Team 13

OPL/CPL Voting System

Software Design Document

Names: Connell Hagen, Michael Mulhall, Grant Oie, Khalid Qasim

Date: (03/01/2024)

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

1.2. Scope

1.3. Overview

1.4. Reference Material

2. SYSTEM OVERVIEW

3. SYSTEM ARCHITECTURE

3.1. Architectural Design

3.2. Decomposition Description

3.3. Design Rationale

4. DATA DESIGN

4.1. Data Description

4.2. Data Dictionary

5. COMPONENT DESIGN

6. HUMAN INTERFACE DESIGN

6.1. Overview of User Interface

6.2. Screen Images

6.3. Screen Objects and Actions

7. REQUIREMENTS MATRIX

8. APPENDICES

1. Introduction

1.1 Purpose

This Software Design Document (SDD) details the architectural and system design of a voting system aimed at facilitating both open-party list and closed-party list elections. This SDD is designed to be a guide for the development, deployment, and maintenance of the voting system, ensuring adherence to the requirements specified in the Software Requirements Specification and providing a clear understanding of the system's design and functionality. This document describes the system architecture and design specifications essential for an effective system. This system will facilitate electoral processes for open-party list and closed-party list elections, ensuring accuracy and transparency in the allocation of seats by the largest remainder approach.

The intended audience for this software design document includes:

Programmers: Programmers will utilize this software design document to understand the system's design and to develop the system in line with requirements.

Testers: Testers will utilize this software design document in developing and executing tests that confirm the system's functionality and performance meet the project requirements.

Election Officials: Election officials will understand the system's procedures and outputs to manage the electoral processes. Election Officials are not directly involved in development.

1.2 Scope

The scope of this software design document is the entire OPL/ CPL election system. The OPL/ CPL election system will all be contained in one directory and will include the software that is used to take in a file of election information, determine which type of election is being held, allocate seats to the winners of the election, in the case of a tie determine the winner fairly, generate an audit file, and display the results to the election official using this system. The benefit of this system is that it provides a fast and accurate result for any size election and can be used for two different, popular types of elections.

1.3 Overview

This document translates the specifications and requirements detailed in the Software Requirements Specification document[1] into a well-structured software model design. It details the high-level system architecture, data design and flow, component design, UI design, and the requirements matrix. The emphasis of this document is on identifying and detailing the large modules and data structures of the system and providing a guideline for implementation. The aim is to ensure that the computerized model fully aligns with the outlined user expectations, election cycle concerns, including the versatile capacity for open and closed party list election methodologies, and the seat allocation across varied contingents.

1.4 Reference Material

[1] CSCI 5801, Group 13 – Software Requirements Specification for CPL/OPL Voting System

2. System Overview

The system is broken into multiple abstract and regular classes that inherit from each other in order to return the final result of an election. The main data structures of the program will be ElectionData objects, which contain vectors of pointers to objects of type Party, which in turn will contain vectors to pointers of Candidate objects. The Party type is an abstract object which OPLParty and CPLParty inherit from. The Candidate type is an abstract object which OPLCandidate and CPLCandidate inherit from.

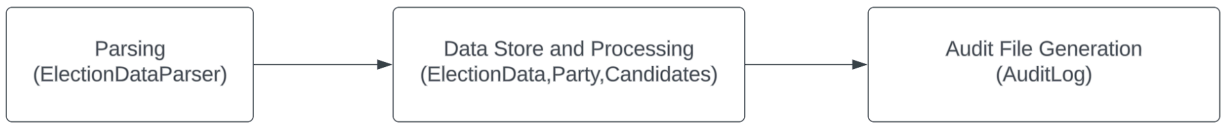
Depending on the type of election, the two classes OPLElectionData and CPLElectionData inherit from the abstract class ElectionData. These classes are where the critical election data will be stored, winners of seats will be calculated, the audit file will be generated, and the function for displaying the results will be written. These ElectionData objects are instantiated by the ElectionDataParser class, which is responsible for breaking down the information provided in the csv election file and constructing the appropriate objects.

What this means is that when the name of the csv file is entered into the text prompt and it has no errors:

1. ElectionDataParser collects and separates all important information to be used by other classes
2. Depending on the election type, one of two Party classes is created for each Party involved in the election
    a. Each of these Party objects is filled with corresponding Candidate objects.
3. Depending on the election type, one of two ElectionData classes then calculates the seat winners, generates an audit file, and displays the results to the user

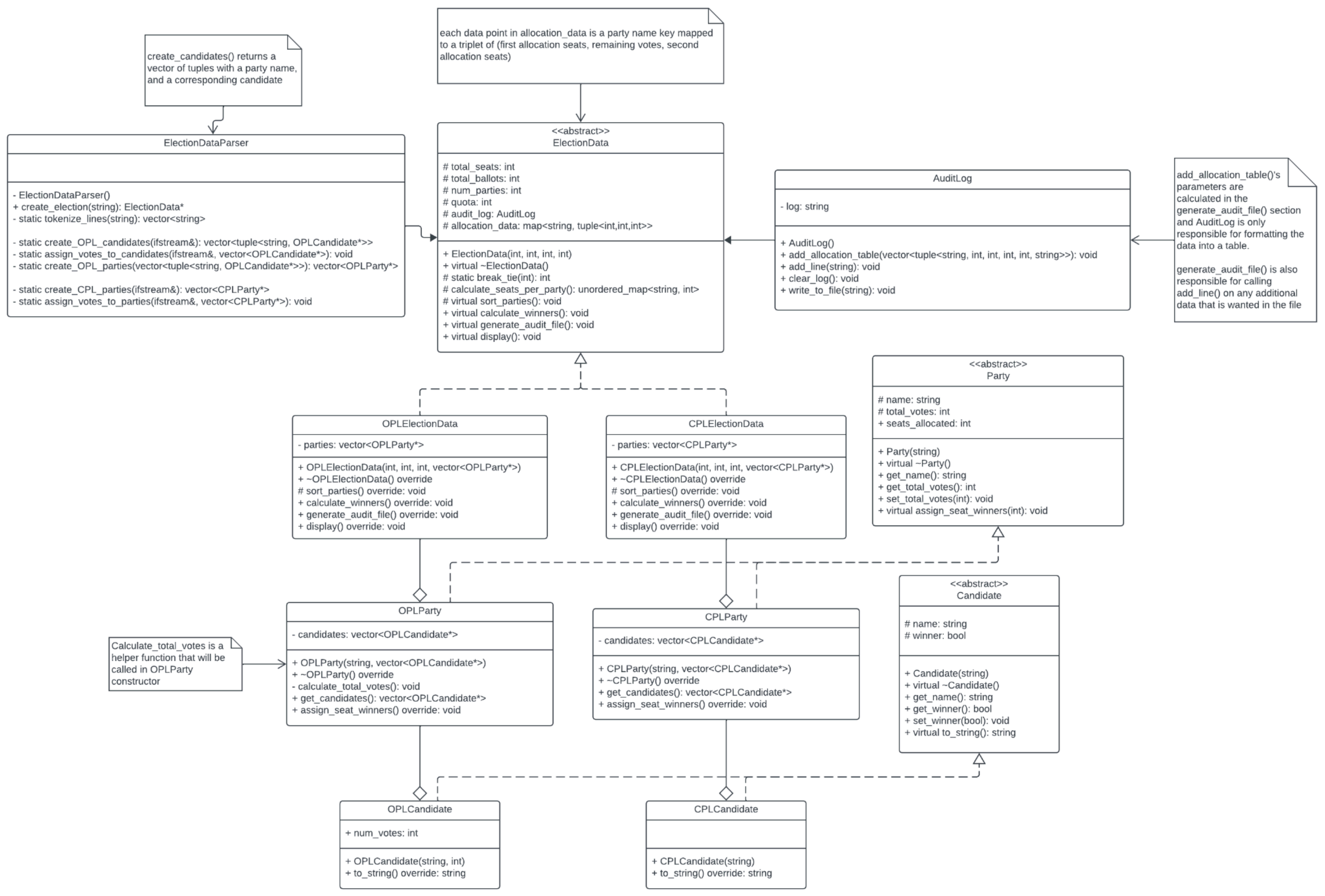
3. System Architecture

3.1 Architectural Design

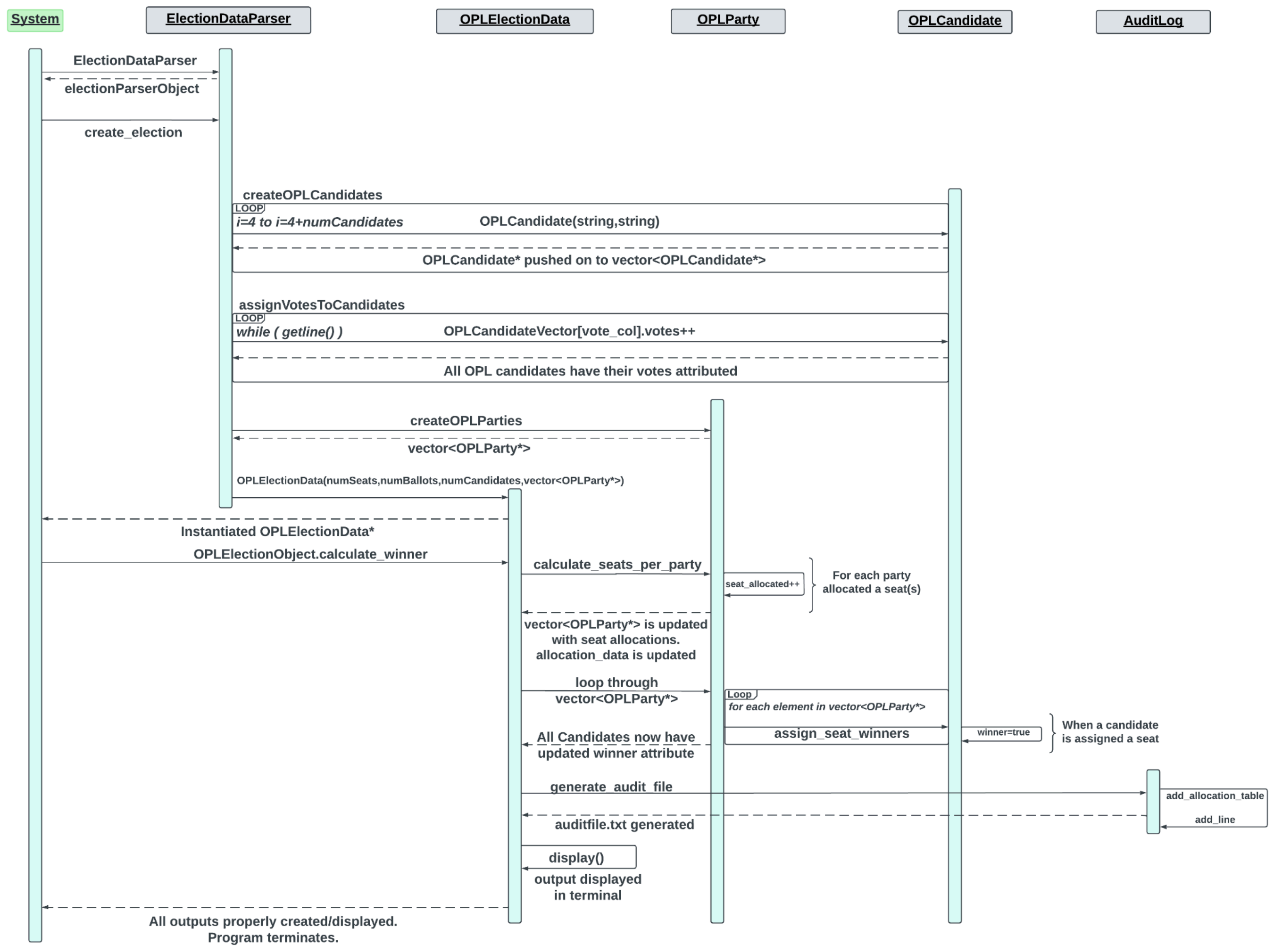


- Parsing System
  - Responsibility: To extract data from the ballot csv file and construct the appropriate objects to compose an ElectionData object
- Data Store and Processing System
  - Responsibility: To control the state data of election-related objects, perform required algorithms to determine election results, pass relevant data to AuditLog for audit file generation, and print the election results
- Audit File Generation
  - Responsibility: To properly format the election results, and save the audit file document locally

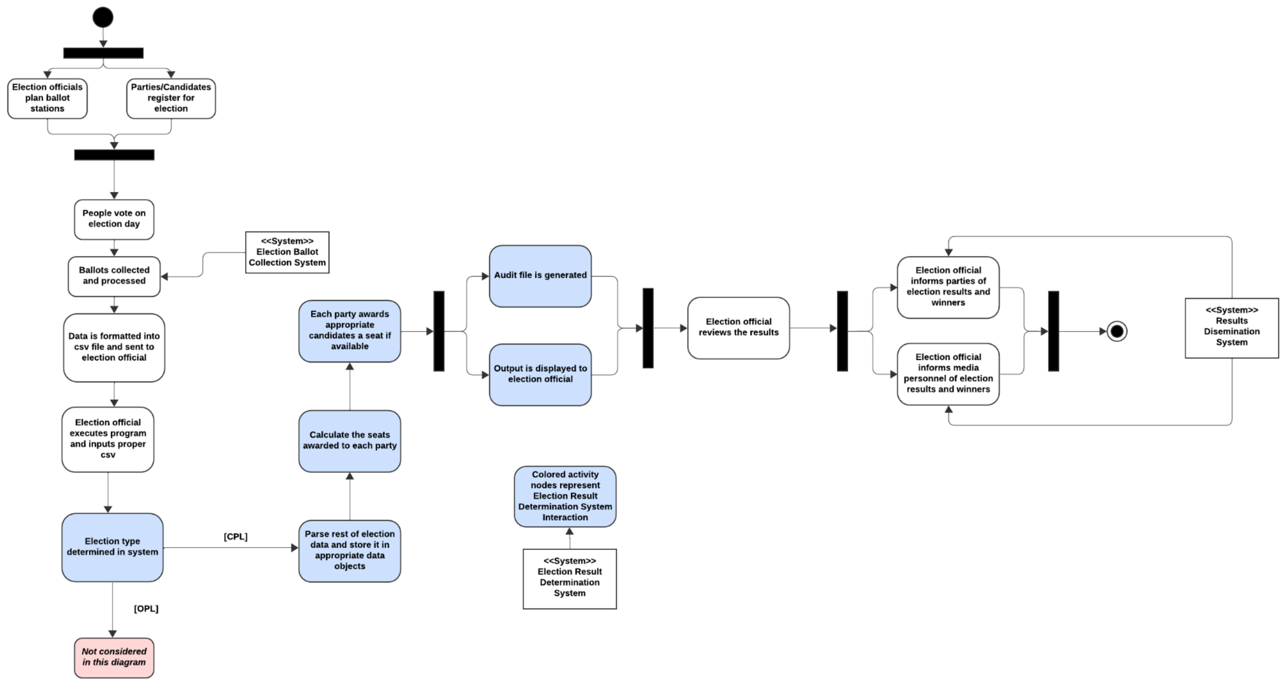
3.2 Decomposition Description


create_candidates() returns a vector of tuples with a party name, and a corresponding candidate
each data point in allocation_data is a party name key mapped to a triplet of (first allocation seats, remaining votes, second allocation seats)
ElectionDataParser
- ElectionDataParser()
+ create_election(string): ElectionData*
- static tokenize_lines(string): vector<string>
- static create_OPL_candidates(ifstream&): vector<tuple<string, OPLCandidate*>>
- static assign_votes_to_candidates(ifstream&, vector<OPLCandidate*>): void
- static create_OPL_parties(vector<tuple<string, OPLCandidate*>>): vector<OPLParty*>
- static create_CPL_parties(ifstream&): vector<CPLParty*>
- static assign_votes_to_parties(ifstream&, vector<CPLParty*>): void
<<abstract>>
ElectionData
# total_seats: int
# total_ballots: int
# num_parties: int
# quota: int
# audit_log: AuditLog
# allocation_data: map<string, tuple<int,int,int>>
+ ElectionData(int, int, int, int)
+ virtual ~ElectionData()
# static break_tie(int): int
# calculate_seats_per_party(): unordered_map<string, int>
# virtual sort_parties(): void
+ virtual calculate_winners(): void
+ virtual generate_audit_file(): void
+ virtual display(): void
AuditLog
- log: string
+ AuditLog()
+ add_allocation_table(vector<tuple<string, int, int, int, int, string>>): void
+ add_line(string): void
+ clear_log(): void
+ write_to_file(string): void
add_allocation_table()'s parameters are calculated in the generate_audit_file() section and AuditLog is only responsible for formatting the data into a table.
generate_audit_file() is also responsible for calling add_line() on any additional data that is wanted in the file
OPLElectionData
- parties: vector<OPLParty*>
+ OPLElectionData(int, int, int, vector<OPLParty*>)
+ ~OPLElectionData() override
# sort_parties() override: void
+ calculate_winners() override: void
+ generate_audit_file() override: void
+ display() override: void
CPLElectionData
- parties: vector<CPLParty*>
+ CPLElectionData(int, int, int, vector<CPLParty*>)
+ ~CPLElectionData() override
# sort_parties() override: void
+ calculate_winners() override: void
+ generate_audit_file() override: void
+ display() override: void
<<abstract>>
Party
# name: string
# total_votes: int
+ seats_allocated: int
+ Party(string)
+ virtual ~Party()
+ get_name(): string
+ get_total_votes(): int
+ set_total_votes(int): void
+ virtual assign_seat_winners(int): void
Calculate_total_votes is a helper function that will be called in OPLParty constructor
OPLParty
- candidates: vector<OPLCandidate*>
+ OPLParty(string, vector<OPLCandidate*>)
+ ~OPLParty() override
- calculate_total_votes(): void
+ get_candidates(): vector<OPLCandidate*>
+ assign_seat_winners() override: void
CPLParty
- candidates: vector<CPLCandidate*>
+ CPLParty(string, vector<CPLCandidate*>)
+ ~CPLParty() override
+ get_candidates(): vector<CPLCandidate*>
+ assign_seat_winners() override: void
<<abstract>>
Candidate
# name: string
# winner: bool
+ Candidate(string)
+ virtual ~Candidate()
+ get_name(): string
+ get_winner(): bool
+ set_winner(bool): void
+ virtual to_string(): string
OPLCandidate
+ num_votes: int
+ OPLCandidate(string, int)
+ to_string() override: string
CPLCandidate
+ CPLCandidate(string)
+ to_string() override: string

UML Diagram of OPL/CPL Voting System


System
ElectionDataParser
OPLElectionData
OPLParty
OPLCandidate
AuditLog
ElectionDataParser
electionParserObject
create_election
createOPLCandidates
LOOP
i=4 to i=4+numCandidates
OPLCandidate(string,string)
OPLCandidate* pushed on to vector<OPLCandidate*>
assignVotesToCandidates
LOOP
while (getline())
OPLCandidateVector[vote_col].votes++
All OPL candidates have their votes attributed
createOPLParties
vector<OPLParty*>
OPLElectionData(numSeats,numBallots,numCandidates,vector<OPLParty*>)
Instantiated OPLElectionData*
OPLElectionObject.calculate_winner
calculate_seats_per_party
seat_allocated++
For each party allocated a seat(s)
vector<OPLParty*> is updated with seat allocations. allocation_data is updated
loop through vector<OPLParty*>
Loop
for each element in vector<OPLParty*>
assign_seat_winners
winner=true
When a candidate is assigned a seat
All Candidates now have updated winner attribute
generate_audit_file
add_allocation_table
add_line
auditfile.txt generated
display()
output displayed in terminal
All outputs properly created/displayed. Program terminates.

Sequence diagram for OPL Election


Election officials plan ballot stations
Parties/Candidates register for election
People vote on election day
<<System>> Election Ballot Collection System
Ballots collected and processed
Data is formatted into csv file and sent to election official
Election official executes program and inputs proper csv
Election type determined in system
[CPL]
[OPL]
Not considered in this diagram
Parse rest of election data and store it in appropriate data objects
Calculate the seats awarded to each party
Each party awards appropriate candidates a seat if available
Audit file is generated
Output is displayed to election official
Election official reviews the results
Election official informs parties of election results and winners
Election official informs media personnel of election results and winners
<<System>> Results Disemination System
Colored activity nodes represent Election Result Determination System Interaction
<<System>> Election Result Determination System

Activity Diagram for a CPL election

3.3 Design Rationale

The architecture chosen for the voting system, detailed earlier in this Software Design Document, utilizes an object-oriented approach, focusing on modularity, reusability, flexibility, and efficiency in the processing of election data. Modularity refers to the design principle of dividing a system into more manageable parts, facilitating easier debugging, testing, and modification. In the voting system, modularity is achieved by organizing the software into classes, such as ElectionData and ElectionDataParser, and subclasses, such as OPLElectionData and CPLElectionData. Each of these classes handles a specific aspect of the voting system; For example, ElectionDataParser is responsible for reading and parsing election data from files, while OPLElectionData and CPLElectionData handle the logic for open and closed party list elections. Reusability is promoted in the system through the use of abstract classes, like Party and Candidates, from which more specific classes are inherited, like OPLParty, CPLParty, OPLCandidate, and CPLCandidate. Inheritance allows for shared functionality and supports scalability by simplifying the process of adding new election configurations as the system evolves. By providing two mechanisms for entering the filename, a command-line argument and a terminal prompt, the system accommodates greater flexibility without complicating logic. Reflecting on efficiency, the design's focus on efficient data parsing and processing, facilitated by ElectionDataParser for collection and separation of information and algorithms within OPLElectionData and CPLElectionData for processing, ensures the system can handle complex election calculations and tie resolutions effectively.

4. Data Design

4.1 Data Description

This system does not use a database. Its main data storage methods are: vectors, objects, maps, and integer and string variables. The information is transformed into these data structures from the election csv file through the ElectionDataParser class. The ElectionDataParser class reads through the election csv file and returns a pointer to an ElectionData object. That ElectionData object is composed several int values representing critical election metrics, as well as a vector of pointers to Party objects which are composed of a vector of pointers to Candidate objects, all of the data needed to construct these objects is compiled by the ElectionDataParser. There are two types of Candidate and Party objects, CPLParty, CPLCandidate, OPLParty, and OPLCandidate, inherited from the Candidate and Party abstract classes.

An unordered map will be used to group together each party name with how many seats each party has won.

Some temporary data generated during the seat allocation process will be stored in a map of (string, tuple), which will be passed to the AuditLog class when it is called to create the audit file with the generate_audit_file() function in ElectionData. The AuditLog class contains functions that will be able to take in all necessary information and write it, in an organized manner, to the audit file that is created.

4.2 Data Dictionary

ElectionDataParser

Attributes	Methods
	- ElectionDataParser()
	+ create_election(string): ElectionData*
	- static tokenize_lines(string): vector<string>
	- static create_OPL_candidates(ifstream&): vector<tuple<string, OPLCandidate*>>
	- static assign_votes_to_candidates(ifstream&, vector<OPLCandidate*>): void
	- static create_OPL_parties(vector<tuple<string, OPLCandidate*>>): vector<OPLParty*>
	- static create_CPL_parties(ifstream&): vector<CPLParty*>
	- static assign_votes_to_parties(ifstream&, vector<CPLParty*>): void

<<abstract>> ElectionData

Attributes	Methods
# total_seats: int	+ ElectionData(int, int, int, int)
# total_ballots: int	+ virtual ~ElectionData()
# num_parties: int	# static break_tie(int): int
# quota: int	# calculate_seats_per_party(): unordered_map<string, int>
# audit_log: AuditLog	# virtual sort_parties(): void
# allocation_data: map<string, tuple<int,int,int>>	+ virtual calculate_winners(): void
	+ virtual generate_audit_file(): void
	+ virtual display(): void

OPLElectionData

Attributes	Methods
- parties: vector<OPLParty*>	+ OPLElectionData(int, int, int, vector<OPLParty*>)
	+ ~OPLElectionData() override
	# sort_parties() override: void
	+ calculate_winners() override: void
	+ generate_audit_file() override: void
	+ display() override: void

CPLElectionData

Attributes	Methods
- parties: vector<CPLParty*>	+ CPLElectionData(int, int, int, vector<CPLParty*>)
	+ ~CPLElectionData() override
	# sort_parties() override: void
	+ calculate_winners() override: void
	+ generate_audit_file() override: void
	+ display() override: void

<<abstract>>?Party

Attributes	Methods
# name: string	+ Party(string)
# total_votes: int	+ virtual ~Party()
+ seats_allocated: int	+ get_name(): string
	+ get_total_votes(): int
	+ set_total_votes(int): void
	+ virtual assign_seat_winners(int): void

OPLParty

Attributes	Methods
- candidates: vector<OPLCandidate*>	+ OPLParty(string, vector<OPLCandidate*>)
	+ ~OPLParty() override
	- calculate_total_votes(): void
	+ get_candidates(): vector<OPLCandidate*>
	+ assign_seat_winners() override: void

CPLParty

Attributes	Methods
- candidates: vector<CPLCandidate*>	+ CPLParty(string, vector<CPLCandidate*>)
	+ ~CPLParty() override
	+ get_candidates(): vector<CPLCandidate*>
	+ assign_seat_winners() override: void

<<abstract>>Candidate

Attributes	Methods
# name: string	+ Candidate(string)
# winner: bool	+ virtual ~Candidate()
	+ get_name(): string
	+ get_winner(): bool
	+ set_winner(bool): void
	+ virtual to_string(): string

OPLCandidate

Attributes	Methods
+ num_votes: int	+ OPLCandidate(string, int)
	+ to_string() override: string

CPLCandidate

Attributes	Methods
	+ CPLCandidate(string)
	+ to_string() override: string

AuditLog

Attributes	Methods
- log: string	+ AuditLog()
	+ add_allocation_table(vector<tuple<string, int, int, int, int, string>>): void
	+ add_line(string): void
	+ clear_log(): void
	+ write_to_file(string): void

5. COMPONENT DESIGN

5.1 ElectionDataParser

5.1.1 Constructor()

- Private constructor; not accessible.

5.1.2 create_election(string): ElectionData*

- Open the file at the specified URL location in the parameter, send the user an error and terminate the program if it cannot be opened. Read the header, and determine the type of election. Read through the file, totaling up the number of votes per party, and number of votes per candidate, if applicable. Create an instance of an implementation of ElectionData corresponding to the proper election type and return it. Intermediate objects such as Party and Candidate implementations will need to be temporarily created to properly construct this object.

5.1.3 static tokenize_lines(string): vector<string>

- Helper function to parse comma separated strings and return a vector of values

5.1.4 static create_OPL_candidates(ifstream&): vector<tuple<string, OPLCandidate*>>

- Receives the data stream, and iterates through the candidate lines of the data, returning a vector of (party, candidateObject) tuples

5.1.5 static assign_votes_to_candidates(ifstream&, vector<OPLCandidate*>): void

- Receives the data stream, and iterates through the vote ballot lines of data, incrementing the appropriate votes attribute in the corresponding OPLCandidate vector

5.1.6 static create_OPL_parties(vector<tuple<string, OPLCandidate*>>): vector<OPLParty*>

- Consolidates the OPLCandidates into their respective party using the vector<tuple<string, OPLCandidate*>> data structure created in create_OPL_candidates, pushes the created parties onto a vector of OPLParty

5.1.7 static create_CPL_parties(ifstream&): vector<CPLParty*>

- Loops through the party/candidate lines of ifstream&, instantiating a CPLParty* with a party name and CPLCandidate object vector generated from the data on each line, pushes that party onto a vector<CPLParty*> and returns this vector at the end

5.1.8 static assign_votes_to_parties(ifstream&, vector<CPLParty*>): void

- Receives the data stream, and iterates through the vote ballot lines of data, incrementing the appropriate total_votes attribute in the corresponding CPLParty vector

5.2 ElectionData

5.2.1 Constructor(int, int, int, int)

- Receives parameters for the settings of (in-order): \`total_seats\`, \`total_ballots\`, \`num_parties\`, \`quota\`. Sets \`audit_log\` to a default AuditLog. Sets \`allocation_data\` to an empty map.

5.2.2 virtual Destructor()

- Destructor to be overridden by implementing classes.

5.2.3 static break_tie(int): int

- Receives a number n for which a random number between 0 and n - 1 is returned. Seeds a random number generator based on the current time. Skips the first 1001 random numbers from \`rand()\`. Returns rand() mod n.

5.2.4 calculate_seats_per_party(): unordered_map<string, int>

- Assigns \`total_seats\` seats to each party. First determines the number of votes required for a guaranteed seat by dividing the total votes by \`total_seats\`. Each party in the ElectionData is assigned the number of seats they are guaranteed. Each remaining seat is repeatedly assigned to the next party with the greatest remainder of votes not accounted for by the guaranteed seats. The result of this is stored in a map mapping the party names to the number of seats they are awarded. This map is returned.

5.2.5 virtual sort_parties(): void

- Sorts the \`parties\` vector in descending order based on number of votes. To be implemented.

5.2.6 virtual calculate_winners(): void

- This is the encapsulating function for this class, internally makes all necessary calls to calculate the winners of the election and produce the required outputs. To be implemented.

5.2.7 virtual generate_audit_file(): void

- Outputs a .txt file with a report about the results of the election. To be implemented.

5.2.8 virtual display(): void

- Outputs to the terminal the winners of each election seat. To be implemented.

5.3 CPLElectionData

5.3.1 Constructor(int, int, int, vector<CPLParty*>)

- Arguments are used to set values for `total_seats`, `total_ballots`, `num_parties`, and `parties` (in order). `quota` is initialized to `ceil(total_ballots / total_seats)`.`audit_log` is initialized to the default `AuditLog`. `allocation_data` is initialized to an empty map.

5.3.2 Destructor() override

- Calls destructor on all `CPLParty`s in `parties`

5.3.3 sort_parties() override: void

- Sorts the `parties` vector in descending order based on number of votes.

5.3.4 calculate_winners() override: void

- Sets the `winner` flag in all aggregated candidates who won a seat to `true`. First calls `seats_per_party()`, iterates through the resulting map, and calls `assign_seat_winners(int)` on each party in `parties` with the same name, and their respective number of seats won in the map. For each party, maps their name to their total votes, first allocation seats, and second allocation seats.

5.3.5 generate_audit_file() override: void

- Sends the data still needed to be written to the log to `audit_log`. This includes a call to `add_allocation_table()` using data from `allocation_data`, for which the missing values needed are calculated from the existing values stored. Also includes a breakdown of all candidates by party with their respective numbers of votes, and a designation for those who are winners. Finishes by calling the `write_to_file()` method of `audit_log`.

5.3.6 display() override: void

- Outputs to the terminal the names of all parties, their total votes, the members within those parties, and an indicator next to the names of the winners of seats.

5.4 OPLElectionData

5.4.1 Constructor(int, int, int, vector<OPLParty*>)

- Arguments are used to set values for `total_seats`, `total_ballots`, `num_parties`, and `parties` (in order). `quota` is initialized to `ceil(total_ballots / total_seats)`.`audit_log` is initialized to the default `AuditLog`. `allocation_data` is initialized to an empty map.

5.4.2 Destructor() override

- Calls destructor on all `OPLParty`s in `parties`

5.4.3 sort_parties() override: void

- Sorts the `parties` vector in descending order based on number of votes.

5.4.4 calculate_winners() override: void

- Sets the `winner` flag in all aggregated candidates who won a seat to `true`. First calls `seats_per_party()`, iterates through the resulting map, and calls `assign_seat_winners(int)` on each party in `parties` with the same name, and their respective number of seats won in the map. For each party, maps their name to their total votes, first allocation seats, and second allocation seats.

5.4.5 generate_audit_file() override: void

- Sends the data still needed to be written to the log to `audit_log`. This includes a call to `add_allocation_table()` using data from `allocation_data`, for which the missing values needed are calculated from the existing values stored. Also includes a breakdown of all candidates by party with their respective numbers of votes, and a designation for those who are winners. Finishes by calling the `write_to_file()` method of `audit_log`.

5.4.6 display() override: void

- Outputs to the terminal the names of all parties, their total votes, the members within those parties, and an indicator next to the names of the winners of seats.

5.5 Party

5.5.1 Constructor(string)

- Sets `name` to the string parameter

5.5.2 virtual Destructor()

- Destructor to be implemented.

5.5.3 get_name(): string

- Getter for `name`

5.5.4 get_total_votes(): int

- Getter for `total_votes`

5.5.5 set_total_votes(int): void

- Setter for `total_votes`

5.5.6 virtual assign_seat_winners(int): void

- Assigns the winning candidates’ `winner` field to `true`. To be implemented.

5.6 CPLParty

5.6.1 Constructor(string, vector<CPLCandidate*>)

- Sends the first parameter (name) to the super-constructor. Sets `candidates` equal to the 2nd parameter.

5.6.2 Destructor() override

- Deletes all `CPLCandidate`s in `candidates`.

5.6.3 get_candidates(): vector<CPLCandidate*>

- Getter for `candidates`.

5.6.4 assign_seat_winners() override: void

- Picks the first `seats_allocated` `CPLCandidate`s in `candidates` to assign `winner` to `true` for.

5.7 OPLParty

5.7.1 Constructor(string, vector<CPLCandidate*>)

- Sends the first parameter (name) to the super-constructor. Sets `candidates` equal to the 2nd parameter. Calls calculate_total_votes() to set the value of `total_votes`.

5.7.2 Destructor() override

- Deletes all `OPLCandidate`s in `candidates`.

5.7.3 calculate_total_votes(): void

- Sets `total_votes` equal to the sum of all of the values of the `num_votes` property of all `OPLCandidate`s in `candidates`

5.7.4 get_candidates(): vector<CPLCandidate*>

- Getter for `candidates`.

5.7.5 assign_seat_winners() override: void

- Picks the `seats_allocated` `OPLCandidate`s in `candidates` with the most votes to assign `winner` to `true` for. If there is a tie for more candidates than seats left to be assigned, the tie is broken by calling `ElectionData.break_tie(int)`. The tie will be documented in the audit report by using `add_line(string)`.

5.8 Candidate

5.8.1 Constructor(string)

- Sets `name` to the string parameter, and `winner` to false.

5.8.2 virtual Destructor()

- Destructor to be overridden by implementing classes.

5.8.3 get_name(): string

- Getter for `name`.

5.8.4 get_winner(): bool

- Getter for `winner’.

5.8.5 set_winner(bool): void

- Setter for `winner`.

5.8.6 virtual to_string(): string

- Returns string representation for the Candidate; to be implemented.

5.9 CPLCandidate

5.9.1 Constructor(string)

- Sends 1st string parameter (name) to `Candidate` super-constructor.

5.9.2 to_string() override: string

- Returns a string in the format of "<name> - Winner" if `winner` is true, else "<name>"

5.10 OPLCandidate

5.10.1 Constructor(string, int)

- Sends 1st string parameter (name) to `Candidate` super-constructor, sets `num_votes` equal to 2nd int parameter

5.10.2 to_string() override: string

- Returns a string in the format of "<name> - Winner" if `winner` is true, else "<name>"

5.11 AuditLog

5.11.1 Constructor()

- Initializes `log` to an empty string.

5.11.2 add_allocation_table(vector<tuple<string, int, int, int, int, string>>): void

- Appends a formatted table and a newline to the log. Parameters are a vector of rows in the table. Each row is represented as a tuple of: (party name, votes, first allocation seats, remaining votes, second allocation of seats, % of vote to % of sets [formatted as “xx% / xx%”])

5.11.3 add_line(string): void

- Appends the string argument and a newline to the log.

5.11.4 clear_log(): void

- Clears all content in the log.

5.11.5 write_to_file(string): void

- Creates a file with the name “audit_<date, time>.txt” in the directory specified by the string parameter.

6. Human Interface Design

6.1 Overview of User Interface

The voting system is designed to provide an intuitive and efficient user interface for two groups of users: Programmers and Testers, and Election Officials. Using a command-line interface, the system is designed to facilitate election functionality and ease of use.

Programmers and Testers:

The command-line interface provides an intuitive experience to Programmers and Testers by allowing direct input of election file names as command-line arguments. These users can effectively navigate and operate the system through command-line arguments, enabling straightforward execution of the voting system, which is ideal for testing and development.

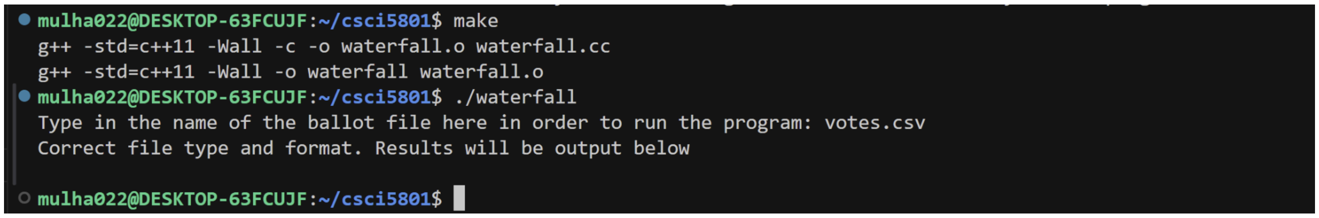
Election Officials:

Election Officials interact with the voting system without a command-line argument. Upon starting the program, election officials will be able to initiate the program from the command-line with no arguments, then enter the filename into a terminal prompt from the program.

6.2 Screen Images

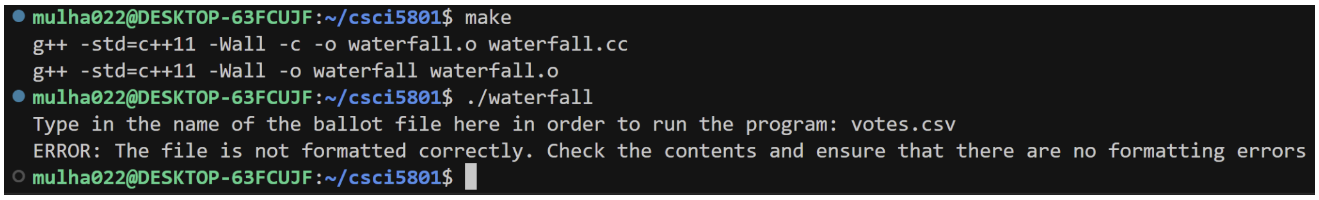
Below are basic examples of what the user will see when interacting with the system:

Correct input:

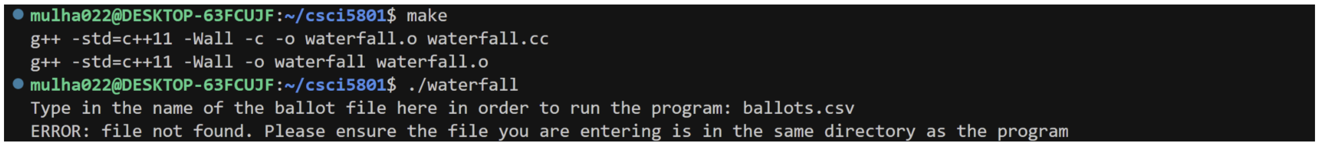

```
● mulha022@DESKTOP-63FCUJF:~/csci5801$ make
  g++ -std=c++11 -Wall -c -o waterfall.o waterfall.cc
  g++ -std=c++11 -Wall -o waterfall waterfall.o
● mulha022@DESKTOP-63FCUJF:~/csci5801$ ./waterfall
  Type in the name of the ballot file here in order to run the program: votes.csv
  Correct file type and format. Results will be output below

○ mulha022@DESKTOP-63FCUJF:~/csci5801$ 
```

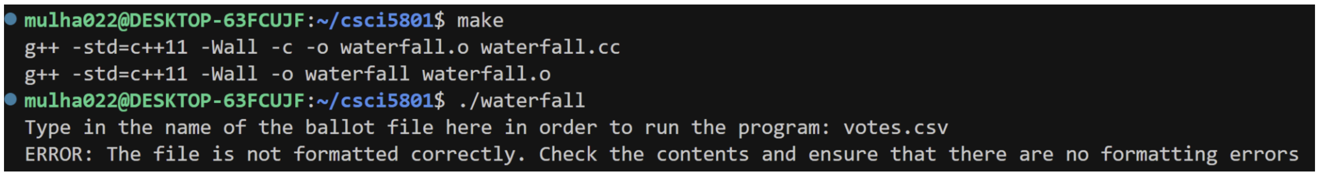
Incorrect file type:


```
● mulha022@DESKTOP-63FCUJF:~/csci5801$ make
  g++ -std=c++11 -Wall -c -o waterfall.o waterfall.cc
  g++ -std=c++11 -Wall -o waterfall waterfall.o
● mulha022@DESKTOP-63FCUJF:~/csci5801$ ./waterfall
  Type in the name of the ballot file here in order to run the program: votes.csv
  ERROR: The file is not formatted correctly. Check the contents and ensure that there are no formatting errors
○ mulha022@DESKTOP-63FCUJF:~/csci5801$ 
```

File not located in the same directory as the System:


```
● mulha022@DESKTOP-63FCUJF:~/csci5801$ make
  g++ -std=c++11 -Wall -c -o waterfall.o waterfall.cc
  g++ -std=c++11 -Wall -o waterfall waterfall.o
● mulha022@DESKTOP-63FCUJF:~/csci5801$ ./waterfall
  Type in the name of the ballot file here in order to run the program: ballots.csv
  ERROR: file not found. Please ensure the file you are entering is in the same directory as the program
```

File not formatted correctly:


```
● mulha022@DESKTOP-63FCUJF:~/csci5801$ make
  g++ -std=c++11 -Wall -c -o waterfall.o waterfall.cc
  g++ -std=c++11 -Wall -o waterfall waterfall.o
● mulha022@DESKTOP-63FCUJF:~/csci5801$ ./waterfall
  Type in the name of the ballot file here in order to run the program: votes.csv
  ERROR: The file is not formatted correctly. Check the contents and ensure that there are no formatting errors
```

6.3 Screen Objects and Actions

There is no specific objects relating to the screen. Since we are only provide outputs via terminal output and .txt file, the screen UI will solely consist of the terminal and text-editing software, which will be determined by the user’s preferences.

7. Requirements Matrix

Functional Requirement(s)	System Component(s)
4.1 - Read in Filename as Command Line Argument	N/A (main)
4.2 - Read in Filename from a Terminal Prompt	5.1.2, 5.2.1
4.3 - Resolving a Tie in the Election with a Fair Coin Flip	5.2.3
4.4 - Parsing of OPL Data and Loading into Data Structures	5.1.3 - 5.1.6
4.5 - Parsing of CPL Data and Loading into Data Structures	5.1.3, 5.1.7 - 5.1.8
4.6 - Performing Seat Allocation for OPL and CPL Elections	5.2.4, 5.3.4, 5.4.4
4.7 - Choosing of OPL Seat Winners	5.7.5
4.8 - Choosing of CPL Seat Winners	5.6.4
4.9 - Generation and Output of Audit Report	5.2.7, 5.3.5, 5.4.5, 5.11.1 - 5.11.5
4.10 - Display of Election Results	5.2.8, 5.3.6