

Team 4

# **STV (Droop Quota)/Plurality Voting System**

## *Software Design Document*

### **Names**

Sara Nelson (*nels8907*)

Brendan Ritchie (*ritch167*)

Yiwen Xu (*xu000515*)

Yifan Zhang (*zhan4372*)

### **Date**

03/20/2020

# Table of Contents

<b>Revision History</b>	<b>3</b>
<b>1. Introduction</b>	<b>4</b>
1.1 Purpose	4
1.2 Scope	4
1.3 Overview	4
1.4 Reference Material	5q
1.5 Definitions and Acronyms	5
<b>2. System Overview</b>	<b>5</b>
<b>3. System Architecture</b>	<b>7</b>
3.1 Architectural Design	7
3.2 Decomposition Description	9
3.3 Design Rationale	11
<b>4. Data Design</b>	<b>12</b>
4.1 Data Description	12
4.2 Data Dictionary	12
4.2.1 VotingAPP Class	12
4.2.2 Election Class	13
4.2.3 PluralityElection Class	14
4.2.4 STVElection Class	15
4.2.5 Candidate Class	16
4.2.4 Ballot Class	17
<b>5. Component Design</b>	<b>19</b>
5.1 VotingApp	19
5.2 Election	21
5.3 PluralityElection	22
5.4 STVElection	23
5.5 Candidate	24
5.6 Ballot	25
<b>6. Human Interface Design</b>	<b>26</b>
6.1 Overview of User Interface	26
6.2 Screen Images	26
6.3 Screen Objects and Actions	26
<b>7. Requirements Matrix</b>	<b>27</b>

---

7.1 ID: UC_01	27
7.2 ID: UC_02	27
7.3 ID: UC_03	27
7.4 ID: UC_04	27
7.5 ID: UC_05	28
7.6 ID: UC_06	28
7.7 ID: BR_01	28
7.8 ID: BR_02	28
<b>Appendix A: Droop Quota Algorithm</b>	<b>29</b>
<b>Appendix B: Plurality Voting Algorithm</b>	<b>30</b>

## Revision History

Name	Date	Reason for Change	Version
SDD	1 March 2020	SDD Draft Creation	0.0
SDD	8 March 2020	Add UML and Activity Diagram	0.1
SDD	13 March 2020	Add Sequence Diagram, introduction, system overview, human interface, appendix A and B	0.2
SDD	20 March 2020	Updated existing diagrams and contents, add data design, component design, and requirements matrix	0.3

# 1. Introduction

## 1.1. Purpose

This Software Design Document describes the architecture and system design of the STV/Plurality Voting System.

The expected audience is typical users of the STV/Plurality Voting System. The users will be election officials who need to tally the votes from an election to determine the winner(s) using the voting system. Furthermore, any programmers who might be interested in working on the STV/Plurality Voting System by further developing it or fixing existing bugs may find this document useful.

## 1.2. Scope

This document contains a complete description of the design of STV/Plurality Voting System.

This software will be a voting system that has the option of either running an STV system or a plurality voting system. Users will be collecting ballots from elections and will be using this system to determine the winner of an election based on the desired system selected. The user will be able to input the number of seats that are open during the election, the file names that contain the ballots, and which algorithm to use. The outcome of the software will be a fair evaluation of the ballots to fill the number of seats available in the election.

More specifically, if the STV (Droop) option is selected, the software will provide a method for randomly shuffling the ballots and calculating the droop. It will avoid the early voter bias that can occur with using the Droop method for voting by the random shuffle.

## 1.3. Overview

The remaining chapters and their contents are listed below.

Section 2 is the System Overview that shows the general description of the functionality, context, and design of the system. Background information is also included in this section as well.

Section 3 is the System Architecture including UML activity diagrams, one UML class diagram, and one sequence diagram.

Section 4 is the Data Design that explains how the information domain of your system is transformed into data structures.

Section 5 concerns the Component Design which summarizes each object member function for all the objects.

Section 6 discusses the User Interface Design.

Section 7 contains the Requirement Matrix which provides a cross-reference that traces components and data structures to the requirements in the SRS document

Section 8 is the Appendices.

## 1.4. Reference Material

1. IEEE Recommended Practice for Software Design Descriptions:  
<https://ieeexplore.ieee.org/document/741934>

## 1.5. Definitions and Acronyms

Term	Definition
Droop Quota	The Droop quota is the quota most commonly used in elections held under the single transferable vote (STV) system. It is also sometimes used in elections held under the largest remainder method of party-list proportional representation (list PR). In an STV election, the quota is the minimum number of votes a candidate must receive in order to be elected. Any votes a candidate receives above the quota are transferred to another candidate.
IEEE	Institute of Electrical and Electronic Engineers
Plurality	Plurality voting is an electoral system in which each voter is allowed to vote for only one candidate, and the candidate who polls the most among their counterparts (a plurality) is elected.
STV	The single transferable vote (STV) is a proportional voting system designed to achieve or closely approach proportional representation through voters ranking candidates in multi-seat organizations or constituencies (voting districts).
User	The user of this system is defined as either a software programmer who is doing testing or an election official that is running the software. Further clarification will be described to differentiate between specific users in the text.

Table 1: Glossary

## 2. System Overview

The system will have the following functionalities in it:

- Set voting algorithm: All users can choose plurality or STV algorithm, through a prompt by the system.
- Set number of seats: All users can set the number of seats, through a prompt by the system.
- Choose ballot files: All users can choose which ballot files the election will be based on by entering their file names through a prompt by the system.
- Open help window: All users can open the help window via a text command.
- Turn off shuffle option: Programmers and testers can turn off the shuffle option through a prompt by the system.
- Run voting algorithm: All users can run the voting algorithm based on the selected algorithm and number of seats.
- Display parameters of the election prior to running the election: The system can display the parameters of the election, including the type of voting algorithm, the ballot file names, and the number of seats to all users.
- Display election results summary: The system can display results of the election, including the number of ballots, number of seats, number of candidates, and winners and the losers with pertinent information such as percentage of votes received for plurality or the order of the winners and losers for STV.
- Cancel prior to a voting algorithm running: All users can prevent the voting algorithm from running before an election starts through a text command when the system asks them to confirm the election parameters.
- Display audit file name and location: The system can display the name and the file path of the election audit file of the last election to all users after the voting algorithm has finished running.
- Run test of voting algorithm: Programmers and testers can run the test of the voting algorithm process based on chosen test files, the selected algorithm, number of seats, and the status of the shuffle option.
- Invalid input handler: If users enter invalid parameters for the election or unrecognizable commands, the system will require users to reenter.

The STV/Plurality Voting System will be running on a Linux operating system. The system will have the following design and implementation constraints:

- Multiple runs of the software voting algorithm will not be allowed for the given set of ballots. This is due to the possibility of results change due to the shuffle of the ballots. If the algorithm is run multiple times, the results could possibly change and affect the integrity of the election.
- The STV/ Plurality Voting System is not verified to run on any operating system other than a Linux OS.
- The system will be implemented in C++ coding language and all source files and a Makefile will be provided. The program is expected to be executed from the command prompt.

### 3. System Architecture

#### 3.1. Architectural Design

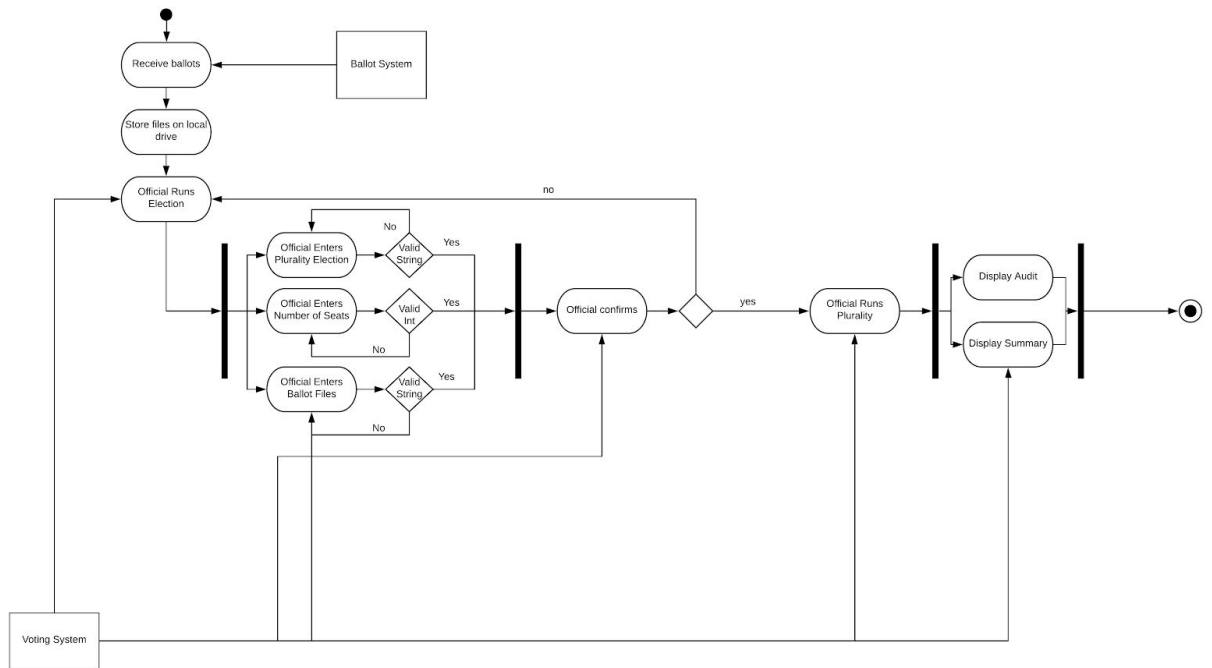


Figure 1: UML Activity Diagram for Plurality Voting Algorithm

This diagram shows the flow of control for the plurality algorithm and refers to the steps involved in the execution of running it.

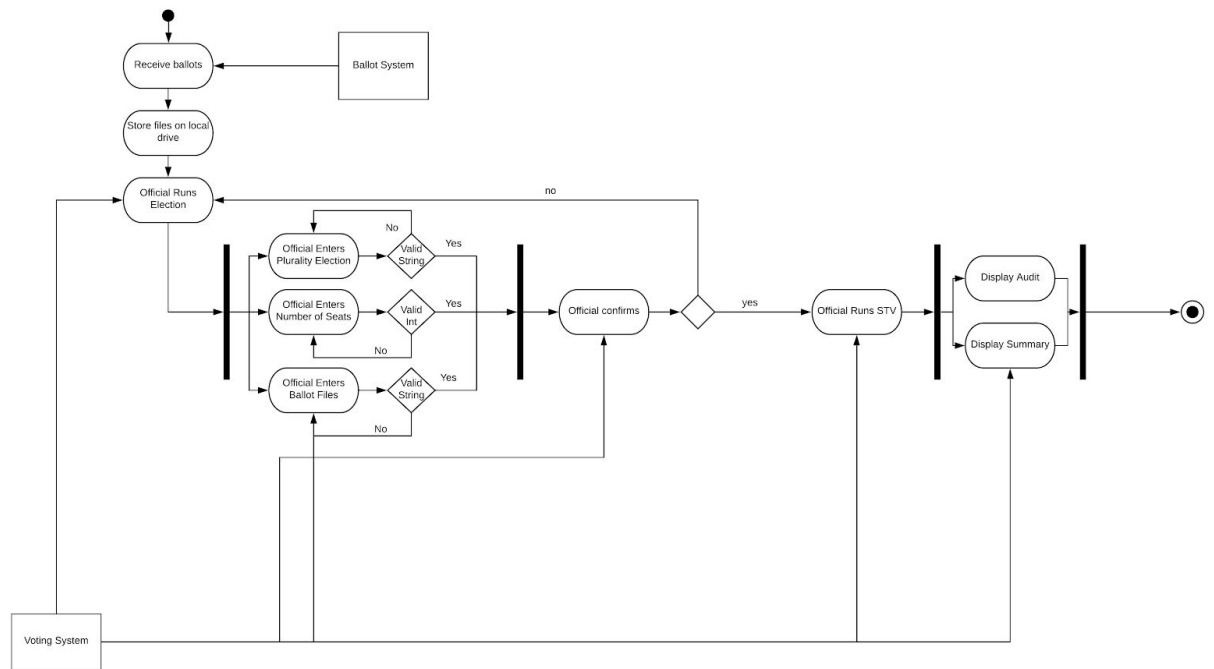


Figure 2: UML Activity Diagram for STV Algorithm

This diagram shows the flow of control for the STV algorithm and refers to the steps involved in the execution of running it.



### 3.2. Decomposition Description

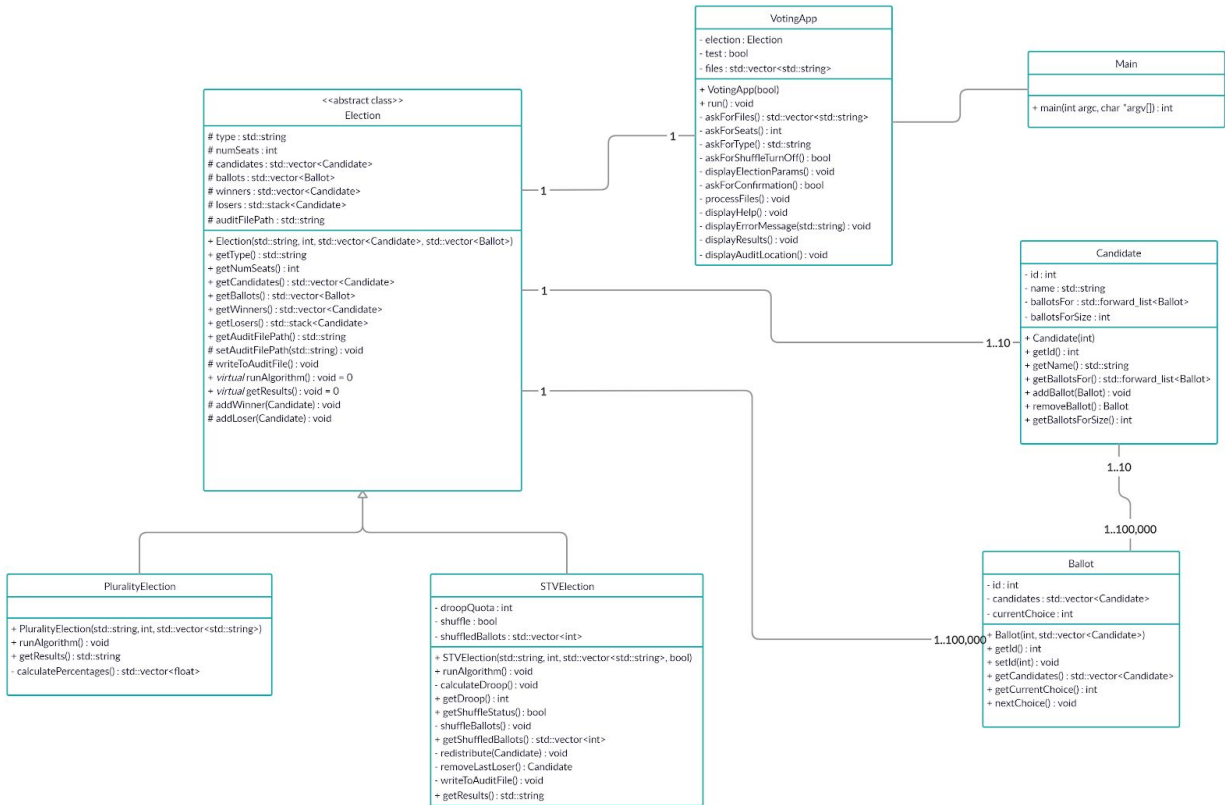


Figure 3: UML Class Diagram

This diagram describes the structure of the voting system by showing the system's classes, their attributes, methods, and the relationships among objects. For more details, refer to Section 4 and Section 5.

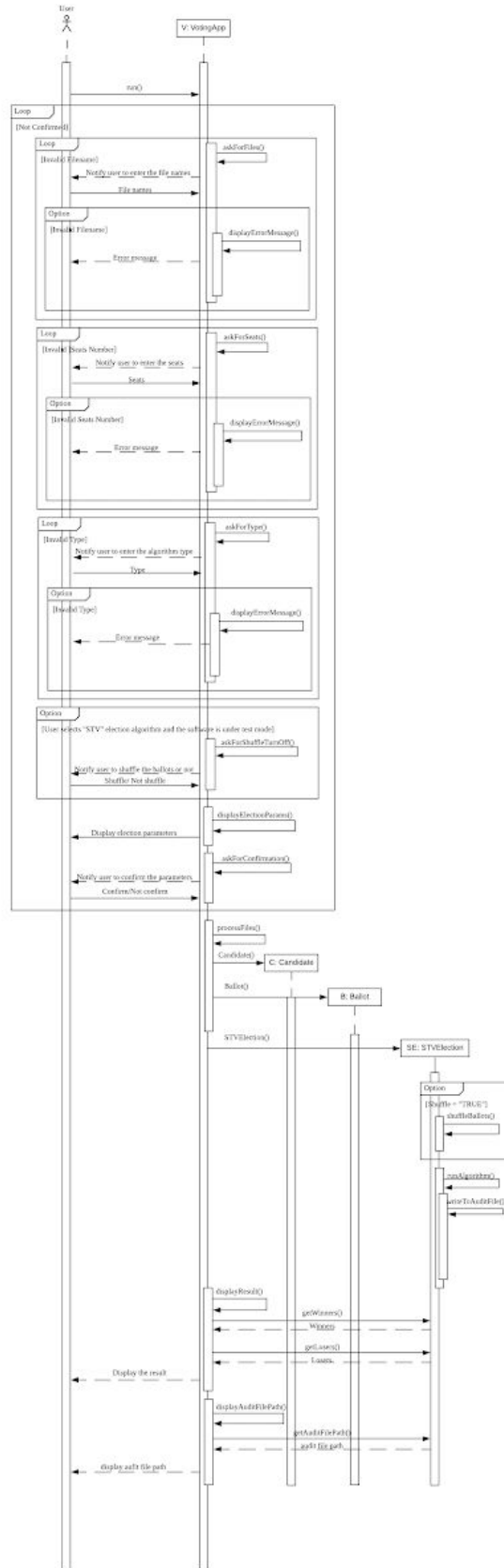


Figure 4: Sequence Diagram of Running the STV Algorithm

1. The user triggers the run method in an instance V of the VotingApp object class from a script from the command line.
2. The method askForFiles in the instance V prompts users to enter the file names and checks the validity of users' input. If the input is invalid, call the method displayErrorMessage and prompt the error message.
3. The method askForSeats in the instance V prompts users to enter the seats and checks the validity of users' input. If the input is invalid, call the method displayErrorMessage and prompt the error message.
4. The method askForType in the instance V prompts users to enter the algorithm type and checks the validity of users' input. If the input is invalid, call the method displayErrorMessage and prompt the error message.
5. If users select "STV" election algorithm and the software is under test mode, the method askForShuffleTurnOff in the instance V is called and prompts users to shuffle the ballots or not.
6. The method displayElectionParam in instance V displays the election parameters to the user.
7. The method askForConfirmation in instance V prompts users to confirm the parameters. If users choose not to confirm it, then return to step 2.
8. The method processFile in instance V calls the constructor of Candidate object class and Ballot object class and creates instances.
9. The instance V calls the constructor of STVElection object class and creates an instance SE.
10. If the value of the shuffle attribute in SE equals "TRUE", the method shuffleBallots in the instance SE is called and the ballots are shuffled.
11. The method runAlgorithm in the instance SE is called and the algorithm is run.
12. The method writeToAuditFile is called in the method runAlgorithm and the audit file is created.
13. After the algorithm is done, the instance V calls the method displayResult, in which the getWinners and getLosers methods in instance SE are called and the result is displayed to the user.
14. The method displayAuditFilePath in the instance V is called, in which method getAuditFilePath in instance SE is called to return the audit file path to the instance V. After that, the audit file path is displayed to the user.

### 3.3. Design Rationale

This system was designed with a few key considerations in mind. Due to simplifying the organization of the software, we decided to have all the ballot processing done in a VotingApp class. In addition, the VotingApp class will be the only class that prints to the screen and requires input from the user. This was determined as a benefit of readability in the software. For determining the data structure for ballots, we were deciding between using queues and linked lists for storing the ballot information. We determined that using forwardly linked lists would be better because it will be easier to shuffle forwardly linked lists rather than queues. We have decided to shuffle ballot indices instead of the ballots themselves to fulfill the user requirement of processing all elections under five minutes. Linked lists will also enable us to keep the order of ballot allocation during redistribution and then we can utilize C++ structures and

not have to manage our own nodes for the ballots. When deciding how many classes we would desire to have, we determined that having a parent Election class would manage the attributes and methods that are needed for both Plurality and STV Election algorithms. In order to test the system, a command-line argument will be passed that will set a flag. This will be processed in `main()` and then passed into the `VotingApp` constructor.

## 4. Data Design

### 4.1. Data Description

The Voting System does not interface with any databases. All files will be located locally on the drive.

### 4.2. Data Dictionary

#### 4.2.1 VotingApp Class

**VotingApp:** `VotingApp` is a class designed for processing input ballot files, creating an instance of `Election` object based on processed data and initializing GUI.

Each `VotingApp` instance contains the following attributes:

- `election`: A private instance of the abstract class `Election`
- `test`: A private boolean variable that shows if the system is under test mode
- `files`: A private vector of strings that stores the names of ballot files

Each `VotingApp` instance has the following methods:

- `VotingApp(bool)`: Public constructor of `VotingApp` class. It takes a boolean value as its parameter and initializes the `test` attribute by the boolean value, and returns nothing.
- `run()`: Public method that is called if the users entered “run”. In this method, the GUI of the system is initialized and displayed to users so that users can interact with the system. This method returns nothing.
- `askForFiles()`: Private method that displays a prompt that requires users to enter names of ballot files separated by white space. If the input is invalid, the method asks users to reenter. This method returns a string.
- `askForSeats()`: Private method that displays a prompt that requires users to enter the number of seats. If the input is invalid, the method asks users to reenter. This method returns an integer.
- `askForType()`: Private method that displays a prompt that requires users to enter the voting algorithm. If the input is invalid, the method asks users to reenter. This method returns a string.

- `askForShuffleTurnOff()`: Private method that can only be called under test mode and displays a prompt that requires users to enter whether they want to turn off the shuffle option. This method returns a boolean variable.
- `displayElectionParams()`: Private method that displays parameters of the election, including the type of voting algorithm, the ballot file names, the number of seats, and the status of the shuffle option if the system is under test mode to all users. This method returns nothing.
- `askForConfirmation()`: Private method that displays a prompt that requires users to enter whether they confirm the parameters of the election. This method returns a boolean variable.
- `processFile(int, std::String, bool)`: Private method that takes the number of seats, the voting algorithm and the status of the shuffle option only if the system is under test mode as parameters, processes ballot files, creates an instance of Election object, and assigns it to its election attribute. This method returns nothing.
- `displayHelp()`: Private method that displays the help window to all users. This method returns nothing.
- `displayErrorMessage(std::String)`: Private method that displays error messages that indicate users' input is invalid and the reason for it to all users, according to the string parameter. This method returns nothing.
- `displayResult()`: Private method that displays the result of the election, including the number of ballots, number of seats, number of candidates, winners, and the losers with pertinent information such as percentage of votes received for plurality or the order of the winners and losers for STV. This method returns nothing.
- `displayAuditLocation()`: Private method that displays the locations of audit files to all users. This method returns nothing.

#### 4.2.2 Election Class

Election: Election is an abstract class designed for storing information of an election such as the voting algorithm of the election, number of seats, all candidates with their ballots, all ballots with their chosen candidates, winners of the election, losers of the election, and the file paths of all audit files. This method also runs the election based on users' chosen voting algorithm and writes the result with details to audit files.

Each Election instance contains the following attributes:

- `type`: A protected string variable that contains users' chosen voting algorithm.
- `numSeats`: A protected integer variable that contains the number of seats.
- `candidates`: A protected vector of Candidate objects, which contains all candidates.
- `ballots`: A protected vector of Ballot objects, which contains all ballots.

- winners: A protected vector of Candidate objects, which contains all winners of the election.
- losers: A protected stack of Candidate objects, which contains all losers of the election.
- auditFilePath: A protected string that contains the file paths of all audit files.
- All these protected attributes can be accessed by its derived class.

Each Election instance has the following methods:

- Election(std:: String, int, std:: Vector<Candidate>, std::vector<Ballot>): Public constructor of Election class. It initializes its type by the String parameter, numSeats by the Integer parameter, candidates by the Vector of Candidate objects parameter, and ballots by the Vector of Ballot objects parameter. This method returns nothing.
- getType(): Public method that returns the type attribute which is a String variable.
- getNumSeats(): Public method that returns the numSeats attribute which is an Integer variable.
- getCandidate(): Public method that returns the candidates attribute which is a Vector of Candidate objects.
- getBallots(): Public method that returns the ballots attribute which is a Vector of Ballot objects.
- getWinners(): Public method that returns the winners attribute which is a Vector of Candidate objects.
- getLosers(): Public method that returns the losers attribute which is a Vector of Candidate objects.
- getAuditFilePath(): Public method that returns the file paths of all audit files, which is a String variable.
- setAuditFilePath(std: String): Protected method that takes a String parameter and assigns it to auditFilePath attribute. This method returns nothing.
- writeToAuditFile(): Protected method that writes the detailed result of the election to audit files. This method returns nothing.
- runAlgorithm(): Pure virtual function for running the election based on users' chosen voting algorithm. This method is implemented in derived classes.
- getResults(): Pure virtual function for returning the result of the election. This method is implemented in derived classes.
- addWinner(Candidate): Protected method that takes an instance of Candidate object and appends it to the winners attribute. This method returns nothing.
- addLoser(Candidate): Protected method that takes an instance of Candidate object and pushes it to the losers attribute. This method returns nothing.
- All protected methods can be accessed by its derived classes.

### 4.2.3 PluralityElection Class

**PluralityElection:** PluralityElection is a derived class that inherits from the Election class. This class is designed for running the plurality algorithm, getting the result and calculating the percentage of ballots that each candidate gets.

Each PluralityElection instance has the following methods:

- **PluralityElection(std::string, int, std::vector<Candidate>, std::vector<Ballot>):** Public constructor of PluralityElection class. It takes a string value, an integer value, a vector of Candidate objects and a vector of Ballot objects as its parameters and initializes its type attribute by the integer parameter, numSeats by integer parameter, candidates by the vector of Candidate objects parameter and ballots by the vector of Ballot objects parameter. This method returns nothing.
- **runAlgorithm():** Public method that runs the election based on the plurality algorithm. This method returns nothing.
- **getResult():** Public method that gets the result of running the plurality algorithm. This method returns a string variable that contains the results of the plurality election, as well as the Candidates and their vote percentage. Also, the string will contain who won and who lost. This method returns a string variable.
- **calculatePercentage():** Private method that gets the percentage of the ballots that each candidate gets. This method returns a vector of float values.

### 4.2.4 STVElection Class

**STVElection:** STVElection is a derived class that inherits from Election class. This class is designed for running the election based on the STV voting algorithm, and shuffling ballots. It also stores unique information for the STV election such as droop quota, the status of the shuffle option, and the shuffled ballots.

Each STVElection instance contains the following attributes:

- **droopQuota:** A private Integer variable that stores droop quota.
- **shuffle:** A private boolean variable that stores the status of the shuffle option.
- **shuffledBallots:** A private Vector of Integers that stores the shuffled indices of the ballots attribute.

Each STVElection instance has the following methods:

- **STVElection(std::string, int, std::vector<Candidate>, std::vector<Ballot>, bool):** Public constructor of STVElection class. It takes a string value, an integer value, a vector of Candidate objects, a vector of Ballot objects and a boolean value as its parameters and initializes its type attribute by the integer parameter, numSeats by integer parameter, candidates by the vector of Candidate objects parameter,

ballots by the vector of Ballot objects parameter and shuffle by the boolean parameter. This method returns nothing.

- `runAlgorithm()`: Public method that runs the election based on the STV algorithm. This method returns nothing.
- `calculateDroop()`: Private method that calculates droop quota based on the formula in Appendix A. This method returns nothing.
- `getDroop()`: Public method that returns the value of the `droopQuota` attribute. This method returns an integer variable.
- `getShuffleStatus()`: Public method that returns the shuffle attribute. This method returns a boolean variable.
- `shuffleBallots()`: Private method that shuffles the indices of the ballots attribute and assigns the shuffled indices to the `shuffledBallots` attribute. The method returns nothing.
- `getShuffledBallots()`: Public method that returns the vector of indexes of shuffled ballots stored in the `shuffledBallots` attribute. This method returns a vector of integer values.
- `redistribute(Candidate)`: Private method that takes an instance of Candidate object and redistributes the ballots of passed Candidate object to other candidates. This method returns nothing.
- `removeLastLoser()`: Private method that removes the last loser which is at the top of the loser Candidate stack. This method returns the Candidate object that has been removed.
- `writeToAuditFile()`: Protected method that writes the detailed result of the election to audit files. This method returns nothing.
- `getResult()`: Public method that returns the result of running the STV algorithm. This method returns a string that describes the winning Candidates and what order they won and the losing Candidates and what order they lost in.

#### 4.2.5 Candidate Class

**Candidate:** Candidate is a class designed for storing the candidate information such as the id, name, the ballots that have been distributed to the candidate and the size of it.

Each Candidate instance contains the following attributes:

- `id`: A private integer variable that stores the id of the candidate.
- `name`: A private string variable that stores the name of the candidate.
- `ballotsFor`: A private forward list of Ballot that stores the ballots that have been distributed to the candidate.
- `ballotsForSize`: A private integer variable that stores the number of ballots that has been distributed to the candidate.

Each Candidate instance has the following methods:



- `Candidate(int)`: Public constructor of Candidate class. It takes an integer value as its parameter and initializes id attribute by the integer value, and returns nothing.
- `getId()`: Public method that returns the id of the candidate. This method returns an integer variable.
- `getName()`: Public method that returns the name of the candidate. This method returns a string variable.
- `getBallotsFor()`: Public method that returns all the ballots that have been distributed to the candidate. This method returns a forward list of Ballot objects
- `addBallot(Ballot)`: Public method that takes an instance of Ballot object as its parameter and adds this instance to the forward list ballotsFor. This method returns nothing.
- `removeBallot()`: Public method that removes the instance of Ballot object from the forward list ballotsFor. This method returns the Ballot object that has been removed.
- `getBallotsForSize()`: Public method that returns ballotsForSize attribute which is the number of ballots that has been distributed to the candidate. This method returns an integer variable.

#### 4.2.4 Ballot Class

Ballot: Ballot is a class designed for storing the ballot information such as the id of the ballot, candidates were chosen, and the id of the Candidate object that the ballot currently belongs to.

Each Ballot instance contains the following attributes:

- `id`: A private integer variable that stores the id of the ballot.
- `candidates`: A private vector of Candidate objects that stores the chosen candidates in order.
- `currentChoice`: A private integer variable that stores the id of the Candidate object that the ballot currently belongs to.

Each Ballot instance has the following methods:

- `Ballot(int, std::vector<Candidate>)`: Public constructor of Ballot class. It takes an integer value and a vector of Candidate objects as its parameters and initializes id attribute, candidates attributes by the integer value and vector of Candidate objects respectively and returns nothing.
- `getId()`: Public method that gets the id of the ballot. This method returns an integer variable.
- `setId(int)`: Public method that sets the id of the ballot. It takes an integer value as its parameter and assigns it to the id attribute. This method returns nothing.
- `getCandidates()`: Public method that returns the value of the candidate attribute of the ballot. This method returns the vector of Candidate.

- `getCurrentChoice()`: Public method that returns the integer value that stores the id of the Candidate instance that the ballot currently belongs to. This method returns an integer variable.
- `nextChoice()`: Public method that sets the `currentChoice` to the next candidate in order. This method returns nothing.

Class Name	Attribute Name	Attribute Type	Attribute Description
Ballot	<code>candidates</code>	<code>std::vector&lt;Candidate&gt;</code>	a private vector of Candidate that stores the Candidate in order.
	<code>currentChoice</code>	<code>int</code>	a private integer variable that stores the id of the Candidate instance that the ballot currently belongs to
	<code>id</code>	<code>int</code>	a private integer variable that stores the id of the ballot
Candidate	<code>ballotsFor</code>	<code>std::forward_list&lt;Ballot&gt;</code>	a private forward list of Ballot that stores the ballots that has been distributed to the Candidate instance.
	<code>ballotsForSize</code>	<code>int</code>	a private integer variable that stores the number of ballots that has been distributed to the Candidate instance.
	<code>id</code>	<code>int</code>	a private integer variable that stores the id of the candidate
	<code>name</code>	<code>std::string</code>	a private string variable that stores the name of the candidate
Election	<code>auditFilePath</code>	<code>std::string</code>	a protected string that contains the file paths of all audit files.
	<code>ballots</code>	<code>std::vector&lt;Candidate&gt;</code>	a protected vector of Ballot objects, which contains all ballots
	<code>candidates</code>	<code>std::vector&lt;Candidate&gt;</code>	a protected vector of Candidate objects, which contains all candidates.
	<code>losers</code>	<code>std::stack&lt;Candidate&gt;</code>	a protected stack of Candidate objects, which contains all losers of the election.

	numSeats	int	a protected integer variable that contains the number of seats.
	type	std::string	a protected string variable that contains users' chosen voting algorithm
	winners	std::vector<Candidate>	a protected vector of Candidate objects, which contains all winners of the election.
STVElection	droopQuota	int	A private int that contains the droop value calculated in the calculateDroop() method
	shuffle	bool	A private boolean containing the shuffle status from the getShuffleStatus() method.
	shuffleBallots	std::vector<int>	A private vector of int provided the index of each shuffled ballot
VotingApp	election	Election	a private instance of the abstract class Election
	files	std::vector<std::string>	a private string that stores the name of ballot files
	test	bool	a private string that stores the name of ballot files

Table 2: Alphabetical List of the System Data

## 5. Component Design

### 5.1 VotingApp

- VotingApp(test\_mode : bool) {
  - // set test member variable to test\_mode parameter
- }
- run() {
  - // while the user hasn't confirmed the parameters
  - // ask for the number of seats, type, and ballot files
  - // process those files

---

```

        // create an appropriate Election and set it to the election member variable
        // run the election
        // display the results of the election
        // display the location of the audit file
    }
    • askForFiles() {
        // ask for the names of ballot files
        // confirm that the user didn't enter erroneous input
        // if they did ask them for input again
        // return a vector of strings which are the names of the files
    }
    • askForSeats() {
        // ask for the number of seats in the election
        // confirm that the user entered a positive integer greater than 0
        // if they didn't ask them for input again
        // return the number of seats
    }
    • askForType() {
        // ask for the election type
        // confirm that the user entered stv or plurality
        // if they didn't ask them for input again
        // return the election type
    }
    • askForShuffleTurnOff() {
        // ask if the user wants to turn of the shuffling of ballots for the stv election
        // confirm the user entered yes or no
        // if they didn't ask them for input again
        // return True or False depending on their answer
    }
    • displayElectionParams() {
        // print out the parameters of the election that the user input to the interface
    }
    • askForConfirmation() {
        // ask the user to confirm the parameters of the election
        // confirm they entered yes or no
        // if they didn't ask again
        // if yes, return True
        // if no, return False
    }
    • processFiles(seats : int, type : std::string, shuffle : bool) {
        // process the first line of first file to create Candidates
        // process the remainder of the file and the rest of the files to create Ballots
        // create appropriate Election object with the Candidates, Ballots, seats, and shuffle and

```

```

        // store it in the election member variable
    }
    • displayHelp() {
        // print out the help message to the interface
    }
    • displayErrorMessage(message : std::string) {
        // print out the message indicating what the error was to the interface
    }
    • displayResults() {
        // print out the results of the election to the interface
    }
    • displayAuditLocation() {
        // print out the location of the audit file to the interface
    }

```

## 5.2 Election

```

    • Election(type : std::string, seats : int, cands : std::vector<Candidate>, bals : std::vector<Ballot>) {
        // store the type, seats, cands, and bals parameters in the appropriate member variables of
        // the Election class and set the default for the other member variables
    }
    • getType() {
        // return the type of the election
    }
    • getNumSeats() {
        // return the number of seats of the election
    }
    • getCandidates() {
        // return the vector of Candidates in the election
    }
    • getBallots() {
        // return the vector of Ballots in the election
    }
    • getWinners() {
        // return the vector of winning Candidates in the election
    }
    • getLosers() {
        // return the stack of losing Candidates in the election
    }
    • getAuditFilePath() {
        // return the name of the audit file path
    }

```

- `setAuditFilePath(name : std::string) {`  
     `// set the name of the audit file path to the auditFilePath member variable`  
   `}`
- `writeToAuditFile() {`  
     `// write to the audit file the details of the election at that point in time`  
   `}`
- `runAlgorithm() {`  
     `// pure virtual function implemented in the child classes of the Election class`  
   `}`
- `getResults() {`  
     `// pure virtual function implemented in the child classes of the Election class`  
   `}`
- `addWinner(win : Candidate) {`  
     `// add the winner to the end of the vector of winning Candidates`  
   `}`
- `addLoser(lose : Candidate) {`  
     `// add the loser to the top of the stack of losing Candidates`  
   `}`

### 5.3 PluralityElection

- `PluralityElection(type : std::string, seats : int, cand : std::vector<Candidate>, bals : std::vector<Ballot>) {`  
     `// utilize the constructor of the parent Election class to store the type, seats, cand, and`  
     `// bals parameters in the appropriate member variables and set the default for the other`  
     `// member variables`  
   `}`
- `runAlgorithm() {`  
     `// tally up the Ballots in the Ballot vector`  
     `// Candidates are assigned Ballots based on who is voted for on the Ballots`  
     `// Candidates who receive the most votes are awarded seats`  
     `// Candidates awarded seats are put in the vector of Winners`  
     `// Candidates not awarded seats are put in the stack of Losers`  
     `// if a tie, randomly determine who is a winner and who is a loser`  
     `// The audit file is being written to this whole time`  
     `// Write happens at beginning to record initial state, then happens at the end with the final`  
     `// results`  
   `}`
- `getResults() {`  
     `// create a string to contain the results of the plurality election`  
     `// the string will contain the Candidates and their vote percentages`  
     `// the string will also contain who won and who lost`  
   `}`

- ```

        // this string is returned
    }
    • calculatePercentages() {
        // the Candidates are looped over to determine what percentage of the vote they received
        // these percentages are calculated and stored in a vector of floats
        // this vector of floats is returned
    }

```

## 5.4 STVElection

- ```

    • STVElection(type : std::string, seats : int, cand : std::vector<Candidate>, bals :
      std::vector<Ballot>, shuffle : bool) {
        // utilize the construction of the parent Election class to store the type, seats, candidates,
        // ballots, and shuffle parameters in the appropriate member variables and set the default
        // for the other member variables
    }
    • runAlgorithm() {
        // While not all ballots have been processed
        // go through ballots vector and distribute ballots based on their top choice
        // if any candidate hits droopQuota value, declare candidate a winner
        // remove ballots of any winner from being redistributed
        // after first run through of ballots, declare the candidate with fewest ballots as loser
        // if a tie, determine loser based on who got first ballot last, or randomly declare loser
        // redistribute the ballots of the loser candidate
        // continue this redistribution process until every Candidate is assigned to the winner
        // vector or loser stack
        // if all the seats are filled by candidates in the winners vector, then done
        // if not all seats are filled then pop candidates off the top loser stack to fill the remaining
        // seats and add them to the winner vector, then done
        // audit file is being written to this whole time, written to at start for initial state, then
        // written to after the first Ballot distribution, then written to after each redistribution of
        // Ballots, then finally written to at the end once all the winners and losers have been
        // determined. The final results are written to the audit file and it is closed.
    }
    • calculateDroop() {
        // Tally up the ballots in the ballot vector
        // Get candidate number from Candidate vector
        // Divide ballot number by (candidate number + 1)
        // Add 1 to value, store value in the droopQuota member variable.
    }
    • getDroop() {

```

```

    // Return the int that is stored in the droopQuota member variable
}
• getShuffleStatus() {
    // Return bool value that is stored in the shuffle member variable
}
• shuffleBallots() {
    // seed the random algorithm with the system time
    // run the random algorithm 100 times to guarantee randomness
    // randomly shuffle the ints in the shuffledBallots vector which will then be used to index
    // into the vector of Ballots
}
• getShuffledBallots() {
    // return the vector of indexes of shuffled ballots stored in the shuffleBallots member
    // variable
}
• redistribute(loser : Candidate) {
    // Take ballots assigned to the loser Candidate and redistribute them to the other
    // Candidates based on the next choice on the Ballots
    // redistribution of ballots is done in the same order the Ballots were assigned to the
    // losing Candidate to maintain integrity
}
• removeLastLoser() {
    // Remove candidate at the top of the loser Candidate stack
    // Return that Candidate
}
• writeToAuditFile() {
    // write to the audit file the details of the STV election at that point in time
}
• getResults() {
    // Create a string to contain the results of the STV election
    // The string will contain the winning Candidates and what order they won in
    // The string will also contain the losing Candidates and what order they lost in
    // This string is returned
}

```

## 5.5 Candidate

- `Candidate(id : int, name : std::string) {`  
    `// set the id and name of the candidate to the parameter values and set the default for the`  
    `// rest of the member variables`  
`}`
- `getId() {`



---

```

        // return the id of the Candidate
    }
    • getName() {
        // return the name of the Candidate
    }
    • getBallotsFor() {
        // return the forward_list of the Ballots for the Candidate
    }
    • addBallot(Ballot) {
        // add a Ballot to the end of the forward_list of Ballots for the Candidate
        // increment the ballotsForSize variable by 1
    }
    • removeBallot() {
        // decrement the ballotsForSize variable by 1
        // remove a Ballot from the front of the forward_list of Ballots for the Candidate and
        // return that Ballot
    }
    • getBallotsForSize() {
        // return the ballotsForSize variable which holds the size of the ballotsFor forward_list
    }

```

## 5.6 Ballot

```

    • Ballot(id : int, choices : std::vector<Candidate>) {
        // set the id and candidates of the Ballot to the parameter values and set the default for the
        // rest of the member variables
    }
    • getId() {
        // return the id of the Ballot
    }
    • setId(id : int) {
        // set the id of the ballot to the parameter value
    }
    • getCandidates() {
        // return the vector of Candidates for this Ballot
    }
    • getCurrentChoice() {
        // return the currentChoice member variable value which is an index of the candidates
        // vector
    }
    • nextChoice() {
        // increment the currentChoice variable value by 1
    }

```

```

// if currentChoice > size of candidates vector then set currentChoice to -1 (this means the
// Ballot is invalid and needs to be thrown out during redistribution)
}

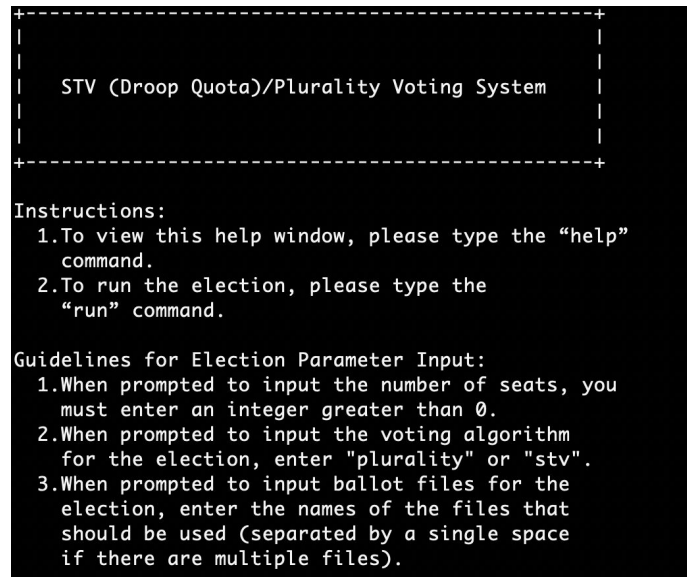
```

## 6. Human Interface Design

### 6.1. Overview of User Interface

There will be a text-based user interface. The software will prompt the user for inputs and the user will use a standard keyboard and mouse to type. For the file inputs, the user will place all desired ballot files in the same directory as the voting system software. It is assumed that the user will know the location of the voting software on their machine and can navigate to the directory. The user will have access to a help window that is displayed below that will guide them through the flow of the voting system.

### 6.2. Screen Images



```

+-----+
| STV (Droop Quota)/Plurality Voting System |
+-----+
Instructions:
1.To view this help window, please type the "help"
  command.
2.To run the election, please type the
  "run" command.
Guidelines for Election Parameter Input:
1.When prompted to input the number of seats, you
  must enter an integer greater than 0.
2.When prompted to input the voting algorithm
  for the election, enter "plurality" or "stv".
3.When prompted to input ballot files for the
  election, enter the names of the files that
  should be used (separated by a single space
  if there are multiple files).

```

Figure 5: Help Window

More screen images will be available once the system has been coded and implemented. The user interface will look exactly like a terminal

### 6.3. Screen Objects and Actions

There is no screen object in this system. The actions are done through the command line like GUI.

## 7. Requirements Matrix

### 7.1. ID: UC\_01

**Description:** User Enters Number of Seats

**Requirement Type:** Functional

**System:**

This requirement will be fulfilled in the VotingApp class as the user is prompted to input the number of seats for the current election. This can be referred to under the askForSeats() method.

### 7.2. ID: UC\_02

**Description:** User Selects Voting Algorithm

**Requirement Type:** Functional

**System:**

This requirement is fulfilled in the VotingApp class as the user is prompted for the type of algorithm that should be run. The user will input a string indicating either STV or Plurality voting algorithms and this is referred to under the askForType() method.

### 7.3. ID: UC\_03

**Description:** User Enters Ballot Files

**Requirement Type:** Functional

**System:**

This requirement is fulfilled in the VotingApp class as the user is prompted for the ballot files to be processed. The user will be prompted for the ballot names of the ballot files and the user will input a string. This is referred to under the askForFiles() method.

### 7.4. ID: UC\_04

**Description:** User Runs Voting Algorithm

**Requirement Type:** Functional

**System:**

This requirement is fulfilled in the VotingApp class as the user is prompted to confirm the input parameters of the voting system. The user will input either “yes” or “no” for confirmation. Once the parameters are confirmed, the voting algorithm will run. This is referred to under run() and askForConfirmation() methods.

### 7.5. ID: UC\_05

**Description:** Programmer Runs Tests of Voting Algorithm

**Requirement Type:** Functional

**System:**

This requirement is fulfilled in the VotingApp class as the programmer is asked to turn off the shuffle. It is implied that if the shuffle is turned off a test of the voting algorithm is being run as the results will not be valid for an actual election. The user will be prompted if they want to turn off the shuffle option for an STV election. The user will then type in either “yes” or “no” which will return a true or false boolean for the algorithm. This is referred to under the askForShuffleTurnOff() method.

### 7.6. ID: UC\_06

**Description:** User Displays Help Window

**Requirement Type:** Functional

**System:**

This requirement is fulfilled under the VotingApp class. A help display will be shown as described in the displayHelp() method. The user will type in “help” in order to receive this message.

### 7.7. ID: BR\_01

**Description:** Up to 100,000 Ballots Should be Processed

**Requirement Type:** Nonfunctional

**System:**

This requirement will be fulfilled by the data structure of Election. The Election class will hold a vector of ballots up to 100,000 in length.

### 7.8. ID: BR\_02

**Description:** Elections should be processed in under 5 minutes

**Requirement Type:** Nonfunctional

**System:**

This time requirement was considered when choosing the data structure of ballots and candidates.

## Appendix A: Droop Quota Algorithm

The Droop Quota algorithm is as follows:

- Step 1: Shuffle the ballots.
- Step 2: Calculate the Droop Quota:

$$\left[ \left( \frac{\text{numberOfBallots}}{\text{numberOfSeats}+1} \right) \right] + 1$$

- Step 3: Distribute the ballots one-at-a-time into piles for the candidates. (The piles are ordered by the first vote received, not alphabetically.) If any candidate has reached the quota (for example, 22 ballots), they are immediately declared elected, those ballots are removed permanently from the process, and the elected candidate's name is recorded (in the order elected, not alphabetically). Subsequently, any ballot that follows in the distribution with their name listed goes to the next name on the list.
- Step 4: After the first round of distributing the candidates, the candidate with the fewest number of votes is permanently eliminated and their ballots are redistributed. (In the case of a tie, the candidate who was last to receive their first ballot is eliminated.) As in the first round, if any candidate reaches quota they are immediately declared elected, etc. There may be multiple rounds repeating this process.
- Step 5: You will continue this process until all ballots have been processed. You will have two lists, the elected and the non-elected. These lists will be ordered based on when they were elected and when they were placed on the non-elected list. It is possible for a "winner" of a seat to be on the non-elected list if they did not reach quota. The candidates placed on the non-elected list last will be higher up on the election list itself. You must provide both lists to the user.
- Step 6: The user wants a report that acts as an audit for the election. This report should show the ballots that were assigned to a candidate as the election progressed. The report does not print to the screen but should be in a text file. All pertinent election data must be included in the report at the top of the report (e.g. type of election, number of seats, number of candidates, winners, losers, etc).

## Appendix B: Plurality Voting Algorithm

The plurality algorithm should work as follows:

- Each ballot will be assigned to a candidate and a count maintained for each candidate.
- The candidate(s) with the greatest number of votes will be declared the winner(s).
- If there is a tie for a given seat, you will randomly select the winner of that seat.