Homework 3 — Election

Due Wednesday, March 6th at 9pm

(100 points)

This is an individual coding assignment. You may have one design buddy, as described below. Having a design buddy is *highly* recommended.

Objectives:

- Develop familiarity with the concept of “static”
- Implement and use the Singleton design pattern
- Design and implement a multi-object program, dealing with the intricacies of objects communicating with one another

Credit:

- `election.zip`
- Your zip file should include your design document.
- Your zip file should include a Makefile. It must compile on the CS VM or on coding.csel.io.

Instructions:

You will write a program that simulates elections. You will use inheritance to implement two kinds of elections: a direct election in which constituents vote directly for candidates and the candidate with the most votes wins (regardless of whether or not that candidate has received a majority of votes), and a representative election in which all representatives from a district vote for the candidate who received the most votes (again, not necessarily a majority) in their district.

You **may** have one design buddy for this assignment. You and your design buddy should figure out which objects are in charge of what, how they will communicate, etc. You and your design buddy cannot write code together. If you have a design buddy (**recommended**), you should indicate this in comments at the top of your main.[cc|cpp] file. Your design document, whether created with a buddy or not, is worth 5 points.

Your design document should be similar to the document that you produced for Homework 2 when diagramming the provided code.

Program Flow:

1) First, you should create an Election or a RepresentativeElection.
2) Next, you should register Candidates for that election.
3) Then, Candidates are allowed to campaign as much as they want. When all Candidates are done campaigning, this step is over.
4) Finally, the Election calls for votes from the Districts.
5) These votes are tallied (differently for Elections vs. RepresentativeElections) and a winner is announced.
6) Return to step 1, using a new Election, but maintaining the same ElectoralMap.

Campaigning and converting constituents:
If a Candidate chooses to campaign in a given District, they can have 1 of the following 4 outcomes:

1) A constituent is converted from Party::None to the Party of the Candidate.
2) A constituent is converted from Party::None to the Party of the Candidate **and** a constituent is converted from the majority Party that is not the Candidate's Party to the Party of the Candidate.
3) A constituent is converted from the majority Party that is not the Candidate's Party to the Party of the Candidate.
4) No one is converted.

To determine which of these situations occurs, the probability of the candidate is calculated according to the following formula:

$$P_{success} = \left(\frac{(constiutents\ from\ the\ Candidate's\ Party + 1) * 2}{constituents\ in\ other\ Parties\ in\ this\ District,\ excluding\ Party::None}\right) * \left(\frac{(constiutents\ from\ the\ Candidate's\ Party + 1) * 2}{area\ of\ this\ District}\right)$$

$$P_{extra\ success} = P_{success} * 0.1$$

$P_{success}$ is the probability of converting a constituent affiliated with Party::None. $P_{extra\ success}$ is the probability of converting a constituent from the majority Party that is not the campaigning Candidate's and is not Party::None to the Candidate's Party. You only need to generate 1 random number. If it fulfills both these categories then two people should be converted. If there are no constituents with the needed affiliation, simply do not convert anyone from that affiliation.

Scenario 1 occurs if $P_{success}$ and not $P_{extra\ success}$. Scenario 2 occurs if $P_{success}$ and $P_{extra\ success}$. Scenario 3 occurs if $P_{success}$ and $P_{extra\ success}$ but there are no constituents associated with Party::None. Scenario 4 occurs if either not $P_{success}$ or if there are no constituents with the needed affiliation.

Creating your ElectoralMap:
Your ElectoralMap may have any number of Districts. This number should be hard coded as a static const int in your ElectoralMap. Your program should work if you change this number for any number of Districts. We recommend starting with 1 or 2 for debugging purposes.

Districts are generated randomly according to the following rules:

- Each Party begins with a random number of constituents between 0 and 9, including Party::None. Generate a random number for each Party.
- Each District is between 5 and 29 square miles, again chosen randomly.
- Each District should be given an id, generated by the ElectoralMap.

When you go from one Election to a new Election, your ElectoralMap must not change. You should generate it once and only once. Your ElectoralMap **must** implement the Singleton design pattern.

Voting:
Constituents vote as follows:

- If the constituent is aligned with Party::None, they vote for the Party that the most constituents in their District is aligned with 70% of the time. If there are multiple Candidates from this Party, a random one is

chosen. The remaining 30% of the time, they vote for a random Candidate not aligned with the District majority Party. If there is only one Candidate or only Candidates from one Party, they vote for that Candidate/Party.

- A constituent aligned with a Party votes for a Candidate aligned with their party 90% of the time (if multiple Candidates aligned with this Party, a random one is chosen). The other 10% of the time they randomly vote for a random Candidate not aligned with their own Party. If there is only one Candidate or only Candidates from one Party, they vote for that Candidate/Party.
- In all cases, if there are multiple candidates from the same Party or with the same number of constituents, a random one is chosen.
- In all cases, constituents **always** cast a vote.
- Constituents aligned with a Party that has no Candidates should act the same as constituents aligned with Party::None.

District votes:

For the RepresentativeElection, there are a total of 5 * the number of Districts votes to be allocated. The number of votes for a given District is calculated according to the formula:

$$Votes_{District} = floor(\left(\frac{\#\ of\ constituents\ in\ this\ District * 1.0}{\#\ of\ constituents\ in\ all\ Districts}\right) * \ total\ number\ of\ district\ votes)$$

If District A has 7 constituents and District B has 5 constituents, District A would end up with 5 votes and District B would end up with 4 votes. Notice that the number of total district votes can be less, but never greater than, the number of total district votes to be allocated.

Name	**Type**	**Purpose**
TextUI	class	Similar to the TextUI from Homework 2, this class should facilitate the prompting of the user for various kinds of information, the routing of that information to the appropriate objects and the display of program data to the user.
Party	enum class	Represents each political party. You may have as many parties as you wish, but you must include `Party::None` and at least one other. Make sure to design your program so that it could support a variable number of parties!
Candidate	struct or class	Candidates have names, party affiliations, and ids. Candidates cannot be affiliated with `Party::None`. Candidates should have ascending ids according to the order in which they are registered.
Election	class	The general manager of elections. Registers candidates, directs them to districts to go campaigning, calls for the actual votes, and reports the winner after tallying the score. At the beginning of an election, a new Election object should be used (as opposed to having some sort of Reset method)

RepresentativeElection	class (inherits Election)	A derived class of Election. Rather than constituents voting directly for candidates, the candidate that wins a majority in each district gets all of that District's votes (similar to with the American electoral college)
District	class	Represents a building block of the ElectoralMap. Districts keep track of their constituents by counting how many constituents are affiliated with each Party. They also handle when constituents change from one party to another. Throughout the course of the program running, the number of total constituents in each District should not change. Districts also have a size in square miles, which affects how easy it is for candidates to campaign there.
ElectoralMap	class (singleton)	Keeps track of Districts, manages the logistics of letting Candidates campaign in Districts, and handles the logic for determining how many votes each District has in a RepresentativeElection. Handles the logic for communicating the votes from each District to the Election. We recommend using a static field to assign ids to each District when they are created so that you can access them with maps from id to district.

Output:
Output files will be posted on the course github on Tuesday, February 25th. You have complete creative freedom in how you design your UI, but it must include the functionality provided in our output files.

Comments and style (15 points):
Your files and functions should have comments. We should know how to run your program and how to use your functions from your comments.

Your variables should have meaningful names. Your code should be easily readable. If you have complex sections of code, you should use inline comments to clarify.

You should follow the conventions set out in our Concise Style Guide, posted on the course github.

Bonus (up to +15 points):
Implement a new type of Election (another subclass of Election) and explain in the comments how constituents vote. More extra credit will be given to more sophisticated voting schemes. "Everyone votes randomly" will receive 1/15 bonus points.