Team 14

Voting System

Software Design Document

Names:
Alex Gude (gudex009)
Colin Hommerding (homme093)
Chris Walaszek (walas013)
Fan Ding (ding0322)
Section: 002

Date: 12/28/2021

Table of Contents

1. Introduction

1.1. Purpose

This software design document describes the architecture and system design of the Voting System. The expected audience is voting officials and programmers that are interested in this system.

1.2. Scope

This document contains a complete description of the design of the Voting System. The goal for the Voting System is to provide an easy-to-use tool for users to calculate ballots, generate and share election results. Users can confirm ballot information, generate a shareable media file that contains the results, and also an audit file that contains the processes of the election.

1.3. Overview

The remaining chapters and their contents are listed below.

Section 2 is a system overview that shows the functionality, context, and design of the Voting System at a high level.

Section 3 is the Architectural Design that specifies the subsystems that collaborate to perform the election. Each of these subsystems has a detailed description concerning the services that it provides to the rest of the system. Each subsystem is expanded into a set of lower-level design operations that collaborate to perform its services.

Section 4 concerns the Data Structure Design. It explains how the input file information is transformed into data structures and how the data are stored, processed, and organized.

Section 5 contains the Component design. It gives an organized description of each function in the system.

Section 6 discusses User Interface Design.

Section 7 is the requirement matrix. It matches system functionality to the function requirements requested in the SRS document,

1.4. Reference Material

List any documents, if any, which were used as sources of information for the test plan.
Voting System Software Requirements Specification (SRS):
https://docs.google.com/document/d/1RMslqZzDVmZrEF0zXGz70teEGBIkukuf/edit
Software Design Document Template:

https://canvas.umn.edu/courses/217913/files/19378011?wrap=1
Software Design Document Instructions:
https://canvas.umn.edu/courses/217913/files/19382790?wrap=1

1.5. Definitions and Acronyms

IR: as known as Instant Runoff, is a voting algorithm developed to ensure that the winning candidate has the majority support. If no candidate reaches 50% of total votes. The candidate with the fewest votes will be eliminated, and his/her votes will be distributed to other candidates. This redistributing process continues until one candidate has equal or above 50% of total votes, or only two candidates remain. Instant Runoff does not require a separate election, because there will also be a winner.

OPL: as known as Open Party List. It's a voting algorithm that proportionally allocates seats to parties by vote quota. Compared to a closed party list, the open party list system allows voters to select individuals rather than just parties.

Audit File: A file that contains each step of the election progress. The purpose of this file is to help users to verify the voting result.

Media File: A file that contains basic information for the voting result, including the winner, total ballots, number of candidates. The purpose of this file is to provide an intuitive result that can be shared through media.

Majority vote: In IR election, the candidate that reaches the majority vote wins the election. The Majority vote can be calculated by this formula: ceil(the total number of ballots)/2.

Popularity vote: In IR election, when there are only 2 candidates left in the election due to all other candidates being eliminated in previous runs. Between these 2 remaining candidates, if none of them reaches the majority vote, then the candidate with the most votes wins the election.

Quota: In OPL election, quota is used to determine how many seats a party can get at the first allocation. The quota can be calculated by taking the total number of valid votes in the district and dividing this by the number of seats.

SRS: Software Requirements Specification

2. System Overview

Voting System is a standalone system built for CSCI 5801 in the Spring 2021 semester. The development language used for this project is C++.

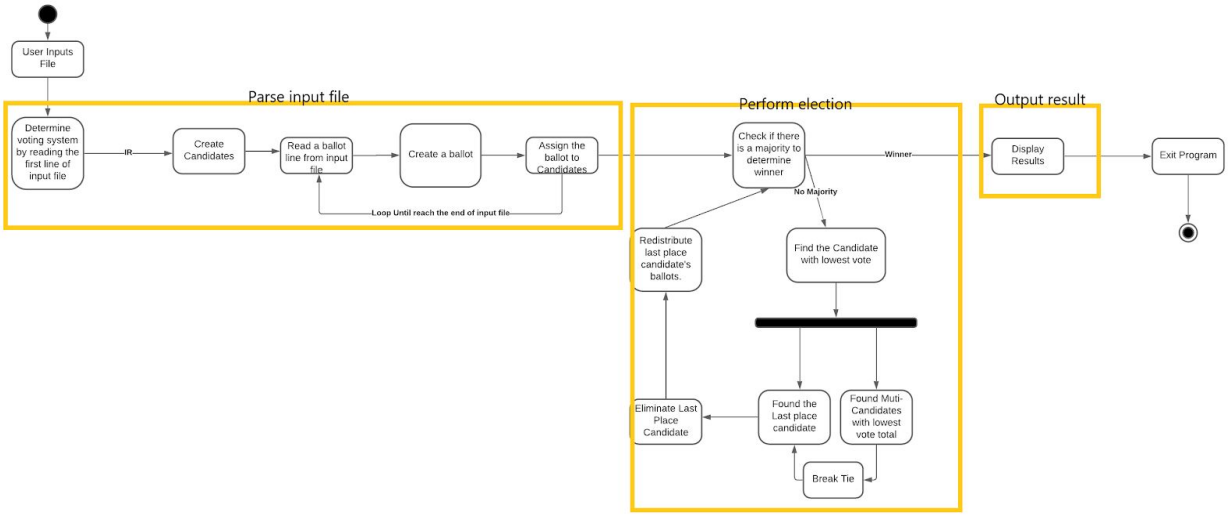
The system was designed to meet a need to automate the counting of election ballots in a quick and accurate manner. Moreover, the types of elections that specifically needed this capability were Open Party List and Instant Runoff, both of which were required to be run on the same program. This program must also produce accurate and detailed audit and media files after calculating the result of an election. The intended users are Election Officials, Media Personnel, and Testers.

To provide this capability, the Voting System must receive a correctly formatted ballot file as specified in section 5.6 of the SRS. After a user confirms the information in this file, it is parsed as either an OPL or IR ballot file accordingly. Each of these represent a feature of the system that is conducted in accordance with the needs of the user - producing both an audit and media file and strictly following the rules of each respective election process. When this process is complete, a brief overview of the election is output to the screen for reading by the user.

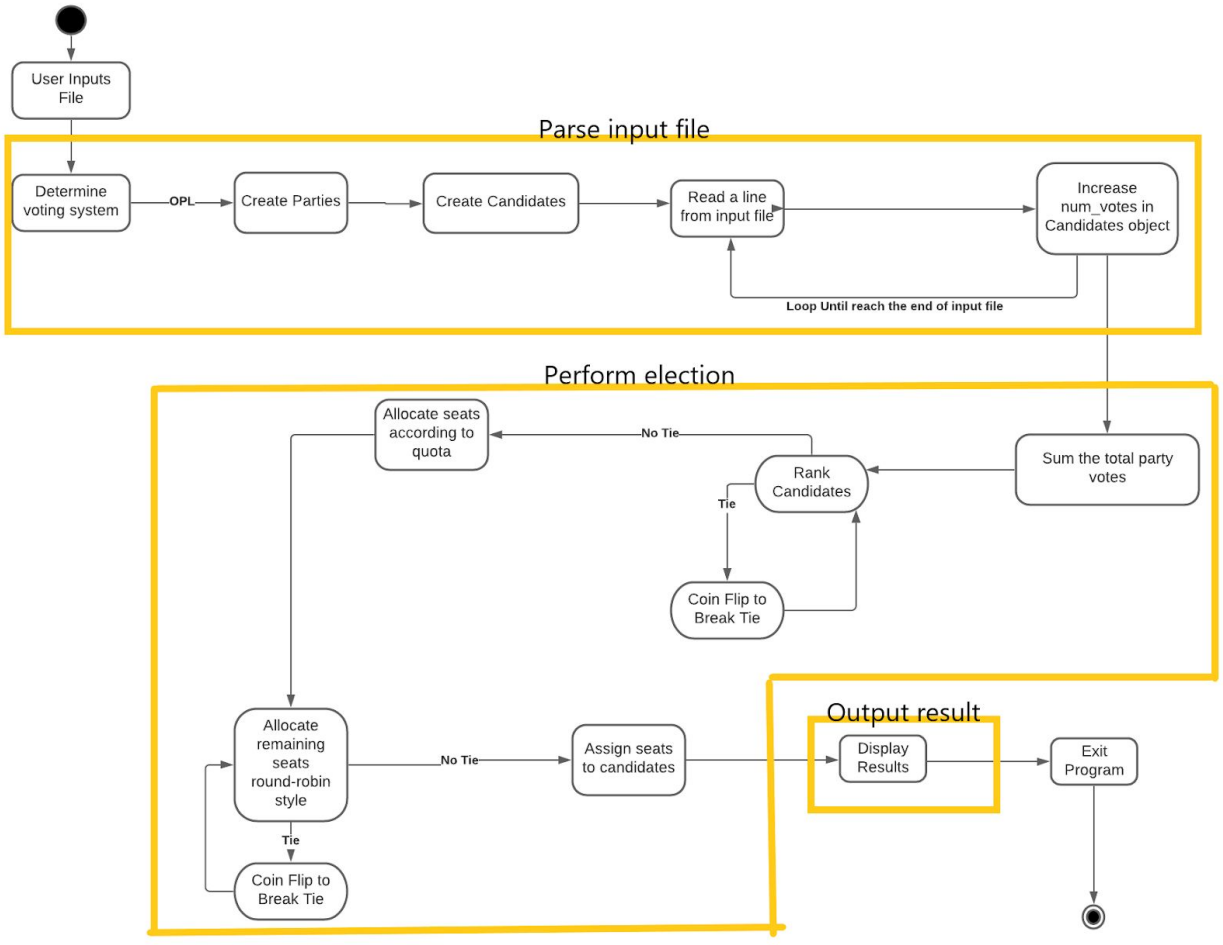
3. System Architecture

3.1. Architectural Design

We divided our Voting System into 3 subsystems: Parse Input File, Perform Election, and Output Results. The Parse Input File subsystem will scan the input file and store information into variables and designed classes. Then the Perform Election subsystem will use that data to calculate the election result. Lastly, the Output Results subsystem will show the result to the user by displaying the preliminary result on screen, and at the same time, a media file will be generated in the same repository as the program. Activity diagrams illustrating these subsystems for IR and OPL elections are shown below.

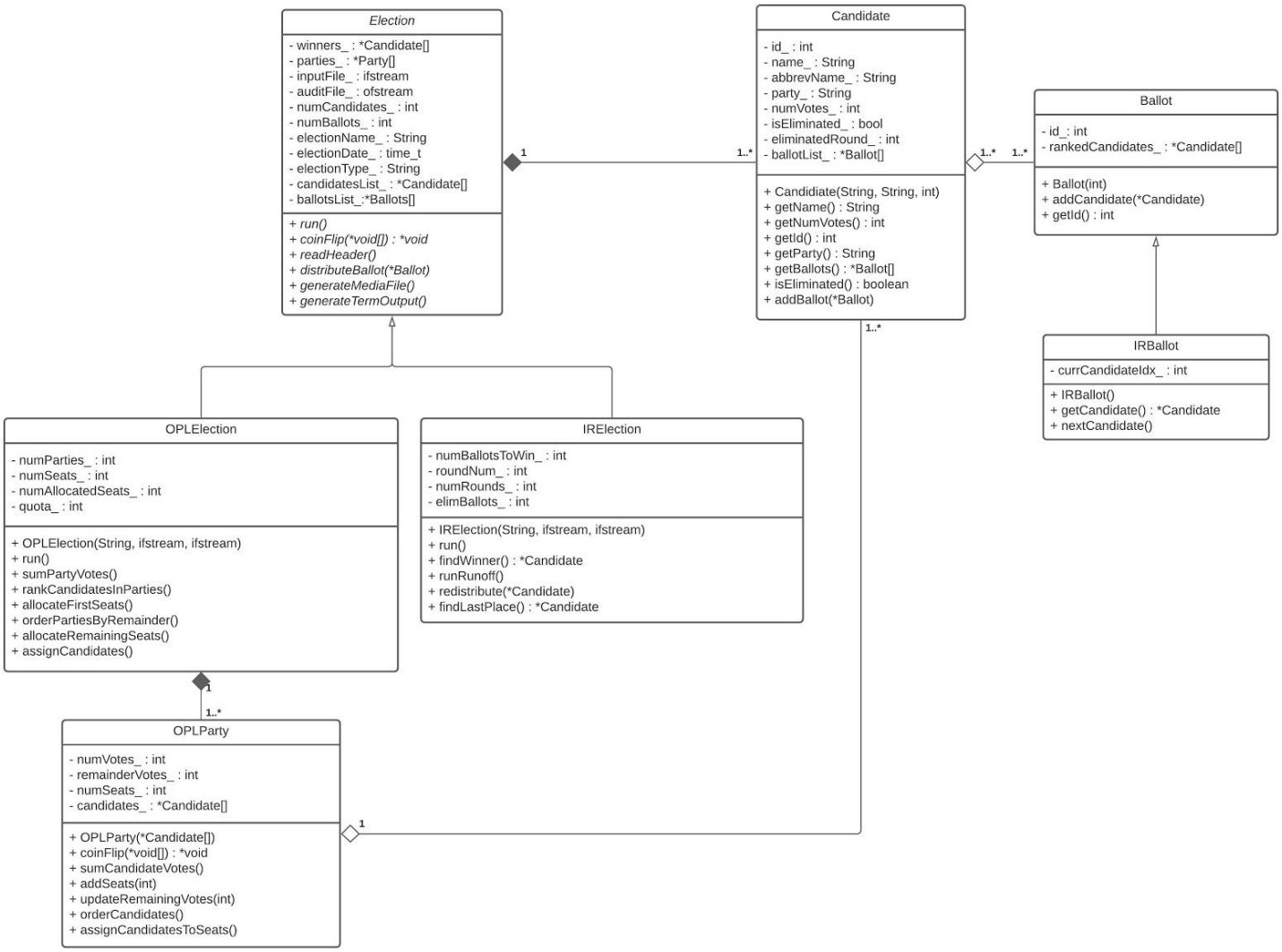


IR Election Subsystem



OPL Election Subsystem

3.2. Decomposition Description



Voting System Class Diagram

3.2.1. Parse Input File

After the program is started and the user enters in the input file and election name, the input file is opened and checked if the voting process is IR or OPL. Depending on the election type, a corresponding Election object (IRElection or OPLElection) is created and the input file, election name, and an open audit file are passed in. The election object's run() method is then called, which drives the whole election process. Through the readHeader() method, all relevant data from the input file, not including the ballots, is parsed and stored in the Election, Candidate, and (in the case of OPL) Party objects.

Pointers to all of the Candidate and Party objects are stored in arrays of the corresponding type in the Election class. Any necessary values, such as the number of votes for a 50% majority in IR or the quota in OPL, are also calculated via the readHeader() method.

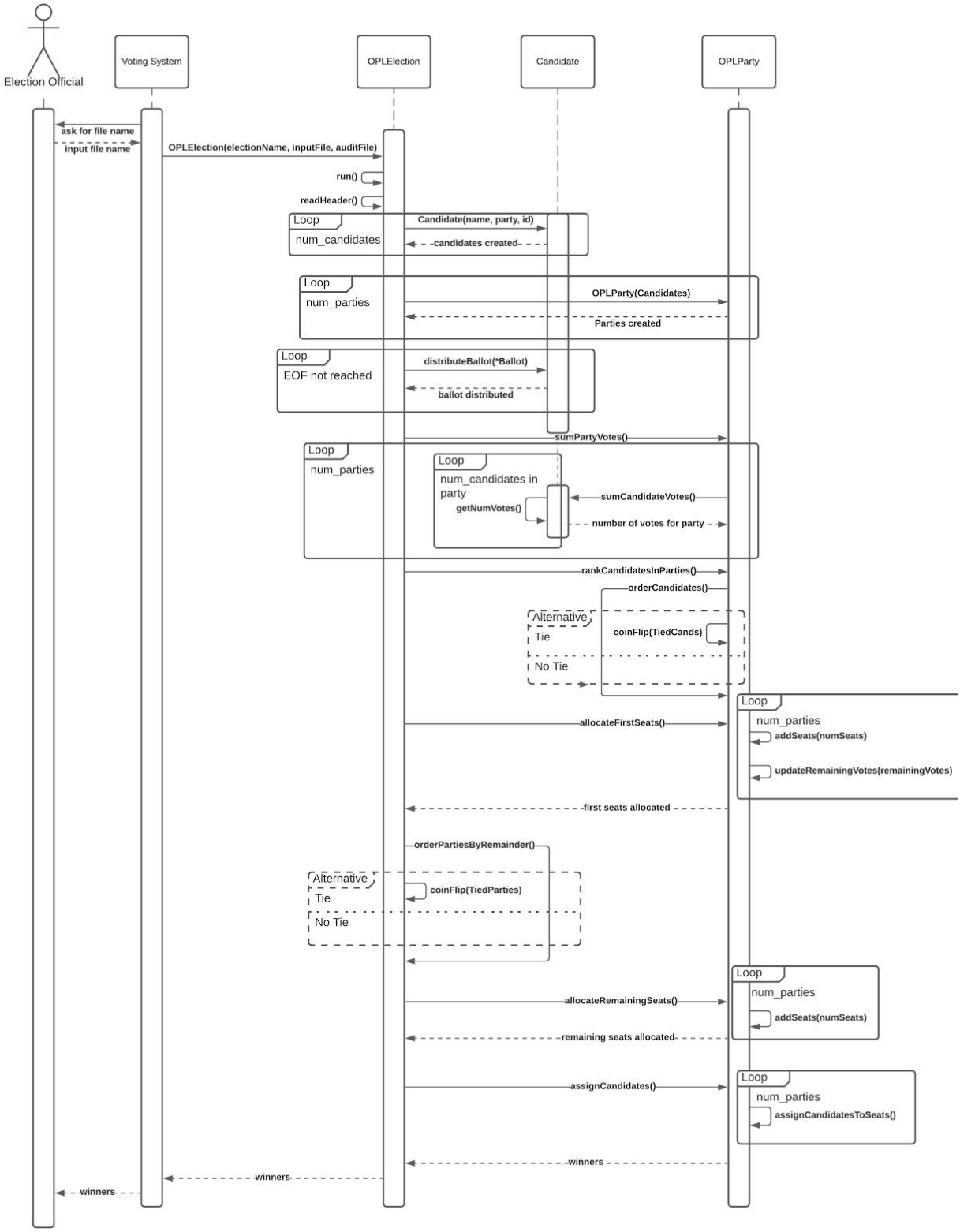
The ballots are then parsed, created, and distributed to the relevant Candidate and (if OPL) Party objects through the distributeBallot() method. All necessary information calculated up to this point is logged in the audit file.

3.2.2. Perform Election

After parsing input, the run() method begins to calculate the results of the election. This process depends on the election type. Throughout the process, the audit file is continually updated in parallel with the election calculation through side effects in the relevant methods.

3.2.2.1 Perform IR Election

The findWinner() method searches the vote distribution for a winner of the election, whether through majority vote or popularity vote if there are only two candidates left. If a winner is found, the election ends. If findWinner returns a nullptr, then the runRunoff() method is called, which finds the candidate with the fewest votes through findLastPlace() (breaking ties if necessary with coinFlip()) and redistributes that candidate's ballots according to each ballot's next choice with redistribute(). This process of calling findWinner() and runRunoff() continues in a while loop until findWinner() does not return a nullptr. When a winner is found, a pointer to the winner's Candidate object is stored in the IRElection's winner_ array.



Sequence Diagram for Perform OPL Election Use Case (described in 3.2.2.2)

3.2.2.2 Perform OPL Election

The OPLElection object calls on the Party objects to order their candidate according to the number of votes each candidate got through the rankCandidatesInParties() method, calling coinFlip() in the case of ties. The allocateFirstSeats() method is then called, which awards seats to each Party object according to the calculated quota and the number of votes each party got by calling the Party's addSeats() method. After the first allocation is counted, allocateFirstSeats() calculates the party's remaining votes and stores the value in each Party object's remainderVotes_ attribute.

Through OrderPartiesByRemainder(), the OPLElection orders the parties according to each party's remainderVotes_ attribute, breaking ties with CoinFlip() if necessary. The allocateRemainingSeats() then assigns additional seats round-robin style, assigning one seat to each party as it goes down the ordered list. If allocateRemainingSeats() reaches the end of the list and not all seats are allocated, it goes back up to the top of the list and allocates seats in the same manner until all seats are allocated.

After all parties are assigned seats, all candidates in each party are awarded seats according to the number of seats their party got through the assignCandidates() method. Pointers to the winners are stored in the OPLElection's winners_ array.

3.2.3. Output Results

As the election is tabulated, any data values needed for the media file and preliminary terminal output are stored inside the Election class through side effects in the relevant functions. Through the generateMediaFile() method, this data is parsed and organized into a .txt file stored in the executable's current directory, for use by the media in publishing the election's results. In a similar manner, generateTermOutput() parses and organizes the data into a terminal output, which shows upon completion of the program.

3.3. Design Rationale

To limit the amount of data to store in the system, we decided to produce the audit file in parallel to the election calculation itself, instead of creating the file after the calculation was completed. We felt that, unlike the media file and the terminal output, the audit file was too large and complex to store all the needed information in memory at once. However, to implement this parallel approach many functions need to have side effects where they update the audit file along with performing a calculation, which reduces the transparency of the system.

4. Data Design

4.1. Data Description

The process will begin with the creation of an object of the proper election type read from the CSV file. The election object will maintain the election by keeping track of candidates in a candidate array, ballots in a ballot array, number of ballots in an integer, and number of candidates in an integer. Then for Instant Runoff Elections, candidate objects are initialized based on the candidates that are read from the CSV file. The same process is used for Open Party List elections but in this case objects for both candidates and parties are initialized.

For Instant Runoff Elections, each vote within the CSV file is initialized into a ballot object that contains information on who that voter voted for by creating the ballot with Ballot() constructor method and then filling in its attributes. Each ballot has an array that stores candidates in the ranked order and a method to obtain the current candidate the ballot is for. Each ballot also has an integer attribute representing the index of the candidate in the array that the ballot is voting for. There is also a method to increment the integer attribute to move onto the next ranked choice. Continuing with Instant Runoff Elections the ballots are then distributed to candidate objects representing the actual candidates themselves. These candidates have ballot arrays that contain all of that candidate's ballots. If ballots need to be redistributed then the candidate with the least amount of votes has every vote in their ballot array moved to that ballot's next preferred candidate's ballot array.

For Open Party List elections no ballot objects will be used. Instead votes will be counted and an integer variable within the candidate object will be incremented to represent each vote. Each party object then has a method that allows it to count all its votes by counting and adding the votes of its candidates. Each party also has an integer variable that keeps track of the number of seats it has been awarded based on its number of votes. The party objects then have a method to award their seats to the candidates based on who has the most votes.

4.2. Data Dictionary

Object	Object Attributes	Object Methods
Candidate: The candidate object represents a candidate running for office. It is capable of storing all information that defines the candidate and can show his	id_ : int name_ : String abbrevName_ : String party_ : String numVotes_ : int	Candidate(String, String, int) getName() : String getNumVotes() : int getId() : int getParty() : String

performance in the election	isEliminated_ : bool eliminatedRound_ : int ballotList_ : *Ballot[]	getBallots() : *Ballot[] isEliminated() : boolean addBallot(*Ballot)
IRBallot: The ballot object is only used for Instant Runoff Elections. It allows for ranked choice voting to be used and is needed to show where each ballot goes as the election progresses.	id_: int RankedCandidates_ : *Candidate[] currCandidateIdx_ : int	IRBallot(int) getId() : int addCandidate(*Candidate) getCandidate() : *Candidate nextCandidate()
IRElection: The IRElection object is needed as an overhead to conduct an Instant Runoff Election so that all information is saved and results can be properly processed.	winners_ : *Candidate[] parties_ : *Party[] inputFile_ : ifstream auditFile_ : ofstream numCandidates_ : int numBallots_ : int electionName_ : String electionDate_ : time_t electionType_ : String candidatesList_ : *Candidate[] ballotsList_:*Ballots[] numBallotsToWin_ : int roundNum_ int numRounds_ : int elimBallots_ : int	run() coinFlip(*void[]): void* readHeader() distributeBallot(*Ballot) generateMediaFile() generateTermOutput() IRElection(String,ifstream,ifstream) findWinner() : *Candidate runRunoff() redistribute(*Candidate) findLastPlace() : *Candidate
OPLElection: The OPLElection object is needed as an overhead to conduct an Instant Runoff Election so that all information is saved and results can be properly processed.	winners_ : *Candidate[] parties_ : *Party[] inputFile_ : ifstream auditFile_ : ofstream numCandidates_ : int numBallots_ : int electionName_ : String electionDate_ : time_t electionType_ : String candidatesList_ : *Candidate[] ballotsList_: *Ballots[] numParties_ : int numSeats_ : int quota_ : int numAllocatedSeats_ : int	run() coinFlip(*void[]) : void* readHeader() distributeBallot(*Ballot) generateMediaFile() generateTermOutput() sumPartyVotes() rankCandidatesInParties() allocateFirstSeats() orderPartiesByRemainder() allocateRemainingSeats() assignCandidates() OPLElection(String, ifstream, ifstream)
OPLParty: The Party object is used to represent each party that has a candidate in the election. It is needed so that awarded seats can be stored and properly distributed to the proper candidates.	numVotes_ : int remainderVotes_ : int numSeats_ : int candidates_ : *Candidate[]	OPLParty() coinFlip(*void[]) : *void sumCandidateVotes() addSeats(int) updateRemainingVotes(int) orderCandidates() assignCandidatesToSeats()

5. Component Design

Candidate:

Name	Return Type	Description
Candidate(String, String, int)	void	Constructor that initialize candidate, parameters are name, party, id
getName()	String	Returns the name of a candidate stored in the name_ attribute.
getNumVotes()	int	Returns the current number of votes a candidate has in the numVotes variable.
getId()	int	Returns the ID of the candidate stored in the id_ attribute.
getParty()	String	Returns the party that the candidate is a member of stored in the party_ variable.
getBallots()	*Ballot[]	Returns an array containing each ballot cast for that candidate contained in the ballotList_ array. Specifically used for Instant Runoff Elections.
isEliminated()	bool	Returns true if the candidate was eliminated and is out of the running for winning the election. Returns false if the candidate is still in the running to win. Is only used in Instant Runoff Elections. The boolean is contained in the isEliminated_ attribute.
addBallot(*Ballot)	void	Takes a pointer to a ballot and adds that ballot to the candidates ballot array.

IRBallots:

Name	Return Type	Description
IRBallot(int)	void	Constructor that initializes a ballot. Takes in an id number as an int
getId()	int	Returns the ID of a ballot stored in the id_ attribute.
getCandidate()	*Candidate	Returns the candidate that the ballot was cast for in the currCandidate variable.

addCandidate(*Candidate)	void	Takes a candidate pointer and adds it to the end of the Ballot's candidate array.
nextCandidate()	void	Increments the value within the currCandidateIdx attribute

IRElection:

Name	Return Type	Description
IRElection(String, ifstream, ifstream)	void	Constructor that initializes an IRElection object where the string is the name of the election, the first ifstream is the input file, and the second ifstream is the file for the audit file.
run()	void	This function is used to run the entire election. It will be the main driver of the actions within the election class.
coinFlip(*void[])	*void	It will take in an array of multiple pointers as the parameter. Then will use random number generation to randomly select an index in the array and return the object at that index.
readHeader()	void	Reads important information from the input file header into the election object. Stores the corresponding information in the parties_, numCandidates_, numBallots_, and candidatesList_ variables. Also calculates and stores the majority vote in numBallotsToWin_.
distributeBallot(*Ballot)	void	Takes ballots located in the elections ballotList_ and places them in the corresponding candidates ballotList_ according to the voter's first ranked choice.
generateMediaFile()	void	Creates the media file according to the results of the election.
generateTermOutput()	void	Prints the preliminary results of the election to the screen.
findWinner()	*Candidate	Searches the vote distribution for a winner of the election, whether through majority vote or popularity vote if there are only two candidates left. If a winner is found, returns a Candidate pointer. If no winner is found, returns a nullptr.
runRunoff()	void	Finds the candidate with the fewest votes through findLastPlace() (breaking ties if necessary with coinFlip()) and redistributes that candidate's ballots according to each ballot's next choice with redistribute().

redistribute(*Candidate)	void	Takes an eliminated candidate and moves all the ballots from their ballotList_ to each ballot's next choice according to the rankings and places it in that candidate's ballotList_.
findLastPlace()	*Candidate	Returns the candidate with the fewest amount of current votes.

OPLElection:

Name	Return Type	Description
OPLElection(String, ifstream, ifstream)	void	Constructor that initializes an OPLElection object where the string is the name of the election, the first ifstream is the input file, and the second ifstream is the file for the audit file.
run()	void	This function is used to run the entire election. It will be the main driver of the actions within the election class.
coinFlip(*void[])	*void	It will take in an array of multiple pointers as the parameter. Then will use random number generation to randomly select an index in the array and return the object at that index.
readHeader()	void	Reads important information from the input file header into the election object. Stores the corresponding information in the parties_, numCandidates_, numBallots_, and candidatesList_ variables. Also calculates and stores quota in quota_.
distributeBallot(*Ballot)	void	Takes ballots located in the elections ballotList_ and places them in the corresponding candidates ballotList_ according to the voter's first ranked choice.
generateMediaFile()	void	Creates the media file according to the results of the election.
generateTermOutput()	void	Prints the preliminary results of the election to the screen.
sumPartyVotes()	void	Adds up the total number of votes within the election.
rankCandidatesInParties()	void	Ranks each candidate in their respective parties according to their numVotes_ attribute.
allocateFirstSeats()	void	Awards seats to each Party object according to the calculated quota and the number of votes each party got by calling the Party's addSeats() method.

orderPartiesByRemainder()	void	Orders the parties according to each party's remainderVotes_ attribute, breaking ties with CoinFlip() if necessary.
allocateRemainingSeats()	void	Assigns additional seats round-robin style, assigning one seat to each party as it goes down the ordered list. If allocateRemainingSeats() reaches the end of the list and not all seats are allocated, it goes back up to the top of the list and allocates seats in the same manner until all seats are allocated.
assignCandidates()	void	All candidates in each party are awarded seats according to the number of seats their party got and pointers to the winners are stored in the OPLElection's winners_ array.

OPLParty:

Name	Return Type	Description
OPLParty(*Candidate[])	void	Constructor that initializes a party object. A list of candidate pointers is passed in as an input parameter.
coinFlip(*void[])	*void	It will take in an array of multiple pointers as the parameter. Then will use random number generation to randomly select an index in the array and return the object at that index.
sumCandidateVotes()	void	Adds up all the votes of each candidate in the party according to each candidate's numVotes_ attribute and stores it in the party's numVotes_ attribute.
addSeats(int)	void	Increments the numSeats_ variable of the party by the int parameter taken in.
updateRemainingVotes(int)	void	Updates the Party's remainderVotes_ attribute.
orderCandidates()	void	Orders the candidates in the party's candidates_ array according to vote total.
assignCandidatesToSeats()	void	Goes through the ranked candidates_ array declaring the amount of winners equal to the party's numSeats_ attribute.

6. Human Interface Design

6.1. Overview of User Interface

The user will be able to use the system by running it on the command line within a terminal. They will first run the system by typing ".voting.exe" and pressing enter. The system will then prompt the user for the election name, which the user can give the system by typing on the command line and hitting enter. After that the system will ask for the election CSV file, which the user will also have to give the system by typing on the command line and hitting enter. Once the input file is given and checked if valid, then the election will be computed and preliminary results will be printed to the terminal to display on screen. These preliminary results will include winners, type of election, and number of seats. It will also include the number of ballots cast and the number of votes each candidate got.

6.2. Screen Images

1. Starting Program

On Screen Terminal

```
user@csel:~$ ./Voting.exe
```

2. Give Program Election Name

On Screen Terminal

```
< Give Election Name:
user@csel:~$ "Maine Senate Election"
```

3. Give Program Input File

On Screen Terminal

```
< Give Input File:
user@csel:~$ "inputfile.CSV"
```

4. Preliminary Results for Instant Runoff Election

Printed to Terminal to Display on Screen

```
*** UNITED STATES MAINE SENATE RACE ***

Type of Election : Instant Runoff

*** CANDIDATES ***
5 Candidates:
Alice Appleseed (Sensible)
Bob Burns (Sensible)
Charlie (Silly)
David Van Doe (Silly)
Isabella Indy (Independent)

*** RESULTS ***
Total Votes - 15,660

Alice Appleseed (Sensible) : (51%) - 7,987 Votes - ELECTED
Bob Burns (Sensible) : (30%) - 4,698 Votes
Charlie (Silly) : (19%) 2,975 Votes
David Van Doe (Silly) : Eliminated Round 2
Isabella Indy (Independent) : Eliminated Round 1
```

5. Preliminary Results for Open Party List Election

Printed to Terminal to Display on Screen

```
*** UNITED KINGDOM GENERAL ELECTION ***

Type of Election : Open Party List

*** ELECTION OVERVIEW ***
4 Parties, 6 Candidates
4 Seats to be filled

*** PARTIES AND CANDIDATES ***
Independent:
  Isabella Indy
Sensible:
  Alice Appleseed
  Bob Burns
Silly:
  Charlie
  David Van Doe
Slightly Silly:
  Erica Esther


*** RESULTS ***
Total Votes - 18,807 Votes

Silly (51%) - 9592 Votes
Sensible (45%) - 8463 Votes
```

```
Independent (2%) - 376 Votes
Slightly Silly (2%) - 376 Votes

*** Winners ***
Silly:
  Charlie - ELECTED
  David Van Doe - ELECTED
Sensible:
  Alice Appleseed - ELECTED
  Bob Burns - ELECTED
```

6.3. Screen Objects and Actions

The main screen object is the terminal. The actions associated with the object are starting the program, giving the program an election name, giving the program the input file, and printing preliminary results. Other objects are the audit and media text files which if the user decides to open are displayed on screen as text files.

7. Requirements Matrix

Section # in SRS	Functional Requirement	Function
4.1.3 - REQ1	User is prompted for input file name	run() main()
4.1.3 - REQ2	Failure to provide a valid file name is met with another prompt	run() main()
4.2.3 - REQ1	System can read election information from the header (OPL)	readHeader()
4.2.3 - REQ1	OPL ties are handled by a coin flip	*<OPLElection>* coinFlip(*void[])
4.2.3 - REQ3	Independent candidates are grouped into one party	readHeader()
4.2.3 - REQ4	Program displays preliminary results upon completion (OPL)	generateTermOutput()
4.2.3 - REQ5	Program follows OPL protocol for vote distribution	sumPartyVotes() rankCandidatesInParties() allocateFirstSeats()

		orderPartiesByRemainder() allocateRemainingSeats() assignCandidates() sumCandidateVotes() addSeats(numSeats) orderCandidates() assignCandidatesToSeats() updateRemainingVotes(remainingVotes)
4.3.3 - REQ1	System can read election information from the header (IR)	readHeader()
4.3.3 - REQ2	IR ties are handled by a coin flip	<*Election*> coinFlip(TiedParties)
4.3.3 - REQ3	Running out of ranked choices means a ballot is disqualified	redistribute(*Candidate elimCandidate)
4.3.3 - REQ4	No majority with only two candidates remaining gives the victory to the candidate with more votes	findWinner()
4.3.3 - REQ5	Program displays preliminary results upon completion (IR)	generateTermOutput()
4.4.3 - REQ1	A detailed audit file is created	main()
4.5.3 - REQ1	A media file is created	generateMediaFile()
4.5.3 - REQ2	The media file has characteristics that make it easily read	generateMediaFile()