Team 7

Voting System

Software Design Document

Names:

Chork Hieng (hieng001)
Ziyoda Mamatkulova (mamat007)
Andrei Anicescu (anice002)
Mohamed Ali (ali00429)

Date: Mar 1, 2024

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This document describes the features and architecture of the Voting System that processes ballots for the election. This system is intended to be used by the election officials to process votes and produce for the election.

Additionally, this document is used for developers to implement and modify this system. It is also useful for development and maintenance on the Voting system.

1.2 Scope

This document serves as a blueprint for the Voting System that includes component and class diagrams, sequence diagrams, activities, and system diagrams for the development and implementation.

The system processes ballots and allocates seats for all parties (and candidates) in the election. The goal is to have one system that can handle both a CPL and OPL election. The system will fairly handle ties and produce an audit file that shows the results for any election. Our goal is to create a fair vote-counting system that speeds up the allocation of seats.

1.3 Overview

This document is structured by sections as described below:

1. **Introduction:** This section provides an introduction to the system and a brief description of the product, an initial overview of the system, and the intended audience of the document as well as the system.
2. **System Overview:** In this section, we provide the descriptions for the vote counting processes, election types, voting algorithms, and edge case handling (i.e. tie case).
3. **System Architecture:** This section focuses heavily on our UML class diagram, activities diagram, and sequence diagram. We use these diagrams to illustrate states and interactions within the system.

4. **Data Design:** This section describes how the system processes and handles data storage.
5. **Component Design:** This section provides more in-depth details of each component of the system such as class components and methods for each component within the system.
6. **Human Interface Design:** This section provides more information about the user interface and how users interact with the system.
7. **Requirements Matrix:** This section provides detailed information about the system architecture and design and how it matches with requirements in SRS documentation.
8. **Appendices:** Further sources, supplemental reading, and references are provided in this section.

1.4 Reference Material

This document is created based on the Software Design Document (SDD) Template.

1.5 Definitions and Acronyms

Keywords	Abbreviation	Definitions
Voting System		Software application created to process votes and get results for the election.
Closed Party Listing	CPL	An election type where voters can choose only the party they want to elect.
Open Party Listing	OPL	An election type where voters can choose the candidate within a party that they want to elect.
Tie Breaker		An algorithm that handles ties between two or more parties or candidates that have the same number of votes.

Software Design Document	SDD	Documentation describing the design and architecture of the system.
Unified Modeling Language	UML	Symbols and flow charts represent activities and diagrams about the system.
UML Class Diagram		Diagrams describing classes and their components in the system.
SDD Sequence Diagram		Diagram describing steps or sequence of action within the system.
SDD Activity Diagram		Diagram describing the main actions a system takes to produce its results.

2. SYSTEM OVERVIEW

This system consists of the following functionalities:

- **File input:** Users (election officials) can type a filename into the program, which contains all election information. It is required that the ballot file be in the same directory as the main program.
- **File handling:** This program does not handle the format of the file, which means that it's assumed that the file is formatted correctly when the program reads it. The program will determine if a file exists with the given filename in the current directory, otherwise, it keeps asking for a filename.
- **Program termination:** The program terminates when it has successfully produced the audit file and printed the results to the terminal. The user can interrupt the program before it completes.
- **Process votes:** The system will determine the type of election based on the file's header. There are two types of elections used in this system, and the system will handle both types differently.
- **Tie Breaker:** The system can handle any ties when two or more parties/candidates have the same amount of votes.
- **Audit file:** The program will produce an audit file that shows election type, numbers of parties, ballots, and seats. It will also show the names of candidates and parties who won seats. There will also be a table to show how votes/seats were allocated.
- **Display results:** Results of the election will be displayed on a screen or terminal after all votes have been processed.
- **Operating system compatibility:** It's required that the program is executed on a computer that uses the Linux Operating System (U of MN CSE Linux Lab Machines).
- **Programming language:** The system is implemented in C++.
- **Runtime constraints:** The system will be able to run 100,000 ballots in under 4 minutes.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The responsibilities will be divided among several subsystems to ensure ease of maintenance. The primary subsystems will include the input handler, vote counter and seat allocator, tiebreaker, and results management. These subsystems will work alongside different aspects of the voting process.

Below is the activity diagram for the CPL election. More information about the activity diagram is also provided in the **SDD Activity Diagram** file.

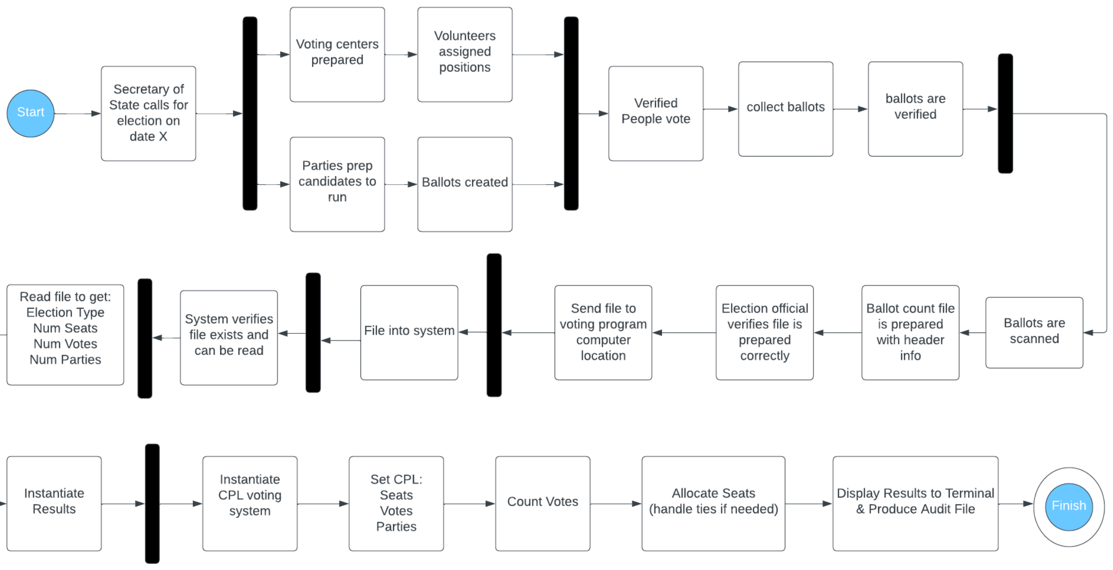

Start
Secretary of State calls for election on date X
Voting centers prepared
Volunteers assigned positions
Parties prep candidates to run
Ballots created
Verified People vote
collect ballots
ballots are verified
Ballots are scanned
Ballot count file is prepared with header info
Election official verifies file is prepared correctly
Send file to voting program computer location
File into system
System verifies file exists and can be read
Read file to get: Election Type Num Seats Num Votes Num Parties
Instantiate Results
Instantiate CPL voting system
Set CPL: Seats Votes Parties
Count Votes
Allocate Seats (handle ties if needed)
Display Results to Terminal & Produce Audit File
Finish

Figure 01: Activity Diagram

3.2 Decomposition Description

Below is the UML class diagram representing how each class is connected to another one. More details about UML class diagram and its data structures are available in the **SDD UML Class Diagram** file.

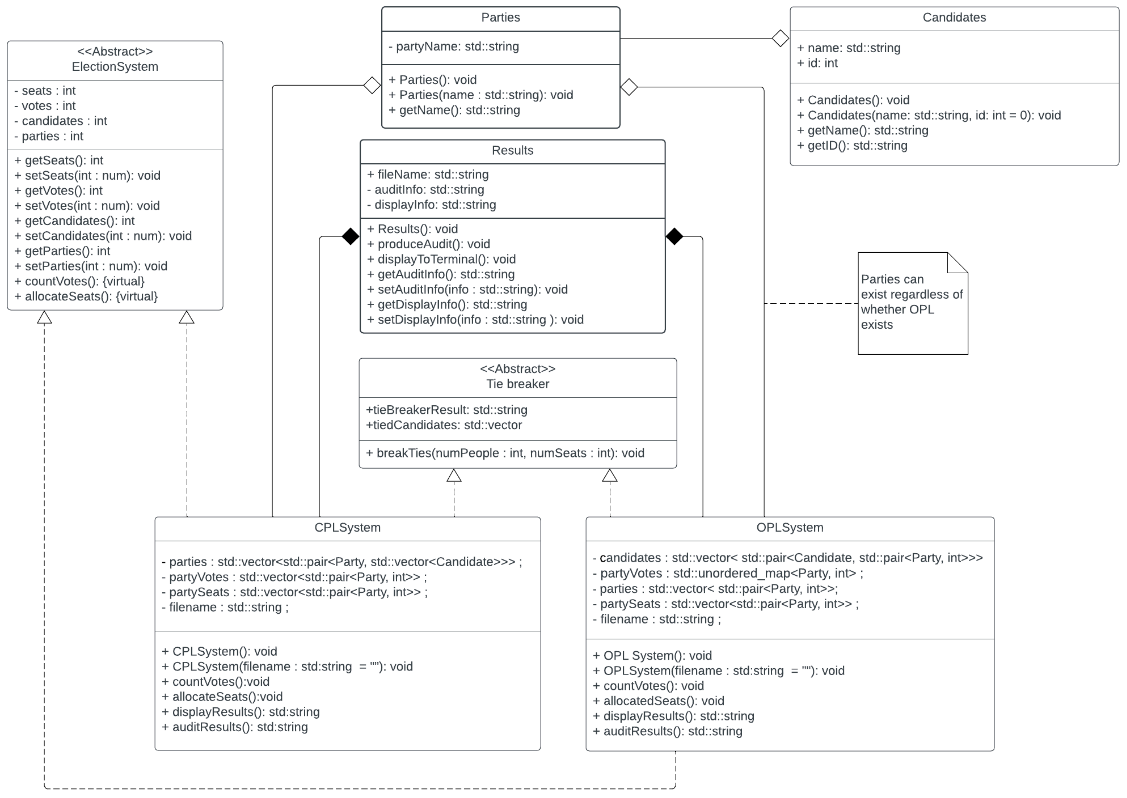

Parties
- partyName: std::string
+ Parties(): void
+ Parties(name : std::string): void
+ getName(): std::string
Candidates
+ name: std::string
+ id: int
+ Candidates(): void
+ Candidates(name: std::string, id: int = 0): void
+ getName(): std::string
+ getID(): std::string
<<Abstract>>
ElectionSystem
- seats : int
- votes : int
- candidates : int
- parties : int
+ getSeats(): int
+ setSeats(int : num): void
+ getVotes(): int
+ setVotes(int : num): void
+ getCandidates(): int
+ setCandidates(int : num): void
+ getParties(): int
+ setParties(int : num): void
+ countVotes(): {virtual}
+ allocateSeats(): {virtual}
Results
+ fileName: std::string
- auditInfo: std::string
- displayInfo: std::string
+ Results(): void
+ produceAudit(): void
+ displayToTerminal(): void
+ getAuditInfo(): std::string
+ setAuditInfo(info : std::string): void
+ getDisplayInfo(): std::string
+ setDisplayInfo(info : std::string): void
Parties can exist regardless of whether OPL exists
<<Abstract>>
Tie breaker
+tieBreakerResult: std::string
+tiedCandidates: std::vector
+ breakTies(numPeople : int, numSeats : int): void
CPLSystem
- parties : std::vector<std::pair<Party, std::vector<Candidate>>> ;
- partyVotes : std::vector<std::pair<Party, int>> ;
- partySeats : std::vector<std::pair<Party, int>> ;
- filename : std::string ;
+ CPLSystem(): void
+ CPLSystem(filename : std:string = ""): void
+ countVotes():void
+ allocateSeats():void
+ displayResults(): std:string
+ auditResults(): std:string
OPLSystem
- candidates : std::vector< std::pair<Candidate, std::pair<Party, int>>>
- partyVotes : std::unordered_map<Party, int> ;
- parties : std::vector< std::pair<Party, int>>;
- partySeats : std::vector<std::pair<Party, int>> ;
- filename : std::string ;
+ OPL System(): void
+ OPLSystem(filename : std:string = ""): void
+ countVotes(): void
+ allocatedSeats(): void
+ displayResults(): std::string
+ auditResults(): std::string

Figure 02: UML Class Diagram

The Sequence diagram below provides the states and sequences of action within the system. For more details, please refer to the **SDD Sequence Diagram** file.

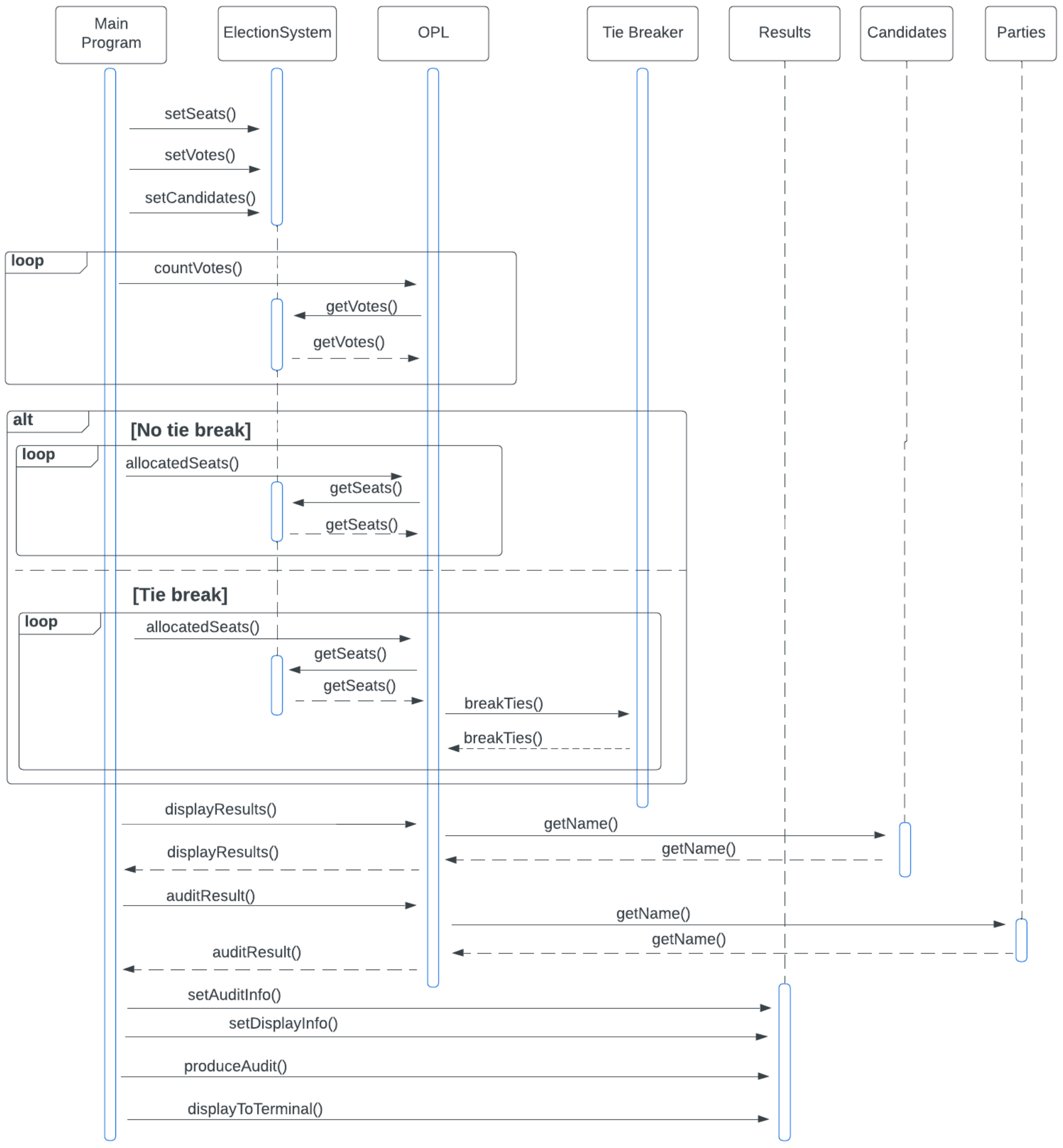


Figure 03: Sequence Diagram

3.3 Design Rationale

The architecture was chosen for extensibility, reusability, and readability. Other similar election types should be easily incorporated into our design. This approach makes it easier to manage the complex functionalities in an electoral process.

4. DATA DESIGN

4.1 Data Description

We use classes to represent entities like Candidates and Parties. We use vectors that have combinations of Parties or Candidates and their votes. This allows us to easily sort the data. More information about this can be found in the **SDD UML Class Diagram** file. We keep primary election info like the number of seats, candidates, parties, and ballots inside of each election type, for ease of access during computation.

We store our results inside a string. This string will be passed to the Results class so it can produce the audit file and print results to the terminal.

4.2 Data Dictionary

Candidates:
- Type: Class
- Attributes: name, id
- Methods: Candidates(),Candidates(name, id:default):constructor, getName(), getID()

CPLSystem:
- Type: Class
- Attributes: parties, partyVotes, partySeats, filename
- Methods: CPLSystem(), CPLSystem(): constructor, countVotes(), allocateSeats(), displayResults(), auditContent()

ElectionSystem:
- Type: Abstract Class
- Attributes: seats, votes, candidates, parties
- Methods: setSeats(), getSeats()

OPLSystem:
- Type: Class
- Attributes: candidates, parties, partyVotes, partySeats, filename
- Methods: countVotes(), allocateSeats(), displayResults(), auditContent()

Parties:

- Type: Class
- Attributes: partyName
- Methods: Parties(), getName()

Results:

- Type: Class
- Attributes: fileName, auditInfo, displayInfo
- Methods: Results(), produceAudit(), displayToTerminal(), getAuditInfo(), setAuditInfo(), getDisplayInfo(), setDisplayInfo()

TieBreaker:

- Type: Abstract class
- Attributes: tieBreakerResult, tiedCandidates
- Methods: breakTies()

5. COMPONENT DESIGN

In this section, we will provide more details about each component of the system with descriptions for each method of the component.

5.1 Candidates Class

- **Default constructor:**
```
Candidates() {
        // Create a new instance of Candidates class
}
```

- **Constructor with parameters:**
```
Candidates(name: std::string, id: int = 0) {
        // Create a new instance of Candidates class
        // Set name of the candidate
        // Id will be default value (unique id)
}
```

- **Get candidate’s name:**
```
getName() {
        // Return candidate’s name as a string
}
```

- **Get candidate’s id:**
```
getID() {
        // Return candidate’s id as a string
}
```

5.2 Parties Class

- **Default constructor:**
```
Parties() {
        // Create a new instance of Parties class
}
```

- **Constructor with parameter:**
```
Parties(name: std::string) {
        // Create a new instance of Parties class
        // Assign Party type (name) to the party
}
```

- **Get party’s name:**
```
getName() {
        // Return party’s name as a string
}
```

5.3 ElectionSystem Class (Abstract Class)

- **Get total number of seats:**
```
GetSeats() {
        // Return number of seats for the election
}
```

- **Set number of seats:**
```
setSeats(num: int) {
        // Set number of seats for the election
}
```

- **Get number of votes:**
```
getVotes() {
        // Return the number of votes for the election
}
```

- **Set number of votes:**
```
setVotes() {
        // Set number of votes for the election
}
```

- **Get number of candidates:**
```
getCadidates() {
        // Return number of candidates
}
```

- **Set number of candidates:**
```
setCandidates(num: int) {
        // Set number of candidates
}
```

- **Get number of parties:**
```
getParties() {
        // Return number of parties
}
```

- **Set number of parties:**
```
setParties(num: int) {
        // Set number of parties
}
```

- **Count number of votes:**
```
countVotes() {
        // Count number of votes
        // This is a virtual function
}
```

- **Allocate seats:**
```
allocateSeats() {
        // Allocate seats for parties
        // This is a virtual function
}
```

5.4 CPLSystem Class

- **Default constructor:**
```
CPLSystem() {
        // default constructor
}
```

- **Constructor with parameter:**
```
CPLSystem(filename: std::string = “ ”) {
        // Constructor with filename parameter
}
```

- **Count number of votes:**
```
countVotes() {
        // Count number of votes
        // Implement from ElectionSystem Class
}
```

- **Allocate seats:**
```
allocateSeats() {
        // Allocate seats for parties
        // Implement from ElectionSystem Class
        // Algorithm for seat allocation is also implemented
}
```

- **Display results of the election:**
```
displayResults() {
        // Display result of the election as string
}
```

- **Produce audit for the election:**
```
auditResults() {
        // Produce audit results for the election as string
}
```

5.5 OPLSystem Class

- **Default constructor:**
  ```
  OPLSystem() {
          // default constructor
  }
  ```

- **Constructor with parameter:**
  ```
  OPLSystem(filename: std::string = “ ”) {
          // Constructor with filename parameter
  }
  ```

- **Count number of votes:**
  ```
  countVotes() {
          // Count number of votes
          // Implement from ElectionSystem Class
  }
  ```

- **Allocate seats:**
  ```
  allocateSeats() {
          // Allocate seats for parties
          // Implement from ElectionSystem Class
          // Algorithm for seat allocation is also implemented
  }
  ```

- **Display results of the election:**
  ```
  displayResults() {
          // Display result of the election as string
  }
  ```

- **Produce audit for the election:**
  ```
  auditResults() {
          // Produce audit results for the election as string
  }
  ```

}

5.6 Tie Breaker Class (Abstract Class)

- **Run algorithm to break tie case:**

```
breakTies(numPeople: int, numSeats: int) {
        // Run tie breaker algorithm based on number of seats available
        // and number of parties that have same votes (highest)
}
```

5.7 Results Class

- **Default constructor:**

```
Results() {
        // Default constructor
}
```

- **Produce audit file:**

```
produceAudit() {
        // Create an audit file for the election
}
```

- **Display results to the terminal:**

```
displayToTerminal() {
        // Display the results to the terminal
}
```

- **Get audit information:**

```
getAuditInfo() {
        // Return information about the election for an audit file
}
```

- **Set audit information:**

```
setAuditInfo(info: std::string) {
        // Set information about the election for an audit file
}
```

- **Get information about the results of the election for display:**

```
getDisplayInfo() {
        // Return results of the election for display
        // as string
}
```

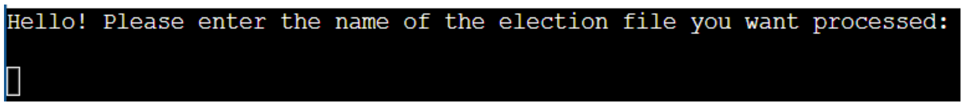
- **Set information about the results of the election for display:**
```
setDisplayInfo(info: std::string) {
        // Set the results of the election for display
}
```

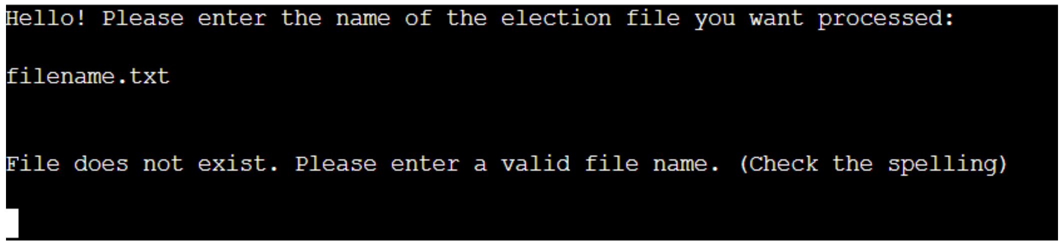
6. HUMAN INTERFACE DESIGN

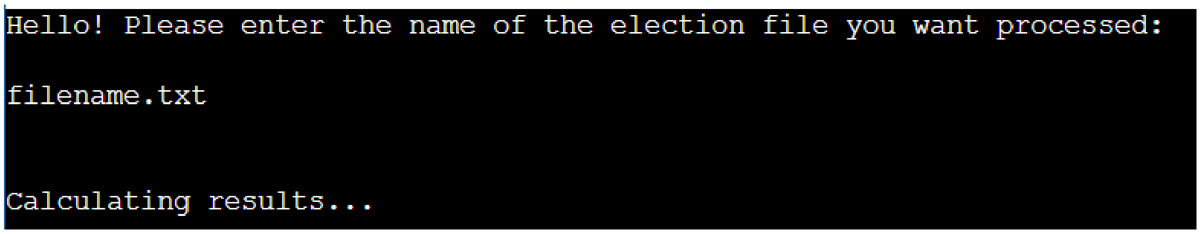
6.1 Overview of User Interface

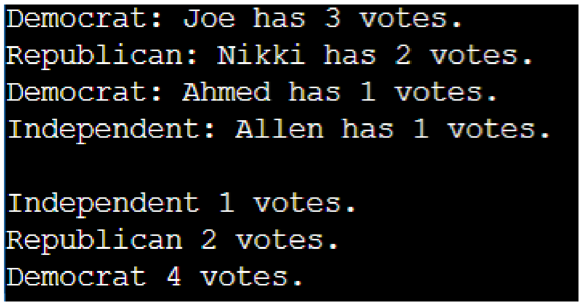
The voting system interface uses a command line interface. The program will have clear instructions to guide users through the process of inputting the election file.

6. 2 Images of Screen

1. 
```
Hello! Please enter the name of the election file you want processed:
```

2. 
```
Hello! Please enter the name of the election file you want processed:

filename.txt


File does not exist. Please enter a valid file name. (Check the spelling)
```

3. 
```
Hello! Please enter the name of the election file you want processed:

filename.txt


Calculating results...
```

4. 
```
Democrat: Joe has 3 votes.
Republican: Nikki has 2 votes.
Democrat: Ahmed has 1 votes.
Independent: Allen has 1 votes.

Independent 1 votes.
Republican 2 votes.
Democrat 4 votes.
```

6.3 Screen Objects and Actions

1. The program will first ask the user to input in a file.
2. If they misspell the file name, or the file they typed in cannot be found, they will be prompted to input the filename until the file can be found.
3. If the file is found in the programs directory, voting count starts.
4. The results will be printed to the terminal when the vote counting and seat allocation is complete. Image 4 is a demonstration, not a depiction of the output of the result of the actual election system program.

7. REQUIREMENTS MATRIX

Functional Requirement ID	**Data Structure**
UC_002 - Filename prompt. The program is run with the filename as an argument in the command line.	The filename prompt is implemented in the main function of the program.
UC_003 - Filename prompt. Program (is running, and) asks the user to type in a filename.	The filename prompt is implemented in the main function of the program.
UC_004 - Extract information from a file.	The extracting information from a file is implemented in the main function of the program.
UC_005 - Process header file.	The processing header file is implemented in the main function of the program.
UC_006 - CPL	The CPL class is created to handle the CPL voting process.
UC_007 - OPL	The OPL class is created to handle the OPL voting process.
UC_008 - Fair process for determining a tie	The tiebreaker class is created for handling ties fairly.
UC_009 - Produce an audit file	The results class is created to produce an audit file.
UC_010 - Display to terminal	The results class is created to display the outputs to terminal

8. APPENDICES

- For more information about SRS functional requirements and requirements matrix in **section 7**, please refer to full **SRS** documentation here.
- Information about UML Class Diagrams can be found here.
- Full description of the Sequence Diagrams can be found here.
- Information about Activity Diagrams can be found here.