Team 3

Voting System

Software Design Document

Names: Jacob Malin, Romeo Sweeney, Nathan Kjaer, Shruti Jayavardhanan
Section: 1

Date: 03/01/2024

TABLE OF CONTENTS

1. INTRODUCTION 2
1.1 Purpose 2
1.2 Scope 2
1.3 Overview 2
1.4 Reference Material 2
1.5 Definitions and Acronyms 2

2. SYSTEM OVERVIEW 2

3. SYSTEM ARCHITECTURE 2
3.1 Architectural Design 2
3.2 Decomposition Description 3
3.3 Design Rationale 3

4. DATA DESIGN 3
4.1 Data Description 3
4.2 Data Dictionary 3

5. COMPONENT DESIGN 3

6. HUMAN INTERFACE DESIGN 4
6.1 Overview of User Interface 4
6.2 Screen Images 4
6.3 Screen Objects and Actions 4

7. REQUIREMENTSMATRIX 4

8. APPENDICES 4

1. INTRODUCTION

1.1 Purpose

This SDD document serves the purpose of providing a comprehensive overview of the architecture and the system design of our voting system. The document aims to articulate the key components, interactions, and decision-making processes involved in the development of the voting system. It acts as a reference document for the development team, stakeholders, and any individuals involved in the implementation, maintenance, or evaluation of the system.

1.2 Scope

This document focuses on the design aspects of the voting system, covering both the OPL and CPL voting algorithms. It outlines the system's architecture, important components, data flow, and interactions to facilitate efficient development, testing, and maintenance. This project's primary goals include implementing versatile voting algorithms, ensuring accurate and reliable seat allocation, generating comprehensive audit files, and displaying results effectively. It excludes the ballot collection process. The project aims to enhance the overall efficiency, transparency, and adaptability of the electoral process, offering a valuable tool for election officials and stakeholders involved in democratic systems.

1.3 Overview

This document serves as a guide for the development of the voting system. It includes the necessary diagrams and explanations for the UML class diagram, sequence diagram for OPL elections, and UML activity diagram for CPL election.

Section 2 gives a brief overview of the context of the Voting System. Section 3 describes a high-level overview of the subsystems of our project, and our rationale for the breakdown of responsibility. Section 4 describes how data is created and moved between subsystems. Section 5 provides pseudo-code for all functions. Section 6 describes the user interface and interaction with our system. Section 7 shows the use cases from the SRS map to our classes.

1.4 Reference Material

The documents references within this document are as follows:

- Class Diagram
- OPL Sequence Diagram
- CPL Activity Diagram
- SRS
- Use Case Document

1.5 Definitions and Acronyms

The acronyms used within this document are as follows:

- CPL (Closed Party Listing) - CPL is a voting method in which voters are allowed to vote for a candidate and their affiliated party. The candidate chosen, should their party secure a seat, is the individual with the most votes among their party members.
- OPL (Open Party Listing) - OPL is a voting method in which voters are allowed to vote for a party, with the candidates being chosen in an ordered sequence to win seats.

2. System Overview

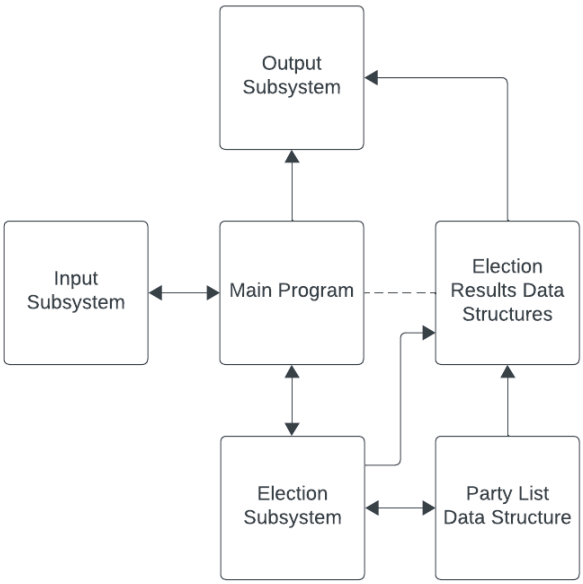
The voting system is designed to facilitate democratic elections, supporting both CPL and OPL algorithms. The system caters to various election scenarios, adapting to the requirements of the officials and the stakeholders. The architecture consists of various subsystems and is designed to facilitate efficient development, testing, and maintenance. It interacts with the ballot collection system through an input file.

3. System Architecture

3.1 Architectural Design

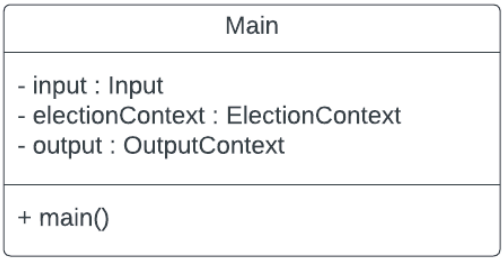
In the diagram below, there are 6 different subsystems that each represent an important component of the overall system. Within the diagram two different lines are used, a dotted line means a basic interaction between subsystems and a solid line with arrows indicates the flow of data between the two subsystems or data structures. The following is a brief description of each subsystem or data structure in the general order that they will be used. The Main Program subsystem is where the main execution of the system will take place, and is the entry point of the entire system. The Input Subsystem handles the input of a filename from either the command line or a text prompt to the user and returns the inputted filename. The Election Subsystem is where the core functionality of the system is located. It is the subsystem that calculates the election results and initializes the Party List Data Structure and some of the Election Results Data Structure. It exchanges data with the Main Program, Party List, and the Election Results Data Structures. The Party List Data Structure is what holds and performs operations on the individual parties and candidates within an election. It interacts with the Election Subsystem and the Election Results Data Structures. The Election Results Data Structures are what holds the final election results, and it is populated by the Election Subsystem and the Party List Data Structures. Once the Election Results are populated, both the Election subsystem and the Party List Data Structure are destroyed. The Election Results data structures is interacted with by the Main Program because the Election Results are passed through the Main System to the Output Subsystem. Lastly, the Output

Subsystem pulls data from the Election Results Data Structures so it can create an audit file and display the results on the screen for the user.

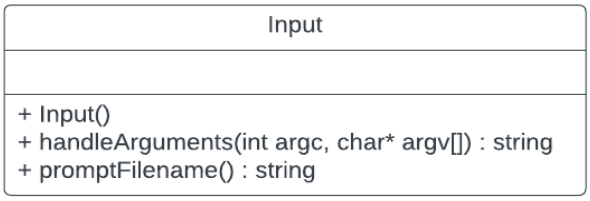


3.2 Decomposition Description

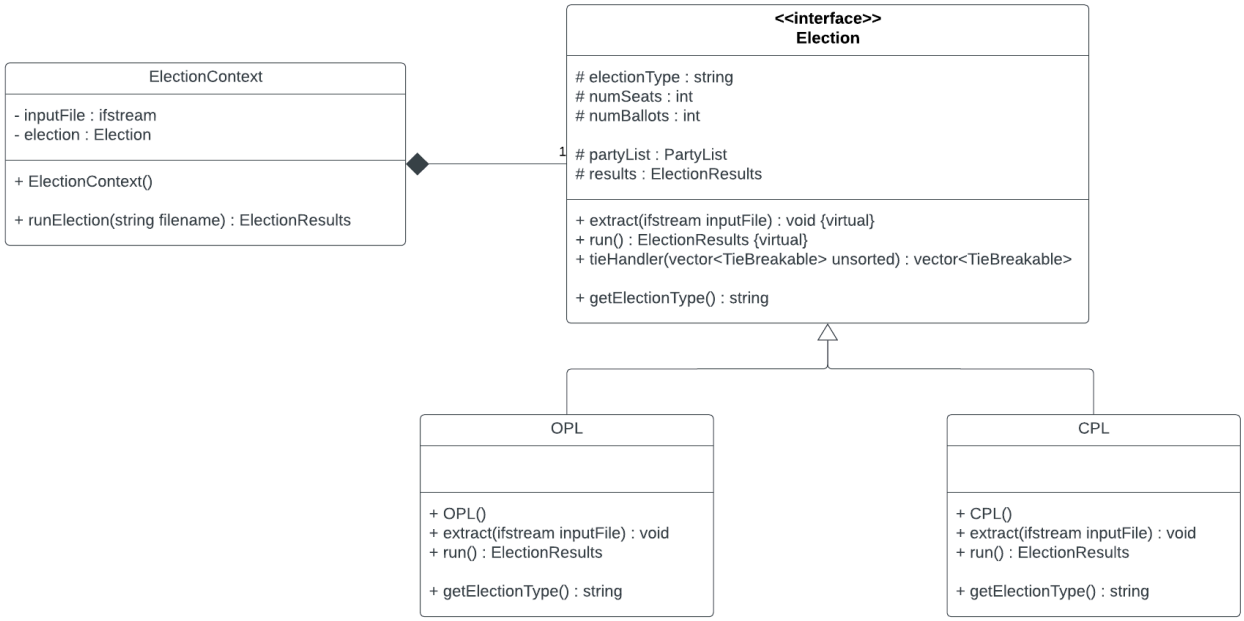
Main Program:



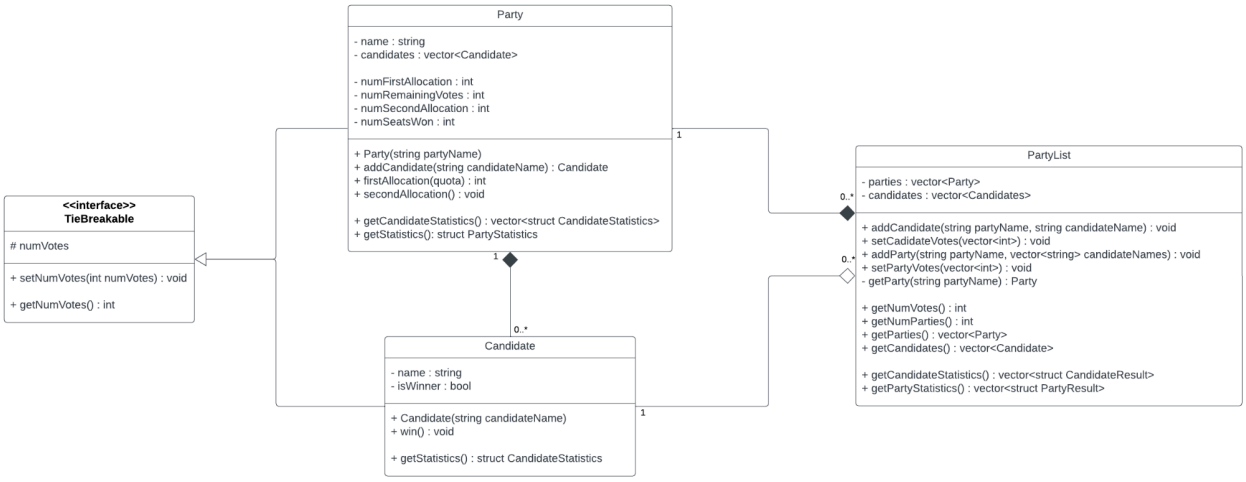
Input Subsystem:



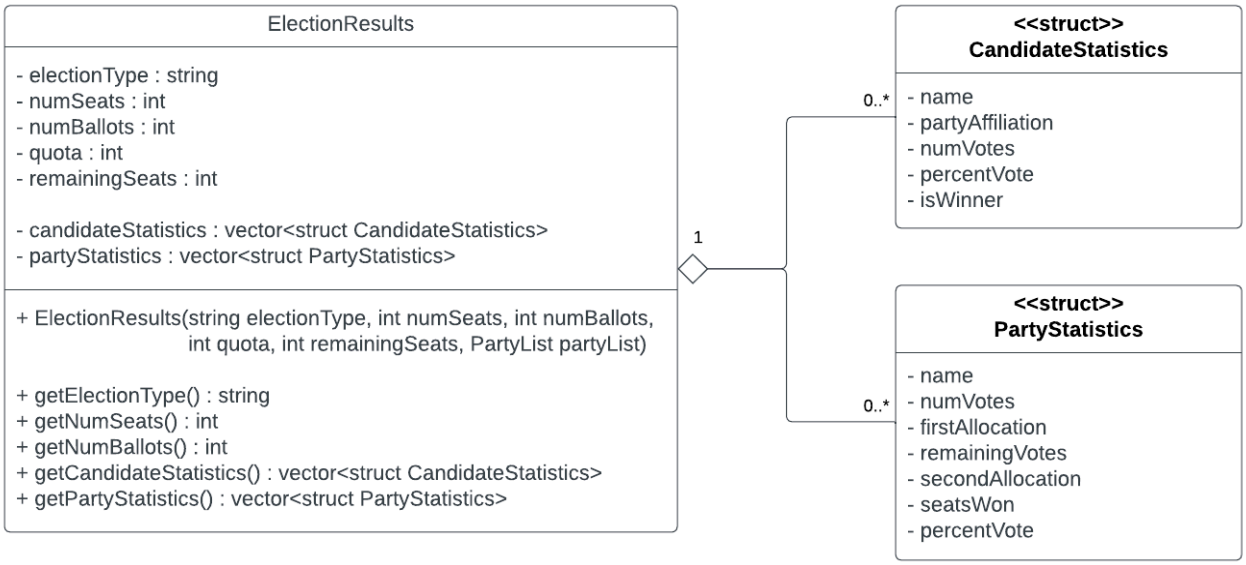
Election Subsystem:



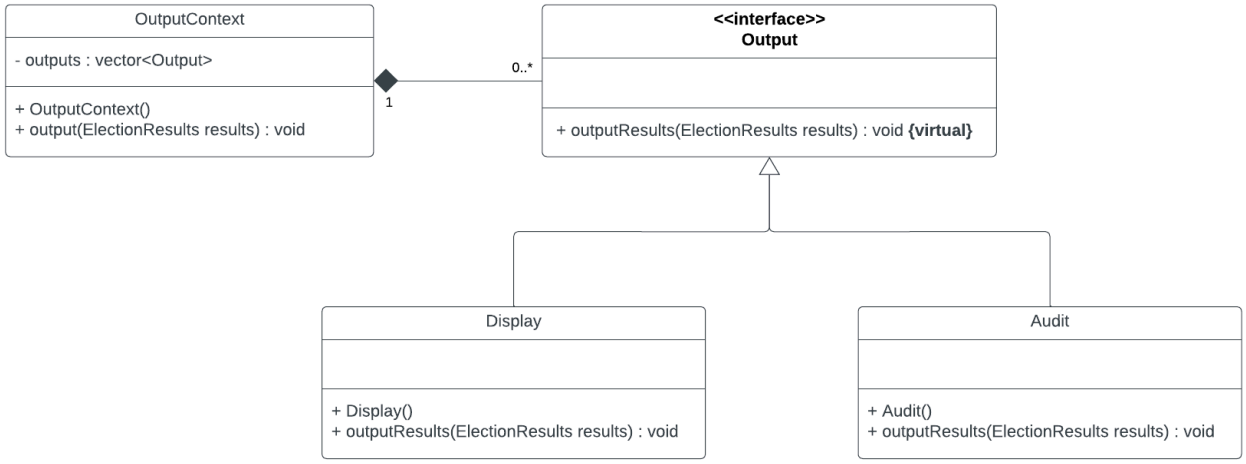
Party and Candidate Data Structures:



Election Results Data Structures:



Output Subsystem:



3.3 Design Rationale

Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that were considered. You may discuss other architectures that were considered, provided that you explain why you didn't choose them.

The primary design rationale for the Main Subsystem was for the main function to have as few function calls as possible, and wherever possible have main only call one function per subsystem.

The Input Subsystem was a grouping of the UC_INPUT_CLI and UC_INTPUT_TEXT because the ideas were similar in that they both were to input a filename.

For the Election Subsystem, OPL and CPL share a common feature in that they are both elections, which was abstracted into the Election interface. Election context was created to create and manage the Election object.

For the Party List Data Structure, a common feature identified between OPL and CPL was the party list, which resulted in the creation of the Party List class, which creates and manages the Party and Candidate objects. It was determined, because of UC_COIN_FLIP, that Party and Candidate classes would need to share some features. This feature is that they can both be tie-broken, which resulted in the TieBreakable interface, which both Party and Candidate inherit from.

For the Election Results Data Structures, it was determined that once the Election Subsystem finished, it would be cumbersome to maintain both the Election Subsystem and the Party List Data Structure as we proceeded into the Output phase. Therefore, the Election Results Data Structure was created to contain all of the information resulting from the election computation. The PartyStatistics struct contains statistics about each party, and the CandidateStatistics struct contains statistics about each candidate.

For the Output Subsystem, in the name of extensibility, an abstract class was created to capture the shared idea of an output method. The OutputContext class was created to manage the list of Outputs, which the OutputContext iterates over to produce the various outputs.

4. DATA DESIGN

4.1 Data Description

Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed, and organized. List any databases or data storage items.

The Election class contains some of the information read in from the file, including election type, number of seats, and number of ballots. The information contained within Election is stored in Election Results once the Election has finished running.

The Party List contains the rest of the information read from the file, including parties, candidates, and their relative numbers. This is maintained in two lists, one containing all of the parties, and one containing all of the candidates. The individual information on each party and candidate is stored within the Party and Candidate classes. The Party List is used by the Election Subsystem and the statistics on the parties and candidates are stored within Election Results once the Election has finished running.

The Party class contains information on a singular party including party name, list of candidates, and number of votes for that party. It also contains the number of seats won, the

first and second allocation of seats, and the number of remaining votes after the first allocation.

The Candidate class contains information about a singular candidate including candidate name, number of votes, and whether or not the candidate has won a seat.

The Election Results contain the results of the Election and statistics about the parties and candidates. The detailed information contained is election type, number of seats, number of ballots, quota, and remaining seats. The statistics are contained within lists of PartyStatistics and CandidateStatistics structs. The election results are obtained from the Election class, and it is passed into the Output Subsystem to be used for output.

There are two statistics structs, the PartyStatistics and the CandidateStatistics. The PartyStatistics struct contains name, num votes, first allocation, remaining votes, second allocation, seats won, and percent vote. The CandidateStatistics struct contains name, party affiliation, number of votes, percent of votes, and if the candidate won a seat.

4.2 Data Dictionary

Main

Data Item Name	Data Type	Description	Return Value (If Applicable)
electionContext	ElectionContext	The ElectionContext object that will control the current election.	N/A
input	Input	The Input object that will handle the command line arguments and retrieve the filename.	N/A
main()	Function	The entry point for the entire system, and where all the system starts executing.	void
output	OutputContext	The OutputContext object that will use Output objects for displaying the results to the screen and creating an audit file.	N/A

Input

Data Item Name	Date Type	Description	Return Value (If Applicable)
handleArguments(int argc, char* argv[])	Function	Takes in an int and char* array, these represent the number of command line arguments and the command line arguments respectively. Returns the	string

		filename retrieved from the command line.	
Input()	Function	The constructor for the Input object, simply creates the object and takes in no arguments.	Input Object
promptFilename()	Function	Prompts the user for a filename input and takes in no parameters. Returns the filename entered by the user.	string

ElectionContext

Data Item Name	Date Type	Description	Return Value (If Applicable)
election	Election	The current Election object that is associated with the ElectionContext.	N/A
ElectionContext()	Function	The constructor for the ElectionContext object. Simply creates the object and takes no inputs.	ElectionContext Object
inputFile	ifstream	The input file stream that will be used to read in all file data.	N/A
runElection(string filename)	Function	The function that will run and calculate the current Election object. It will then return an ElectionResults object when it is done processing all of the election data.	ElectionResults Object

Election Interface
Note: This is an interface from which OPL and CPL objects inherit.

Data Item Name	Date Type	Description	Return Value (If Applicable)
electionType	string	This election object type. This can be OPL or CPL.	N/A
virtual extract(ifstream inputFile)	Function	Extracts all information regarding an election from the input file. As it extracts the information it also populates the data structures accordingly.	void
getElectionType()	Function	This function gets the current election type.	string
numBallots	int	The total number of ballots in the current election.	N/A
numSeats	int	The number of seats available in the current election.	N/A
partyList	PartyList	The PartyList object associated with the current object	N/A

results	ElectionResults	The current elections ElectionResults object.	N/A
virtual run()	Function	Calculates the current election data and populates the results field ElectionResults object.	ElectionResults Object
tieHandler(vector< TieBreakable*> unsorted)	Function	This function takes in a vector of TieBreakable object pointers and returns a tie-broken list of the same objects. This is used to resolve ties within the election.	vector< TieBreakable*>
getElectionType()	Function	This function gets the current election type.	string

CPL

Note: CPL inherits from the Election Interface, all data that is described in the Election Interface is also a part of CPL. To reduce redundancy, only virtual functions from Election are included and described below.

Data Item Name	Date Type	Description	Return Value (If Applicable)
CPL()	Function	Creates and returns the new CPL election object	CPL Object
extract(ifstream inputFile)	Function	Extracts information from the file and stores it in PartyList in CPL format.	void
run()	Function	Runs and calculates an election using CPL rules	ElectionResults

OPL

Note: OPL inherits from the Election Interface, all data that is described in the Election Interface is also a part of OPL. To reduce redundancy, only virtual functions from Election are included and described below.

Data Item Name	Date Type	Description	Return Value (If Applicable)
extract(ifstream inputFile)	Function	Extracts information from the file and stores it in PartyList in OPL format.	void
OPL()	Function	Creates and returns the new OPL election object	OPL Object
run()	Function	Runs and calculates an election using OPL rules	ElectionResults

PartyList

Data Item Name	Date Type	Description	Return Value (If Applicable)
addCandidate(string partyName, string candidateName)	Function	Adds a candidate into the system when they are read in from the file. Also adds the candidate to their associated party immediately.	void
addParty(string partyName, vector<string> candidateNames)	Function	Adds a party to the system when it is read in from the file.	void
candidates	vector<Candidate*>	A vector of all candidates in the current election	N/A
getCandidates()	Function	Retrieves the vector of Candidate objects associated with the election.	vector<Candidate*>
getCandidateStatistics()	Function	Retrieves the CandidateStatistics struct from the candidates' data and combines all structs into a vector which now contains all CandidateStatistics data for the election	vector<struct CandidateStatistics>
getNumParties()	Function	Retrieves the total number of parties in the election.	int
getNumVotes()	Function	Retrieves the total number of votes in the election.	int
getParty(string partyName)	Function	Retrieves a party from the system given a party name.	Party*
getParties()	Function	Retrieves the vector of Party objects associated with the election.	vector<Party*>
getPartyStatistics()	Function	Retrieves the PartyStatistics struct from the candidates' data and combines all structs into a vector which now contains all PartyStatistics data for the election.	vector<struct PartyStatistics>
parties	vector<Party*>	A vector of all parties in the current election	N/A
setCadidateVotes(vector<int> ballots)	Function	Sets all candidates' votes in the election by taking in a vector of all individual candidate votes.	void

setPartyVotes(vector<int> ballots)	Function	Sets all party votes by taking in a vector of individual party votes.	void

TieBreakable Interface
Note: This is an interface from which Party and Candidate inherit.

Data Item Name	Date Type	Description	Return Value (If Applicable)
getNumVotes()	Function	Retrieves the number of votes for the Candidate or Party object.	int
numVotes	int	The number of votes that the Candidate or Party has.	N/A
setNumVotes(int numVotes)	Function	Sets the numVotes field in the object.	void

Party
Note: Party inherits from the TieBreakable interface, all attributes and methods in TieBreakable are also a part of Party. To reduce redundancy, no methods or attributes from TieBreakable were listed as they work the same in both Party and Candidate.

Data Item Name	Date Type	Description	Return Value (If Applicable)
addCandidate(string candidateName)	Function	Adds a candidate to the current Party object	Candidate*
candidates	vector<Candidate*>	A vector of all Candidate objects in the party	N/A
firstAllocation(quota)	Function	Calculates the number of seats won during the first allocation	int
getCandidateStatistics()	Function	Gets the CandidateStatistics structs for all candidates in the current party	vector<struct CandidateStatistics>
getStatistics()	Function	Gets statistics for the Party object	struct PartyStatistics
name	string	Holds the name of the party	N/A
numFirstAllocation	int	Holds the number of seats won in the first allocation	N/A
numRemainingVotes	int	Holds the remaining votes after the first allocation	N/A
numSeatsWon	int	Holds the total number of seats won	N/A

numSecondAllocation	int	Holds the number of seats won after the second allocation	N/A
Party(string partyName)	Function	The constructor for the Party object, takes in the Party name and returns the created Party object	Party Object
secondAllocation()	Function	Adds the appropriate number of seats during the second allocation to the numSeatsWon attribute	void

Candidate

Note: Candidate inherits from the TieBreakable interface, all attributes and methods in TieBreakable are also a part of Candidate. To reduce redundancy, no methods or attributes from TieBreakable were listed as they work the same in both Candidate and Party.

Data Item Name	Date Type	Description	Return Value (If Applicable)
Candidate(string candidateName)	Function	The constructor for the Candidate object,	Candidate Object
getStatistics()	Function	Gets the CandidateStatistics struct for the current Candidate object	struct CandidateStatistics
isWinner	boolean	Holds the value of if the Candidate has won a seat or not	N/A
name	string	Holds the name of the Candidate	N/A
win()	Function	Sets the win field to true because the candidate has won a seat.	void

ElectionResults

Data Item Name	Date Type	Description	Return Value (If Applicable)
candidateStatistics	vector< struct CandidateStatistics>	Holds all CandidateStatistics structs for the election	N/A
ElectionResults(string electionType, int numSeats, int numBallots int quota, int remainingSeats, PartyList partyList)	Function	Constructor for an ElectionResults object, takes in electionType, numSeats, numBallots, quota, remaining seats, and the PartyList object in order to store all information needed about the election	ElectionResults Object
electionType	string	Holds the electionType of the election	N/A

getCandidateStatistics()	Function	Retrieve the vector of CandidateStatistics objects.	vector< struct CandidateStatistics>
getElectionType()	Function	Retrieve the electionType attribute	string
getNumBallots()	Function	Retrieve the stores number of ballots	int
getNumSeats()	Function	Retrieve the total number of seats in the election	int
getPartyStatistics()	Function	Retrieve the vector of PartyStatistics objects.	vector< struct PartyStatistics>
numBallots	int	Holds the number of ballots in the election	N/A
numSeats	int	Holds the total number of seats in the election	N/A
partyStatistics	vector< struct PartyStatistics>	Holds all PartyStatistics structs for the election	N/A
quota	int	Holds the quota used for the first allocation of seats	N/A
remainingSeats	int	Holds the number of seats that were unable to be allocated	N/A

PartyStatistics Struct

Data Item Name	Date Type	Description	Return Value (If Applicable)
firstAllocation	int	Holds the number of seats allocated in the first allocation	N/A
name	string	Holds the name of the Party object	N/A
numVotes	int	Holds the number of votes for the Party object	N/A
percentVote	int	Holds the percentage of the total vote the Party has	N/A
remainingVotes	int	Holds the number of votes remaining after the first allocation	N/A
seatsWon	int	Holds the number of total seats the party won	N/A
secondAllocation	int	Holds the number of seats that the Party won during the second allocation	N/A

CandidateStatistics Struct

Data Item Name	Date Type	Description	Return Value (If Applicable)
isWinner	boolean	Holds if the candidate has won a seat or not	N/A
name	string	Holds the name of the candidate	N/A
numVotes	int	Holds the number of votes cast for the candidate	N/A
partyAffiliation	string	Holds the party the candidate is affiliated with	N/A
percentVote	int	Holds the percentage of the total vote the Candidate has	N/A

OutputContext

Data Item Name	Date Type	Description	Return Value (If Applicable)
output(ElectionResults results)	Function	Calls all Output objects in the outputs vector outputResults function to handle all outputs.	void
OutputContext()	Function	The constructor for the OutputContext object, simply creates and returns the object.	OutputContext Object
outputs	vector<Output>	Contains all desired Output classes, which will each output a different result	N/A

Output Interface
Note: Output is an interface from which Display and Audit inherit.

Data Item Name	Date Type	Description	Return Value (If Applicable)
virtual outputResults(ElectionResults results)	Function	Handles a specific output method that is implemented differently in each class that inherits it.	void

Display
Note: Display inherits from the Output Interface, all methods that are a part of Output are also a part of Display. All virtual methods in Output were overridden in Display so that Display can work correctly.

Data Item Name	Date Type	Description	Return Value (If Applicable)
Display()	Function	The constructor for the Display object, simply creates and returns the object.	Display Object

outputResults(ElectionResults results)	Function	Outputs election results in the correct format to the terminal	void

Audit
Note: Audit inherits from the Output Interface, all methods that are a part of Output are also a part of Audit. All virtual methods in Output were overridden in Audit so that Audit can work correctly.

Data Item Name	Date Type	Description	Return Value (If Applicable)
Audit()	Function	The constructor for the Audit object, simply creates and returns the object.	Audit Object
outputResults(ElectionResults results)	Function	Outputs election results in the correct format to an audit file	void

5. Component Design

- Main Program:
    - Creates an Input instance.
    - Runs handleArguments with the command line arguments, which may or may not succeed.
    - Else, enter a loop that will restart on error:
        - If no filename, run promptFilename.
        - Creates an ElectionContext instance.
        - Calls ElectionContext.runElection(filename) function as a part of the ElectionContext with the filename as an argument, which returns the ElectionResults.
        - Create an OutputContext instance.
        - Call output with the ElectionResults as the argument.
- Input Subsystem:
    - handleArguments
        - If only one argument, extract filename. If the filename is not valid, error.
        - Else, error.
    - promptFilename
        - Prompt user for filename.
        - Read and return the inputted filename.
        - If the filename is not valid, error.
- Election Subsystem:
    - ElectionContext:
        - runElection:
            - Open file using the input filename.
            - Read in election type.
            - Initialize Election object based on election type.
            - Call Election.extract(inputFile).

                - Call Election.run().
        - Election:
            - extract:
                - Create a PartyList object.
                - Extract election information from the input file and store the information in the PartyList.
            - run:
                - Calculate results and update Party List and ElectionResults data structures.
                - Return ElectionResults data structure.
            - tieHandler:
                - Takes in a vector of TieBreakable object pointers.
                - Sorts the unsorted vector based on TieBreakable.getNumVotes().
                - Returns the sorted vector.
            - getElectionType:
                - Gets the type of an Election, either “OPL” or “CPL”.
            - OPL
                - run:
                    - Refer to the Main Course of UC_OPL.
            - CPL
                - run:
                    - Refer to the Main Course of UC_CPL.
- Party List Data Structure:
    - PartyList:
        - addCandidate:
            - Calls getParty(partyName).
            - Calls Party.addCandidate(candidateName).
        - setCandidateVotes:
            - For each candidate:
                - Call Candidate.setNumVotes(numVotes).
        - addParty:
            - Create a Party object.
            - For each candidateName:
                - Call Party.addCandidate(candidateName).
        - setPartyVotes:
            - For each party:
                - Call Party.setNumVotes(numVotes).
        - getParty:
            - If the party does not exist, create the Party object.
        - getPartyStatistics:
            - For each party:
                - Call Party.getStatistics().
                - Populate what is missing.
            - Store the results in a vector of PartyStatistics.
        - getParties:
            - Retrieves the vector of pointers to all Party objects in the PartyList.
        - getCandidates:

                - Retrieves the vector of pointers to all Candidate objects in the PartyList.
            - getCandidateStatitics:
                - For each party:
                    - Call Party.getCandidateStatistics().
                    - For each CandidateStatistics:
                        - Populate what is missing.
                - Store the results in a vector of CandidateStatistics.
            - getNumVotes:
                - For each Party:
                    - Call Party.getNumVotes().
                - Return the total number of votes in the PartyList.
            - getNumParties:
                - Retrieves the total number of parties in the PartyList.
    - Party:
        - addCandidate:
            - Create a Candidate object.
        - firstAllocation:
            - Divide the number of votes received by the quota.
            - Return and store in numSeatsWon whichever is lower, the quotient or the number of candidates.
        - secondAllocation:
            - If the number of candidates is less than the numSeatsWon, increment numSeatsWon and return 1.
            - Else return 0.
        - getCandidateStatitics:
            - For each candidate:
                - Call Candidate.getStatistics().
                - Populate what is missing.
            - Store the results in a vector of CandidateStatistics.
        - getStatistics:
            - Populate a PartyStatistics struct as much as possible.
    - Candidate:
        - win:
            - Set isWinner to true.
        - getStatistics:
            - Populate a CandidateStatistics struct as much as possible.
    - TieBreakable:
        - setNumVotes:
            - Set numVotes.
        - getNumVotes:
            - Return numVotes.
- Election Results Data Structure:
    - ElectionResults
        - getElectionType:
            - Returns election type (CPL or OPL).
        - getNumSeats:
            - Returns the total number of seats in the election.

        - getNumBallots:
            - Returns the total number of ballots in the election.
        - getCandidateStatistics:
            - Returns the vector of CandidateStatistics.
        - getPartyStatistics:
            - Returns the vector of the PartyStatistics.
- Output Subsystem:
    - OutputContext:
        - output:
            - For each Output in outputs:
                - Call Output.outputResults().
    - Output:
        - Audit:
            - outputResults:
                - Write information from ElectionResults to a file.
        - Display:
            - outputResults:
                - Display overall results including seat allocation and party/candidate performance.

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

From a user's perspective, the voting system allows for a simple user interface interaction with the help of two different ways to run an election. The primary way is a text prompt file input wherein a user may input a filename that contains election data. This method is mostly designed for non-technical users such as election officials. The other method is the generic method used by technical users such as programmers and tested, the command line argument. This method does not use any special user interface since it is meant to be a simple and lightweight method to run the program quickly.

Once the program finishes running an election, it will provide a display interface that provides some statistics about the election and general information about the results.

6.2 Screen Images

Text Prompt File Input UI:

```
> Voting System - Version 1.0
> This system processes the winners of the election using either Open
Party Listing or Closed Party Listing Voting.
>
> Please enter the file name of the input file and press the [Enter]
key to submit:
>
```

Terminal UI:

Display Election Results UI:

```
> Voting System - Version 1.0
> Election Results
>
> Election type: <OPL or CPL>
> Number of seats: <number>
> Ballots cast: <number>
>
> Winners: <List of winners>
>     Winner 1 <Party Affiliation>
>     Winner 2 <Party Affiliation>
>
> Candidate Statistics: <An alphabetized list of all candidates>
>     Candidate 1: <number> votes, <percent> of vote, <won/lost>
>     Candidate 2: <number> votes, <percent> of vote, <won/lost>
>     ...
```

6.3 Screen Objects and Actions

None

7. Requirements Matrix

Req. ID	Requirement Description	System Components	Data Structures
UC_INPUT_CLI	A method for testers to use for inputting a file that works from the	Input: handleArguments	string

	command line.		
UC_INPUT_TEXT	A method for users to use when inputting a file that works from inside the program.	Input: promptFilename	string
UC_EXTRACT	Extract info from a given file and store it for later processing.	Election <<Interface>>: extract OPL: extract CPL: extract	PartyList Party Candidate
UC_CPL	A program to evaluate an election based on Closed Party Listing rules.	CPL: run	ElectionResults PartyStatistics <<struct>> CandidateStatistics <<struct>>
UC_OPL	A program to evaluate an election based on Open Party Listing rules.	OPL: run	ElectionResults PartyStatistics <<struct>> CandidateStatistics <<struct>>
UC_AUDIT	The system generates an audit file containing the election details (such as seat allocations, winners, and relevant statistics).	Audit: outputResults	N/A
UC_DISPLAY	The system displays election results on the screen (such as seat allocations, winners, and relevant statistics).	Display: outputResults	N/A
UC_COIN_FLIP	The system provides a mechanism for handling ties via a “coin flip”.	Election <<interface>>: tieHandler TieBreakable <<interface>>: getNumVotes	vector<TieBreakable>
UC_ONE_FOR_ALL	The system processes and manages the results of both OPL and CPL elections.	See Class Diagram for a visual representation of all system components.	See Class Diagram for a visual representation of all system data structures.

8. Appendices

None