variations of this system have been described in the literature, so you cannot rely on the internet to give you consistent answers about how "it" works. Instead, I will describe below the version I want you to implement. Do not worry if it differs from other systems that are around, and also do not worry if it is better or worse than some of the other versions.

This problem is very much like the SixOh system you implemented in problem set 3 in that you start with an input file describing the election and the ballots cast, apply the vote counting algorithm, and print out the results. You will find many anologies with the skater ranking problem but also many differences. Some of your code from PS3 can be reused for this problem, but you must be careful to account for the differences.

Compared to the previous problem sets, I'm going to give you fewer details about the classes I want you to create or what functions they should contain. Nevertheless, I want you to produce code with good modular structure and good programming practices. The textbook gives many design principles that should be followed – no global variables, don't put too much code into `main()` or any other single function, keep data members private and put the functions that understand their semantics into the same class rather than manipulating them from outside using setter and getter functions, etc.

I will post some sample inputs and outputs to `/c/cs427/assignments/ps4`.

3 Single Transferrable Voting System

The Single Transferrable Vote (STV) voting system is used to select a subset of *candidates* to fill a fixed number of vacant *seats* according to a set of *ballots*.

Compared to the previous assignment, the candidates correspond to the skaters and the ballots correspond to the judges' marks. But the differences are several.

- In STV, the final result is the *subset* of candidates who are selected to fill the vacant seats. The winners are not ranked, nor are the losers.
- In STV, each voter assigns a rank (ordinal) to candidates directly. In our version, every candidate must be ranked, beginning with 1 for most-preferred candidate. Ties and gaps in the ordinals are not permitted, so if there are n candidates, then each number in $[1, \ldots, n]$ must be used.
- In STV, the actual vote counting process is more complicated, as you will see below.

3.1 The Election File

The election file is a CSV-format file representing a spreadsheet with one column for each candidate and $3 + b$ rows, where b is the number of ballots. The rows are as follows:

1. Row 1 has two fields: "Seats" and the number of seats to be filled by the election.
2. Row 2 has two fields: "Candidates" and the number of candidates.
3. Row 3 contains the names of the candidates.
4. Each subsequent row represents a ballot. It consists of n ordinals giving the ballot's rank for each candidate.

3.2 Counting the Ballots

Once the ballots have been read from the file and checked for validity, the counting process can begin.

The ballots are counted in *rounds*. Each round begins by tallying (or updating) the votes for each candidate and comparing the total with a predetermined threshold called a *quota*. Every candidate with a vote total at least equal to the quota is *elected*. If no candidate is elected in a round, then the candidate with the *smallest* vote total is *dropped* from the race. In either case, at least one candidate is removed from further consideration at each round.

In the case that no candidate is elected, two or more candidates might be tied for the smallest number of votes, making it unclear which one to drop. For this assignment, *drop the first such candidate*. Here, we assume the candidates are in the same order as they were in the input file. By using a deterministic rule rather than choosing one randomly (which might seem fairer to people), your output should match mine.

Each ballot has a current *favored* candidate and a rational number *value* in the rational interval $[0, 1]$. The favored candidate is the highest-ranked candidate that is still *active*, i.e., in the running. The value is the amount of voting power that has not been used so far. Initially, the favored candidate of a ballot is the one of rank 1, and the ballot value is 1. The vote total for each candidate is the total value of all ballots favoring that candidate.

Whenever a candidate is elected, the quota necessary for election is "charged" to the ballots favoring that candidate in proportion to their values. This reduces the ballot's value and is how ballots can end up with fractional values, even though they all start with value 1. Details are given in section 3.4.

For each candidate elected or dropped from the race, the ballots previously favoring that candidate are changed to favor their next-ranked candidate who is still in the race, and the ballot's value is added to the tally of the newly-favored candidate.

At the end of a round, if all seats are filled or there are no remaining candidates, the counting terminates. Otherwise, another counting round takes place. Even though no remaining candidates had quota vote total in the previous round, some candidates might exceed the quota in the next round because of ballots with residual value that used part of their value on the just-elected candidate and now favor the next in line candidate who is still active. This process repeats until the counting terminates.

3.3 Computing the Quota

The quota is the number of votes that a candidate must accrue on a given round in order to be elected. The quota cannot be set too low or it might happen that more candidates were elected than there are unfilled seats. On the other hand, if the quota is set too high, then perhaps no candidates will be elected, and instead they will be dropped one by one.

We choose the formula:

$$\text{quota} = \left\lfloor \frac{\text{number of ballots}}{\text{vacant seats} + 1} \right\rfloor + 1 \tag{1}$$

This number ensures that the number of winners does not exceed the number of seats, since

$$\text{quota} \times (\text{vacant seats} + 1)$$

is strictly greater than the number of ballots.

3.4 Adjusting the Ballot Values

Let E_c be the set of ballots favoring c, and let v_b be the current value of ballot b. The total vote received by candidate c is

$$T_c = \sum_{b \in E_c} v_b. \tag{2}$$

Candidate c is elected if $T_c \geq q$, where q is the quota. In this case, each ballot $b \in E_c$ is *charged* an amount αv_b, where

$$\alpha = \frac{q}{\sum_{b \in E_c} v_b}. \tag{3}$$

Note that $\sum_{b \in E_c} \alpha v_b = q$, so a total of q is charged to the voters who elected c. The value of each $b \in E_c$ is reduced by αv_b, so b's new value is $(1 - \alpha) v_b$.

3.5 Example

An example may help make the above clearer. Suppose there are 6 vacant seats, 20 candidates for those seats, and 200 voters. Initially, each ballot has value 1, so the total value of all ballots is 200. According to equation 1, the quota is $\lfloor 200/7 \rfloor + 1 = 28 + 1 = 29$. Note that it is impossible for 7 candidates to each receive 29 votes since $29 \times 7 = 203$ exceeds the total ballot value.

It's worth noting at this point that any candidate receiving first-place votes from only 29 voters will win a seat. STV is designed to achieve *proportional representation*, meaning that the spectrum of opinions represented by the winners should be in rough proportion to the spectrum of opinions of the voters. Thus, a minority of 29 voters who stand behind a candidate can win a seat in the election.

If all 6 seats are filled after the first round, we are done. If not, we adjust the value of all ballots favoring an elected candidate.

Suppose on the first round that candidate A is favored by 50 ballots, and candidate B is favored by 27 ballots. Then A's vote tally is 50 and B's vote tally is 27. A is elected since $50 \geq 29$ but B is not elected on this round since $27 < 29$.

When A is elected, the ballot value of each of A's 50 supporters is charged $\alpha = 29/50$, so the value of those ballots drops to $1 - 29/50 = 21/50$.

Now suppose that among those 50 ballots favoring A that 5 of them ranked B as their second choice. After A is elected, they each now favor B and have value $21/50$. B's vote total is now increased by $5 \times 21/50 = 21/10$, so B's new vote total becomes $27 + 21/10 = 291/10 > 29$. Hence, B gets elected on the second round. Those 32 second-round B supporters have their values reduced by the factor $(1 - \beta)$, where $\beta = 29/(291/10) = 290/291$. Thus, the former A supporters now have value $(1 - \alpha)(1 - \beta) = (21/50) \times (1/291) = 21/14550 \approx 0.00144330\ldots$, and the original B supporters have value $(1 - \beta) = (1/291) \approx .00343643\ldots$.

After five seats have been filled, the total remaining vote value over all ballots is $200 - 5 * 29 = 55$. As the counting proceeds, the remaining candidates will be eliminated one by one until somebody garners 29 votes. This must happen when only one candidate remains, if not sooner, since in that round, that candidate will have all 55 votes and will be elected.

4 Programming Notes

You should create the following classes:

- `Vote`: A pair consisting of a candidate and an ordinal number representing the rank of that candidate. A vector of votes can be sorted on the rank field to yield a preference-ordered list of candidates.

- `Ballot`: Contains a vector of `Vote` objects in preference order, and a rational number giving the value of the ballot.

- `Candidate`: Information specific to a single candidate during the ballot-counting process. This class contains the candidate's name, a status value showing whether the candidate is still active, elected, or dropped, and other information needed for printing the results such as the round number at which the elected candidate was chosen. It also contains fields used during the election such as a rational number giving the candidate's current tally of votes.

- `Election`: This is the main *controller* class that carries out the vote-counting process. It maintains all of the information describing the election such as the number of seats remaining to be filled, the number of active candidates, a vector of all candidates and a vector of all ballots. It's where the code goes to read the election data from an election data file, to perform the sequence of rounds that determine the election outcome, and to print the results.

  Do not put this code in `main()`. Main should follow our standard template used in previous assignments where it simply calls `run()` in a `try` block. `run()` in turn should just gather the command line arguments, instantiate `Election`, and call a member function to compute and print the election results.

The question always comes up about where to put any given function. The general answer is to put the function in the class that has the information that the function needs to do its job. This is sometimes called the *Expert Principle* of object-oriented design: put the function in the class for which it is the expert.

This principle almost always argues against using so-called "setter" functions, by which I mean a function that sets a private data member to the value of its argument without any further processing or checking. For a setter function not to be misused, caller must have the knowledge of what values are meaningful. But that knowledge belongs in the class with the data member, not outside.

For example, the function to read the file and create the election data structure belongs in the `Election` class since its job is to initialize the election parameters, and that's where they live. On the other hand, the function to add a vote to a candidate's current tally belongs in the candidate class, since the tally field lives in that class, and only the one candidate is affected by that operation.

Sometimes, a task requires expertise that is split between classes. In that case, the high-level expert should *delegate* responsibility for those parts of the task for which another class contains the expertise.

For example, consider the code that transfers excess votes from a newly-elected candidate to the other candidates who end up benefiting from those votes. This task can be broken down into simpler steps:

1. Return the excess votes to the ballots from which they came.

2. Update the favored candidate for those ballots to the most-preferred active candidate.

3. Update the tallies of the newly favored candidates with the ballot's residual value.

Step 1 begins in `Election` since that is where the knowledge is of which candidate was just elected. The job of actually updating the `Ballot` internal data structure should be delegated to a member function of `Ballot`.

Step 2 takes place in `Ballot` since that is where the voter's preference list lives, but of course the job to determine whether a particular candidate is still active is delegated to the class with that information.

Step 3 begins in `Ballot` since that is where the ballot value lives. However, the job of incrementing the favored candidate's tally should be delegated to a member function of `Candidate`.

4.1 Rational Arithmetic

Because ballot values can become tiny fractions (as in the example above), there is an issue of precision and roundoff error in the representation of the ballot values. It's conceivable that two candidates near the end of the election would be so nearly tied that a single ballot with a miniscule residual value could nevertheless swing the election from one candidate to another.

To avoid this problem, we represent the vote values as exact *rational* numbers. The highly optimized Gnu Multiple Precision library (GMP) provides functions for performing arithmetic on arbitrary-length rational numbers. GMP was originally designed for use with C . More recently, C++ extensions have been added that make it much easier for the C++ program to use it for ordinary calculations. However, certain functions (such as printing) are still only available through the C interface, so you will need some knowledge of both interfaces. See the GMP manual at `http://gmplib.org/manual/index.html` for documentation. Pay special attention to section 12, which describes the C++ interface that you will be using. You can also read the manual on the Zoo by typing `info gmp`.

Please feel free to ask for help with any C++ and GMP questions.

5 Expected Output

Your program should produce output similar to the sample output files that I have provided on the Zoo in `/c/cs427/assignments/ps4`.

The first section just prints out the election data from the input file. The second section shows who is elected or dropped at each round, along with their tallies. The final section gives the elected candidates in alphabetical order together with the round number at which they were elected.

You are not required to exactly match my spacing, but otherwise the information should agree.

6 Grading Rubric

Your assignment will be graded according to the scale given in Figure 1.

#	**Pts.**	**Item**
1.	5	A well-formed `Makefile` or `makefile` is submitted that specifies compiler options `-O1 -g -Wall -std=c++11`.
2.	5	Running `make` successfully compiles and links the project and results in an executable file `stv`.
3.	40	Running `stv` on the graders' test files produces correct output or a reasonable error message in case of bad input.
4.	10	Each class and function definition is preceded by a comment that describes clearly what it does.
5.	30	All of the instructions in sections 2 and 4 are followed. Good OO design principles are followed. No global variables. No setters (without convincing justification). No public data members.
6.	10	All relevant requirements from previous problem sets are followed.
	100	Total points.

Figure 1: Grading rubric.