Homework 3 — Election

See Canvas for due date

(125 points)

This is an individual coding assignment. You may have one design buddy, as described below. Having a design buddy is *highly* recommended. You will be attending interview grading for this assignment the week before spring break.

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:

- `hw3_design.pdf` (deliverable 1)
- `Makefile, ElectoralMap.[h,cpp], District.[h, cpp], main.cpp` (deliverable 2)
- `Makefile, all header and implementation files, main.cpp` (deliverable 3)
- `output.txt`

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.

Part 1 (Design Document & getting started):

Design Document (deliverable 1):

You **may** have one design buddy for this assignment. You and your design buddy should figure out which classes are in charge of what (see class description table later in the assignment), how they will communicate (what methods they will have), and what data they will control (what fields will they have). You and your design buddy cannot write code together, but you may discuss what each method should be doing in general.

If you have a design buddy (**recommended**), you should indicate this in comments at the top of your main.cpp file as well as in your design document. Your design document, whether created with a buddy or not, is worth 10 points.

Your design document should be similar to the document that you produced for PE 6 when diagramming the provided code. You do not need to follow a formal diagramming language, but your diagram needs to be legible and communicate the information described above.
It must include:

a) the interfaces that your classes will have (what methods will they have, what methods will they call/use?)
b) the data members (fields/attribute) that your classes will have
c) pseudocode for your main function

Program Flow:

1) First, you should create an Election or a RepresentativeElection based on whether the user chooses "direct" or "representative".
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.

Next, a note on randomness:
You only want to seed your random number generator once each time your program runs. You should not seed your random number inside of a loop, for instance.

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:

$$P_{success} = min(100,\ \left(\frac{(constiutents\ from\ the\ Candidate's\ Party + 1) * 2}{constituents\ in\ other\ Parties\ in\ this\ District,\ excluding\ Party::None}\right) * \left(\frac{(constiutents\ from\ the\ Candidate's\ Party + 1) * 2}{area\ of\ this\ District}\right))$$

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

$P_{success}$ 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:

- A constituent aligned with a Party votes for a Candidate aligned with their party 100% of the time if one exists.
    - if multiple Candidates are aligned with this Party, a random one of these options is chosen.
- If the constituent is aligned with Party::None, they vote for the Party that the most constituents in their District is aligned with 100% of the time.
    - If there are multiple Candidates from this Party, a random one is chosen.
    - If there are no Candidates aligned with the majority Party, they randomly pick a Candidate.
- If there is no one aligned with a constituent's party and the constituent is not aligned with Party::None, they vote as if they were aligned with Party::None.
- In all cases, if there are multiple candidates from the same Party or multiple parties with the same number of constituents, a random choice is made.
- In all cases, constituents **always** cast a vote.

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 PE 6, 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 two others. 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 are on Canvas. You have complete creative freedom in how you design your UI, but it must include the functionality provided in our output files.

Getting Started (deliverable 2):
Note: You may find incorporating std::map into your program very helpful.
1. Write the code for a class ElectoralMap that implements the singleton design pattern. When instantiated, this ElectoralMap should assign unique ids to 4 different Districts and store them in a std::map with int id mapping to District. You may want to use a static field in the District class to help with generating sequential unique ids.
    1. Your District should be a regular class.
    2. You should override the operator<< for both the ElectoralMap and the District.
    3. You should use a static const int field in your ElectoralMap to designate the number of Districts, similar to what we saw in the Earth examples for number of continents.
    4. Each District should start with a random area between 5 and 29 square miles.
    5. You should implement a get_district(int id) method in ElectoralMap that lets you access Districts by id.
    6. You do **not** need to implement any other methods. (Though you will want to for your complete homework!)
    7. Test your code when you change your number of districts to a number other than 4.

2. Write a main.cpp concurrently with step 1 and include adequate code to test your ElectoralMap and District classes.

Part 2 (deliverable 3):
Implement the rest of your program

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.

Simplified Rubric (meant to give you an idea of how points are distributed and general requirements, does not show all details)

Part 1	design document (10 points) Must have plans for the structure, methods, and fields of the following classes: -TextUI	20

	- Election - RepresentativeElection - ElectoralMap - District Must also have pseudocode for your main function <u>code implementation (10 points)</u> - compiles - appropriate ElectoralMap implementation - appropriate District implementation - appropriate main function	
Party		3
Candidate	- Candidates are assigned ascending ids in the order in which they are registered	7
Election	- no candidates from a previous election are automatically registered - can register a candidate for the election. Including the ability to register multiple candidates affiliated with the same party for the election. - correctly reports the winner, counting each constituent vote equally - new Election object used at the beginning of every election	15
RepresentativeElection	- Must inherit Election - must override at least 1 method to conduct the representative election (dynamic dispatch)	10
District	- initialized appropriately - correctly converts constituents	5 (this is for the logic beyond the points for getting started)
ElectoralMap	- correct logic for determining district votes in a representative election - ElectoralMap remains the same from election to election	10 (this is for the logic beyond the points for getting started)
Voting	- this is for the voting logic, no matter which class it is in	10
Other Functionality	- can register candidates - can go campaigning - vote can happen - elections finish with winners - can start another election at the end of one election	30
Style & comments	- follows style guidelines - variables have meaningful names - implementations are in appropriate locations - does not have redundant code/unnecessary conditional branches (continued) - file comments - function comments - inline comments for complex code	15