

FINAL REPORT

Group Name: Amadeus
Project Title: Election-Voter
Group: Lab 2 Group 1

Group Member:
Sze XiJie (101222928) (Leader)
Chin Kai Lun (101221815)
Ooi Yuk Quan (101230222)
Chen Jun Yao (101222889)



Link to Presentation:
<https://youtu.be/IIZ-1iOvZ4s>

1 Table of Contents

1	Project Description.....	4
2	Project Goals and Objectives	4
2.1	Project Goals	4
2.2	Project Objectives	4
3	Problem Statement.....	4
4	Design Concept	5
5	Compatibility of Design.....	7
5.1	How data is Stored	7
5.2	Code Explanation	7
5.2.1	Candidate Program	7
5.2.2	Voter Program.....	15
5.3	Program Walkthrough.....	38
5.3.1	Candidate Program	38
5.3.2	Voter Program.....	39
5.4	Test Plan.....	42
5.4.1	Candidate Program	42
5.4.2	Voter Program.....	43
6	Project Outcomes	47
6.1.1	Candidate Program	47
6.1.2	Voter Program.....	47
	Figure 1: Flowchart for Candidate Program.....	5
	Figure 2: Flowchart for Voter Program	6
	Figure 3: Program Description for Candidate Program	7
	Figure 4: divisioncategorize function	8
	Figure 5: viewcandidateinfo function	9
	Figure 6: createcandidatedb function.....	9
	Figure 7: createcandidatetable function	9
	Figure 8: insertdivisiondata function	10
	Figure 9: selectcandidatedata function	10
	Figure 10: callcandidatedata function	11
	Figure 11: usedpartyname function.....	11
	Figure 12: usedcandidateid function	12
	Figure 13: divisionsizecheck function	12
	Figure 14: partycheck function	13
	Figure 15: mainmenu function.....	13
	Figure 16: mainselectionmenu function	14
	Figure 17: main function.....	14
	Figure 18: Program Description for Voter Program.....	15

Figure 19: viewcandidaeinfo function	15
Figure 20: VoterDatabase function.....	16
Figure 21: createcandidatedb function.....	16
Figure 22: createcandidatetable function	16
Figure 23: createvoterdb function	17
Figure 24:createvotertable function	17
Figure 25: insertvoterdata function	19
Figure 26: viewallcandidateinfo function	19
Figure 27: selectalldivision function	19
Figure 28: viewdivision function	20
Figure 29: selectbasedondivision function	20
Figure 30: viewparty function	20
Figure 31: selectbasedonparty function	21
Figure 32: selectcandidatadata function	21
Figure 33: selectvoterdata function.....	21
Figure 34: minmaxvotealldivision function.....	22
Figure 35: maxvotecount function.....	22
Figure 36: minvotecount function	23
Figure 37: counttotalvote function	23
Figure 38: callcandidatedata function	24
Figure 39: callvoterdata function.....	24
Figure 40: callvotedata function	24
Figure 41: usedpartyname function.....	25
Figure 42: checkvoterstatus function	25
Figure 43: usedcandidateidbasedondivision function	26
Figure 44: usedcandidateid function	26
Figure 45: divisionsizecheck function	27
Figure 46: partycheck function	27
Figure 47: checkvoterdivision function.....	28
Figure 48: candidaeinfovoteselection function	28
Figure 49: storevoterid function	29
Figure 50: voterstatusupdate function	29
Figure 51: votecountupdate function	29
Figure 52: candidatevotebasedonid function	30
Figure 53: VoteCandidate function.....	31
Figure 54: selectcandidatedata function	31
Figure 55: ViewResultMenu function	32
Figure 56: ResultSelectionMenu function.....	32
Figure 57: ViewingCandidateMenu function	33
Figure 58: VoterMainMenu function	33
Figure 59:ViewingSelectionMenu function.....	36
Figure 60: MainSelectionMenu function	37
Figure 61: main function	37
Figure 62: Main Menu UI For Candidate Program.....	38
Figure 63: View Candidate UI.....	38
Figure 64: Add Candidate UI	39
Figure 65: Exit Program UI	39
Figure 66: Main Menu UI For Voter Program	39

Figure 67: Viewing Candidates Menu UI.....	40
Figure 68: Viewing Candidates Based on Division UI.....	40
Figure 69: Viewing Candidate Based on party UI.....	40
Figure 70: Register Voter Menu UI	41
Figure 71: Vote Menu UI.....	41
Figure 72: View Voting Results and Summary UI.....	42
Figure 73: View Voting Results and Summary Based on Division Menu UI	42
Figure 74: Exiting Program UI	42
Table 1: Valid Input for Main Menu in Candidate Program.....	42
Table 2: Invalid Input for Main Menu in Candidate Program	43
Table 3: Valid Input for Add Candidate Menu	43
Table 4: Invalid Input for Add Candidate Menu.....	43
Table 5: Valid Input for Main Menu in Voter Program	43
Table 6: Invalid Input for Main Menu in Voter Program	43
Table 7: Valid Input for View Candidate Menu.....	44
Table 8: Invalid Input for View Candidate Menu	44
Table 7: Valid Input for View Candidate in Specific Division Menu	44
Table 8: Invalid Input for View Candidate in Specific Division Menu	44
Table 7: Valid Input for View Candidate Based on Party Menu.....	44
Table 8: Invalid Input for View Candidate Based on Party Menu	45
Table 7: Valid Input for Register Voter Menu	45
Table 8: Invalid Input for Register Voter Menu	45
Table 7: Valid Input for Vote Menu	45
Table 8: Invalid Input for Vote Menu	46
Table 7: Valid Input for View Voting Results and Summary Menu.....	46
Table 8: Invalid Input for View Voting Results and Summary Menu	46
Table 7: Valid Input for View Voting Results and Summary in Specific Division Menu	46
Table 8: Invalid Input for View Voting Results and Summary in Specific Division Menu	46

1 Project Description

This project is meant to create a program which handles the support (or votes) by the voters to candidates in an election. In today's world, more activities are being executed digitally. An election can be done with the help of a simple program.

2 Project Goals and Objectives

2.1 Project Goals

The project goal is to enable election voting to be done quickly and provide good and reliable voting experiences for candidates and voters.

2.2 Project Objectives

The objective of the project is to let voters or candidates to receive vote results without any delays, to let both candidate and voter to easily view candidates' information, to ensure that the voter can only vote once, to properly save and store information of both voter and candidate and to reduce the workload for election workers.

3 Problem Statement

Due to the COVID-19 pandemic, physical voting had become very dangerous and thus unreliable. Apart from that, the operating hours for the voting service department is limited as well as the workers need to have their rest. Voters might not have time to vote due to a variety of reasons which will provide them with more even more reasons not to vote, leading to ineffective voting results. Some voters might try to vote two times for a candidate, which is problematic, if the security services are not good.

4 Design Concept

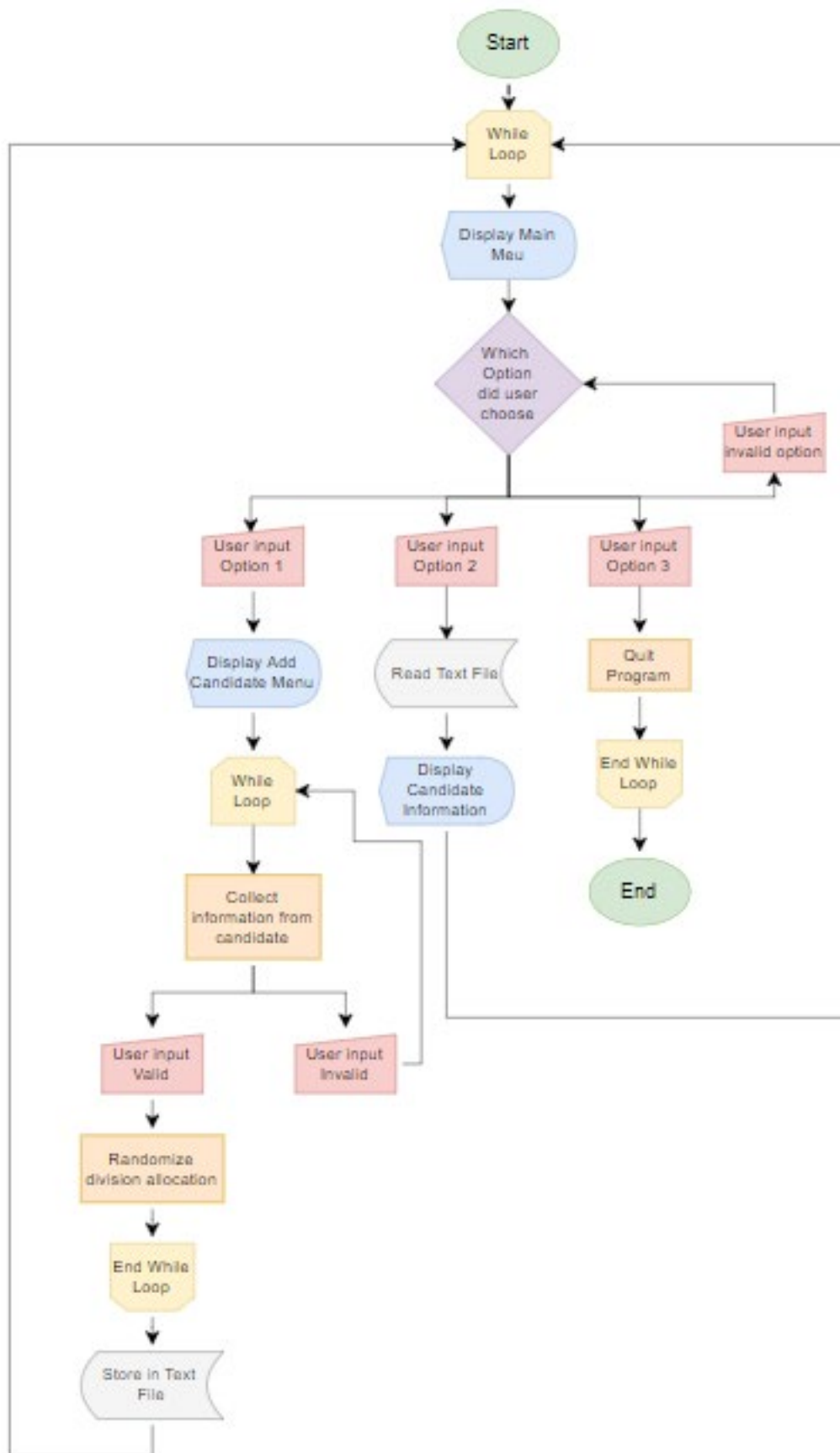


Figure 1: Flowchart for Candidate Program

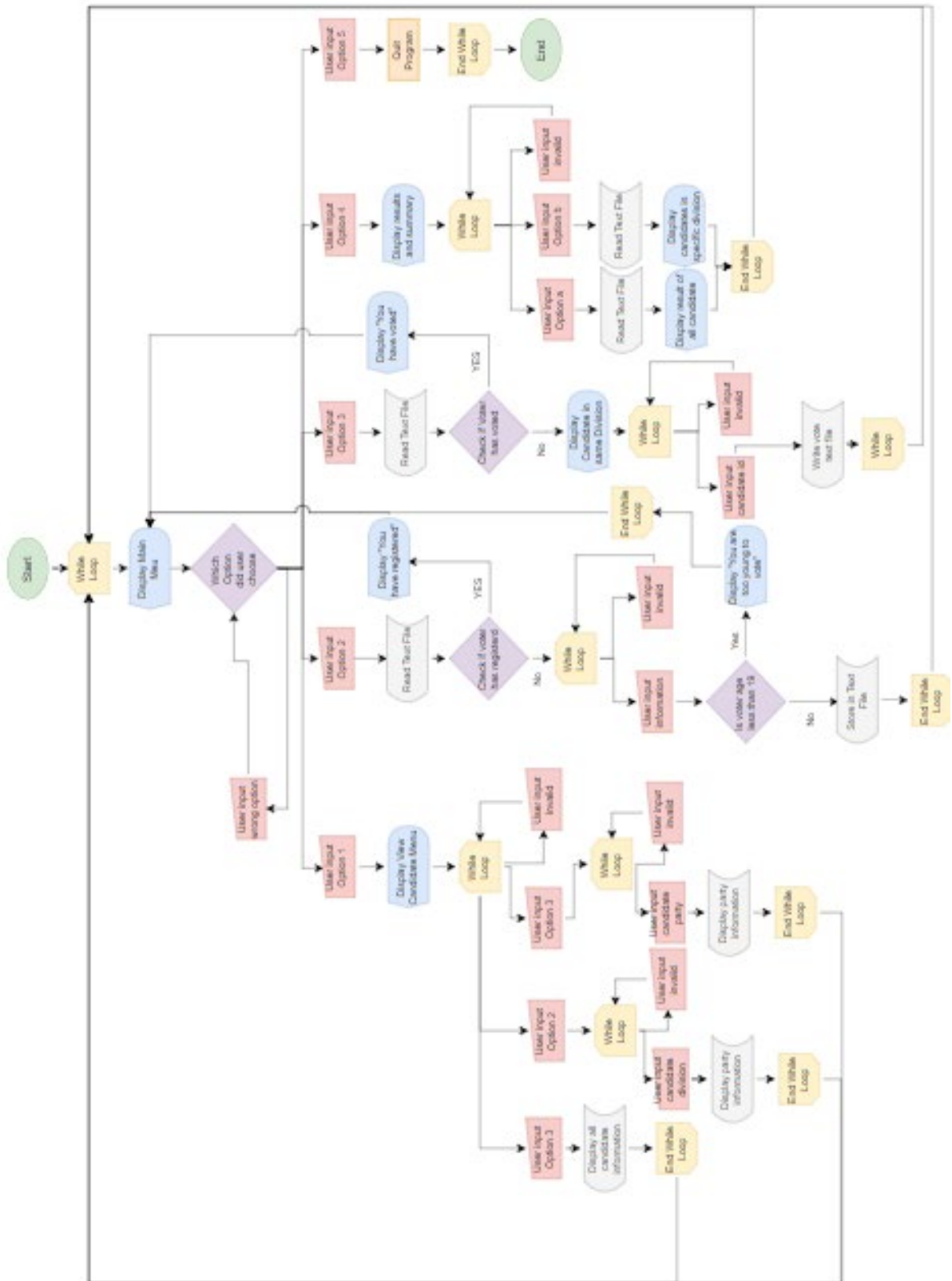


Figure 2: Flowchart for Voter Program

5 Compatibility of Design

5.1 How data is Stored

For both Candidate and Voters data, it will both be stored in a database using SQLite 3. The program will then read the files to write or retrieve data from both candidate and voter side. Candidate and voters' data are both stored in separate database file, with the naming of candidate.txt and voter.txt.

For Candidates, their name, candidate ID, Party, Division and Vote Count is saved in the database. For Voters, their name, voter ID, Division and Status is saved in the database.

5.2 Code Explanation

5.2.1 Candidate Program

```
1  /*
2   Name: Sze Xijie (101222928), Chin Kai Lun (101221815), Chen Jun Yao (101222889), Ooi Yik Quan (101230222)
3   Project Team Name: Amadeus
4   Lab Group: Lab 2 Group 1
5   Project Title: Election-Voter
6   Project Description: Enables voters and candidates to join the election and vote with ease
7  */
8
9  #include <iostream>
10 #include <string>
11 #include <vector>
12 #include <fstream>
13 #include <iomanip>
14 #include <sqlite3.h>
15 #include <algorithm>
16 #include <cctype>
17
18 using namespace std;
19
20 // Calling all functions
21 void mainmenu(); void mainselectionmenu(bool* quit, string selection, string casearray[]);
22 static int createcandidatedb(const char* s); static int createcandidatetable(const char* s);
23 static int selectcandidatedata(const char* s); static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName);
24 int divisionsizecheck(const char* s); void viewcandidateinfo(); void divisioncategorize();
25 static int insertdivisiondata(const char* s, int divisionnumber, int partynumber, int tablesizes, string partyname, string candidateidname);
26 static int partycheck(const char* s, int partysize); vector<string> usedcandidateid(const char* s);
27 static int usedpartyname(const char* s, int division);
```

Figure 3: Program Description for Candidate Program

```
29 // Divide the candidates into random division
30 void divisioncategorize()
31 {
32     bool categorize = false;
33
34     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
35
36     createcandidatedb(division); // Creates The Database If It Is Not Created
37     createcandidatetable(division); // Creates The Candidate Table
38
39     srand(time(NULL)); // Set Random Number To Random So That It Will Always Generate Random Numbers
40
41     while (!categorize)
42     {
43         int randomdivision = ((rand() % 4) + 1); // Generate Random Number From 1 To 4
44
45         if (divisionsizecheck(division) < 12) // Check If The Database Has Reached 12 Candidates
46         {
47             int partylimit = partycheck(division, randomdivision); // Get The Return Value From partycheck function
48
49             string party;
50             string upperletterparty;
51             string candidateid;
52
53             string partyname[3] = { "Einstein", "Tesla", "Mozart" }; // Party Name
```



```

54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98

switch (randomdivision)
{
case 1:
{
    if (partylimit <= 3) // If The Party In Division 1 Is Less Than 3
    {
        party = partyname[usedpartyname(division, randomdivision)]; // Assign Party Name
        upperletterparty = party;
        transform(upperletterparty.begin(), upperletterparty.end(), upperletterparty.begin(), ::toupper); // Changes The User Input To All UpperCase

        candidateid = upperletterparty.substr(0, 3) + "0" + to_string(randomdivision); // Assign Candidate ID

        insertdivisiondata(division, randomdivision, partylimit, divisionsizecheck(division), party, candidateid); // Call insertdivisiondata Function To Insert Candidate

        categorize = true;

        break;
    }
    else
    {
        categorize = false;
    }
}
case 2:
{
    if (partylimit <= 3) // If The Party In Division 2 Is Less Than 3
    {
        party = partyname[usedpartyname(division, randomdivision)]; // Assign Party Name
        upperletterparty = party;
        transform(upperletterparty.begin(), upperletterparty.end(), upperletterparty.begin(), ::toupper); // Changes The User Input To All UpperCase

        candidateid = upperletterparty.substr(0, 3) + "0" + to_string(randomdivision); // Assign Candidate ID

        insertdivisiondata(division, randomdivision, partylimit, divisionsizecheck(division), party, candidateid); // Call insertdivisiondata Function To Insert Candidate

        categorize = true;

        break;
    }
    else
    {
        categorize = false;
    }
}
}

```

```

99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143

case 3:
{
    if (partylimit <= 3) // If The Party In Division 3 Is Less Than 3
    {
        party = partyname[usedpartyname(division, randomdivision)]; // Assign Party Name
        upperletterparty = party;
        transform(upperletterparty.begin(), upperletterparty.end(), upperletterparty.begin(), ::toupper); // Changes The User Input To All UpperCase

        candidateid = upperletterparty.substr(0, 3) + "0" + to_string(randomdivision); // Assign Candidate ID

        insertdivisiondata(division, randomdivision, partylimit, divisionsizecheck(division), party, candidateid); // Call insertdivisiondata Function To Insert Candidate

        categorize = true;

        break;
    }
    else
    {
        categorize = false;
    }
}
case 4:
{
    if (partylimit <= 3) // If The Party In Division 4 Is Less Than 3
    {
        party = partyname[usedpartyname(division, randomdivision)]; // Assign Party Name
        upperletterparty = party;
        transform(upperletterparty.begin(), upperletterparty.end(), upperletterparty.begin(), ::toupper); // Changes The User Input To All UpperCase

        candidateid = upperletterparty.substr(0, 3) + "0" + to_string(randomdivision); // Assign Candidate ID

        insertdivisiondata(division, randomdivision, partylimit, divisionsizecheck(division), party, candidateid); // Call insertdivisiondata Function To Insert Candidate

        categorize = true;

        break;
    }
    else
    {
        categorize = false;
    }
}
default:
    break;
}

```

```

145
146
147
148
149
150
151

else
{
    cout << "I Am Sorry, Candidate Registration Is Over.\n" << endl; // Display A Tex Message If The Candidate Has Reached 12
    categorize = true;
}
}
}

```

Figure 4: divisioncategorize function

```

153 // View Candidate Information
154 void viewcandidateinfo()
155 {
156     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
157
158     if (divisionsizecheck(division) == 0)
159     {
160         cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
161     }
162     else
163     {
164         createcandidatedb(division); // Creates The Database If It Is Not Created
165         createcandidatetable(division); // Creates The Candidate Table
166         selectcandidatedata(division); // Calls selectcandidatedata Function To Select The Candidate Information From Database And Display It
167     }
168 }

```

Figure 5: viewcandidateinfo function

```

170 // Creates The Database
171 static int createcandidatedb(const char* s)
172 {
173     sqlite3* DB;
174     int exit = 0;
175
176     exit = sqlite3_open(s, &DB); // Opens The Database
177
178     sqlite3_close(DB); // Close The Database
179
180     return 0;
181 }

```

Figure 6: createcandidatedb function

```

183 // Creates The Candidate Table In The Database
184 static int createcandidatetable(const char* s)
185 {
186     sqlite3* DB;
187     int exit = 0;
188     char* messageerror;
189
190     string sql = "CREATE TABLE IF NOT EXISTS CandidateTable("
191         "Name      TEXT NOT NULL, "
192         "CandidateID TEXT NOT NULL, "
193         "Party TEXT NOT NULL, "
194         "Division TEXT NOT NULL, "
195         "Vote      TEXT NOT NULL );"; // A Database Format From sqlite3
196
197     exit = sqlite3_open(s, &DB); // Opens The Database
198
199     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
200     exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageerror);
201
202     return 0;
203 }

```

Figure 7: createcandidatetable function

```

205 // Inserts Candidate Information Into The Database
206 static int insertdivisiondata(const char* s, int divisionnumber, int partynumber, int tablesizes, string partyname, string candidateidname)
207 {
208     sqlite3* DB;
209     char* messageError;
210
211     bool insertname = false;
212
213     string candidatename;
214     string candidateid = candidateidname;
215     string party = partyname;
216     string division = to_string(divisionnumber);
217     string voteCount = "0";
218
219     while (!insertname)
220     {
221         cout << "Enter Your Name (With No Space): " << endl;
222         getline(cin, candidatename); // Receives User Input For Their Name
223
224         srand(time(NULL)); // Set Random Number To Random So That It Will Always Generate Random Numbers
225
226         system("cls"); // Clears the previous displayed text on the console
227
228         if (all_of(begin(candidatename), end(candidatename), isalpha)) // Check If The User Input Is All Letters
229         {
230             insertname = true;
231         }
232         else // User Input Is Wrong
233         {
234             cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
235             insertname = false;
236         }
237     }
238
239     string sql = ("INSERT INTO CandidateTable (Name, CandidateID, Party, Division, Vote) VALUES('" + candidatename + "','" + candidateid + "','" + party + "','" + division + "','" + voteCount + "')"); // Inserts The User Input Into The Database
240     int exit = sqlite3_open(s, &DB); // Opens The Database
241
242     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
243     exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);
244
245     if (exit != SQLITE_OK) // If There Is An Error When Inserting Data
246     {
247         cerr << "Error In Inserting Data!\n" << endl; // Display Error Message
248         sqlite3_free(&messageError); // Remove The Error Message
249     }
250     else // If There Is No Error When Inserting Data
251     {
252         cout << "Data Inserted Successfully!\n" << endl; // Display A Text Message
253     }
254     return 0;
255 }

```

Figure 8: insertdivisiondata function

```

257 // Select Candidate Data
258 static int selectcandidatedata(const char* s)
259 {
260     sqlite3* DB;
261     char* messageError;
262
263     string sql = "SELECT * FROM CandidateTable ORDER BY Division ASC;"; // Specify Which Data To Select
264
265     int exit = sqlite3_open(s, &DB); // Opens The Database
266
267     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
268     exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
269
270     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
271     {
272         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
273         sqlite3_free(&messageError); // Remove The Error Message
274     }
275     else // If There Is No Error When Selecting Data
276     {
277         cout << "Data Selected Successfully!\n" << endl; // Display A Text Message
278     }
279     return 0;
280 }

```

Figure 9: selectcandidatedata function

```

281 // Calls The Candidate Data To Be Displayed
282 static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName)
283 {
284     for (int i = 0; i < argc; i++)
285     {
286         cout << azColName[i] << ": " << argv[i] << endl; // Display The Column Name Followed By The Data
287     }
288
289     cout << endl; // Create New Line
290
291     return 0;
292 }

```

Figure 10: callcandidatedata function

```

294 // Store Used Party Name
295 static int usedpartyname(const char* s, int division)
296 {
297     sqlite3* DB;
298     sqlite3_stmt* stmt;
299
300     int partynumber;
301
302     int rc = sqlite3_open(s, &DB); // Opens The Database
303
304     string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'; "; // Specify The Data To Select
305
306     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
307
308     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
309
310     for (;;)
311     {
312         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
313
314         if (rc == SQLITE_DONE) // If Reached The End Of The Data
315         {
316             break;
317         }
318
319         if (rc != SQLITE_ROW) // If There Is No Data In Database
320         {
321             partynumber = 0; // Set maxcandidate To 0
322             break;
323         }
324
325         partynumber = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
326     }
327     sqlite3_finalize(stmt); // Finalize the Data Reading
328
329     return partynumber;
330 }
331

```

Figure 11: usedpartyname function

```

333 // Store Used Candidate ID
334 vector <string> usedcandidateid(const char* s)
335 {
336     sqlite3* DB;
337     sqlite3_stmt* stmt;
338
339     string id;
340
341     vector <string> candidateid;
342     int i = 0;
343
344     int rc = sqlite3_open(s, &DB); // Opens The Database
345
346     rc = sqlite3_prepare_v2(DB, "SELECT CandidateID FROM CandidateTable;", -1, &stmt, NULL); // Specify Which Data To Select
347
348     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
349
350     for (;;)
351     {
352         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
353
354         if (rc == SQLITE_DONE) // If Reached The End Of The Data
355         {
356             break;
357         }
358
359         string name = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable
360
361         candidateid.push_back(name); // Insert The Data Into Vector
362     }
363     sqlite3_finalize(stmt); // Finalize the Data Reading
364
365     return candidateid;
366 }

```

Figure 12: usedcandidateid function

```

368 // Check Size of Candidate Table
369 static int divisionsizecheck(const char* s)
370 {
371     sqlite3* DB;
372     sqlite3_stmt* stmt;
373
374     int maxcandidate = 0;
375
376     int rc = sqlite3_open(s, &DB); // Opens The Database
377
378     rc = sqlite3_prepare_v2(DB, "SELECT Count(Party) FROM CandidateTable;", -1, &stmt, NULL); // Specify The Data To Select
379
380     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
381
382     for (;;)
383     {
384         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
385
386         if (rc == SQLITE_DONE) // If Reached The End Of The Data
387         {
388             break;
389         }
390
391         if (rc != SQLITE_ROW) // If There Is No Data In Database
392         {
393             maxcandidate = 0; // Set maxcandidate To 0
394             break;
395         }
396
397         maxcandidate = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
398     }
399     sqlite3_finalize(stmt); // Finalize the Data Reading
400
401     return maxcandidate;
402 }

```

Figure 13: divisionsizecheck function

```

405 // Check Party Size
406 static int partycheck(const char* s, int division)
407 {
408     sqlite3* DB;
409     sqlite3_stmt* stmt;
410
411     int maxparty = 0;
412
413     string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'"; // Specify The Data To Select
414
415     /* An open database, SQL to be evaluated, Callback Function, 1st argument to callback, Error msg written here*/
416
417     int rc = sqlite3_open(s, &DB); // Opens The Database
418
419     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Gets The Data Following The Specified Data To Select
420
421     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
422
423     for (;;)
424     {
425         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
426
427         if (rc == SQLITE_DONE) // If Reached The End Of The Data
428         {
429             break;
430         }
431
432         maxparty = sqlite3_column_int(stmt, 0) + 1; // Assign Data To maxparty
433     }
434
435     sqlite3_finalize(stmt); // Finalize the Data Reading
436
437     return maxparty;
438 }

```

Figure 14: partycheck function

```

440 // mainmenu() is the function that stores the codes for the main menu screen
441 void mainmenu()
442 {
443     bool mnquit = false;
444     bool* quit = &mnquit;
445
446     // While the user did not quit the program
447     while (!*quit)
448     {
449         // Prompt the user to select an option
450         string selection;
451         string mainmenuarray[] = { "Adding Candidate\n", "Viewing Candidates\n", "Exiting Program", "Your Selected Option Is Invalid, Please Try Again\n" }; // Array For Storing Text
452
453         cout << "Please enter the respective numbers to select the option!\n[1] Add Candidate\n[2] View Candidates\n[3] Exit\n" << endl;
454         getline(cin, selection); // Gets User Input
455
456         system("cls"); // Clears the previous displayed text on the console
457
458         mainselectionmenu(quit, selection, mainmenuarray); // Calls mainselectionmenu function
459     }
460 }

```

Figure 15: mainmenu function

```

462 // Displays the option for the user
463 void mainselectionmenu(bool* quit, string selection, string casearray[])
464 {
465     if (selection == "1") // If the user selected option 1
466     {
467         cout << casearray[0] << endl; // Display the text for option 1
468         divisioncategorize(); // Calls registrationmenu function
469     }
470     else if (selection == "2") // If the user selected option 2
471     {
472         cout << casearray[1] << endl; // Display the text for option 2
473         viewcandidateinfo(); // Calls candidatedisplay function
474     }
475     else if (selection == "3") // If the user selected option 3
476     {
477         *quit = true; // Quit the program
478         cout << casearray[2] << endl; // Display text for option 3
479     }
480     else // If user input anything else
481     {
482         cout << casearray[3] << endl; // Display text for wrong input
483     }
484 }

```

Figure 16: mainselectionmenu function

```

486 int main()
487 {
488     mainmenu();
489 }

```

Figure 17: main function

5.2.2 Voter Program

```
1  /*
2   Name: Sze XiJie (101222928), Chin Kai Lun (101221815), Chen Jun Yao (101222889), Ooi Yik Quan (101230222)
3   Project Team Name: Amadeus
4   Lab Group: Lab 2 Group 1
5   Project Title: Election-Voter
6   Project Description: Enables voters and candidates to join the election and vote with ease
7   */
8
9  #include <iostream>
10 #include <string>
11 #include <vector>
12 #include <fstream>
13 #include <iomanip>
14 #include <sqlite3.h>
15 #include <algorithm>
16 #include <cctype>
17
18 using namespace std;
19
20 // Calling The Functions
21 void VoterMainMenu(); void MainSelectionMenu(bool* Quit, string Selection, string CaseArray[]);
22 void ViewingCandidateMenu(); void ViewingSelectionMenu(bool* Quit, string Selection, string CaseArray[]);
23 void ViewResultMenu(); void ResultSelectionMenu(bool* Quit, string Selection, string CaseArray[]);
24 static int createcandidatedb(const char* s);
25 static int createcandidatetable(const char* s);
26 static int createvoterdb(const char* s);
27 static int createvotertable(const char* s);
28 static int selectcandidatedata(const char* s);
29 static int selectbasedonparty(const char* s, string index);
30 static int selectbasedondivision(const char* s, string division);
31 static int selectalldivision(const char* s);
32 static int minmaxvotealldivision(const char* s, int divisionchoosen, int maximumvote, int minimumvote);
33 static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName);
34 int divisionsizecheck(const char* s);
35 void viewcandidateinfo(); void VoterDatabase();
36 static int insertvoterdata(const char* s);
37 static int partycheck(const char* s, int partysize); vector<string> usedcandidateid(const char* s);
38 static int usedpartyname(const char* s, int division);
39 void viewallcandidateinfo();
40 void viewdivision(string divisionnumber);
41 void viewparty(string partyindex); void VoteCandidate();
42 static int counttotalvote(const char* s, int divisionchoosen);
43 static int callvotedata(void* NotUsed, int argc, char** argv, char** azColName);
44 static int maxvotecount(const char* s, int divisionchoosen);
45 static int minvotecount(const char* s, int divisionchoosen);
46 static int callvoterdata(void* NotUsed, int argc, char** argv, char** azColName);
47 static int selectvoterdata(const char* s);
48 static string checkvoterstatus(const char* s, string voterid);
49 static string checkvoterdivision(const char* s, string voterid);
50 static int candidateinfovoteselection(const char* s, string division);
51 vector<string> storevotereid(const char* s);
52 vector<string> usedcandidateidbasedondivision(const char* s, string division);
53 static int candidatevotebasedonid(const char* s, string candidateid);
54 static int votecountupdate(const char* s, string votecount, string candidateid);
```

Figure 18: Program Description for Voter Program

```
56 // View Candidate Information
57 void viewcandidateinfo()
58 {
59     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
60
61     if (divisionsizecheck(division) == 0)
62     {
63         cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
64     }
65     else
66     {
67         createcandidatedb(division); // Creates The Database If It Is Not Created
68         createcandidatetable(division); // Creates The Candidate Table
69         selectcandidatedata(division); // Calls selectcandidatedata Function To Select The Candidate Information From Database And Display It
70     }
71 }
```

Figure 19: viewcandidaeinfo function


```

73 // Voter's Database
74 void VoterDatabase()
75 {
76     const char* voter = R"(C:\\StoreData\\Voter.db)"; // Calls The Database
77
78     createvoterdb(voter);
79     createvotertable(voter);
80     insertvoterdata(voter);
81 }

```

Figure 20: VoterDatabase function

```

83 // Creates The Database
84 static int createcandidatedb(const char* s)
85 {
86     sqlite3* DB;
87     int exit = 0;
88
89     exit = sqlite3_open(s, &DB); // Opens The Database
90
91     sqlite3_close(DB); // Close The Database
92
93     return 0;
94 }

```

Figure 21: createcandidatedb function

```

96 // Creates The Candidate Table In The Database
97 static int createcandidatetable(const char* s)
98 {
99     sqlite3* DB;
100     int exit = 0;
101     char* messageerror;
102
103     string sql = "CREATE TABLE IF NOT EXISTS VoterTable("
104         "Name      TEXT NOT NULL, "
105         "CandidateID TEXT NOT NULL, "
106         "Party      TEXT NOT NULL, "
107         "Division   TEXT NOT NULL, "
108         "Vote       TEXT NOT NULL );"; // A Database Format From sqlite3
109
110     exit = sqlite3_open(s, &DB); // Opens The Database
111
112     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
113     exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageerror);
114
115     return 0;
116 }

```

Figure 22: createcandidatetable function

```

120 // Creates The Voter Database
121 static int createvoterdb(const char* s)
122 {
123     sqlite3* DB;
124     int exit = 0;
125
126     exit = sqlite3_open(s, &DB); // Opens The Database
127
128     sqlite3_close(DB); // Close The Database
129
130     return 0;
131 }

```

Figure 23: createvoterdb function

```

133 // Creates The Voter Table In The Database
134 static int createvotertable(const char* s)
135 {
136     sqlite3* DB;
137     int exit = 0;
138     char* messageerror;
139
140     string sql = "CREATE TABLE IF NOT EXISTS VoterTable("
141         "Name      TEXT NOT NULL, "
142         "VoterID    TEXT NOT NULL, "
143         "Age        TEXT NOT NULL, "
144         "Division   TEXT NOT NULL, "
145         "Status     TEXT NOT NULL );"; // A Database Format From sqlite3
146
147     exit = sqlite3_open(s, &DB); // Opens The Database
148
149     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
150     exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageerror);
151
152     return 0;
153 }

```

Figure 24:createvotertable function

```

155 // Inserts Voter Information Into The Database
156 static int insertvoterdata(const char* s)
157 {
158     sqlite3* DB;
159     char* messageError;
160
161     bool insertname = false;
162     bool insertage = false;
163     bool insertdivision = false;
164
165     string voterfirstname;
166     string voterlastname;
167     string votername;
168     string age;
169     string voterid;
170     string division;
171     string status = "N";
172
173     while (!insertname)
174     {
175         cout << "Enter Your First Name: " << endl;
176         getline(cin, voterfirstname); // Receives User Input For Their Name
177
178         srand(time(NULL)); // Set Random Number To Random So That It Will Always Generate Random Numbers
179
180         system("cls"); // Clears the previous displayed text on the console
181
182         if (all_of(begin(voterfirstname), end(voterfirstname), isalpha)) // Check If The User Input Is All Letters
183         {
184             cout << "Enter Your Last Name: " << endl;
185             getline(cin, voterlastname); // Receives User Input For Their Name
186
187             system("cls"); // Clears the previous displayed text on the console
188
189             if (all_of(begin(voterlastname), end(voterlastname), isalpha)) // Check If The User Input Is All Letters
190             {
191                 while (!insertdivision)
192                 {
193                     cout << "Enter Your Desired Division: " << endl;
194                     getline(cin, division); // Receives User Input For Their Division
195
196                     system("cls"); // Clears the previous displayed text on the console

```

```

196         system("cls"); // Clears the previous displayed text on the console
197
198         if (division.find_first_not_of("1234") == string::npos)
199         {
200             while (!insertage)
201             {
202                 cout << "Enter Your Age: " << endl;
203                 getline(cin, age); // Receives User Input For Their Age
204
205                 system("cls"); // Clears the previous displayed text on the console
206
207                 if (age.find_first_not_of("0123456789") == string::npos)
208                 {
209                     if (stoi(age) < 19)
210                     {
211                         cout << "I Am Sorry, You Are Too Young To Be Registered As A Voter!\n" << endl; // Display A Text Message
212                         insertname = true;
213                         insertage = true;
214                         insertdivision = true;
215                     }
216                     else
217                     {
218                         votername = voterfirstname + " " + voterlastname;
219                         voterid = voterfirstname + voterlastname;
220
221                         string sql = ("INSERT INTO VoterTable (Name, VoterID, Age, Division, Status) VALUES('" + votername + "','" + voterid + "','" + age + "','" + division + "','" + status + "')"); // Inserts The User Input Into The Database
222
223                         int exit = sqlite3_open(":memory:", &DB); // Opens The Database
224
225                         // An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
226                         exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);
227
228                         if (exit != SQLITE_OK) // If There Is An Error When Inserting Data
229                         {
230                             cerr << "Error In Inserting Data!\n" << endl; // Display Error Message
231                             sqlite3_free(messageError); // Remove The Error Message
232                         }
233                         else // If There Is No Error When Inserting Data
234                         {
235                             cout << "Data Inserted Successfully!\n" << endl; // Display A Text Message
236
237                             insertname = true;
238                             insertage = true;
239                             insertdivision = true;
240                         }
241                     }
242                 }
243             }
244             else // User Input Is Wrong
245             {
246                 cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message

```

```

246     }
247     else // User Input Is Wrong
248     {
249         cout << "I Am Sorry, You Entered A Wrong Division!\n" << endl; // Display A Text Message
250     }
251 }
252 }
253 else // User Input Is Wrong
254 {
255     cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
256 }
257 }
258 else // User Input Is Wrong
259 {
260     cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
261 }
262 }
263
264 return 0;
265 }

```

Figure 25: insertvoterdata function

```

267 // View All Candidate Information
268 void viewallcandidateinfo()
269 {
270     const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database
271
272     if (divisionsizecheck(division) == 0)
273     {
274         cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
275     }
276     else
277     {
278         createcandidatedb(division); // Creates The Database If It Is Not Created
279         createcandidatetable(division); // Creates The Candidate Table
280         selectalldivision(division); // Calls selectalldivision Function To Select The Candidate Information From Database And Display It
281     }
282 }

```

Figure 26: viewallcandidateinfo function

```

284 // Select Candidate Data From All Division
285 static int selectalldivision(const char* s)
286 {
287     sqlite3* DB;
288     char* messageError;
289
290     string sql = "SELECT * FROM CandidateTable ORDER BY Division ASC;"; // Specify Which Data To Select
291
292     int exit = sqlite3_open(s, &DB); // Opens The Database
293
294     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
295     exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
296
297     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
298     {
299         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
300         sqlite3_free(&messageError); // Remove The Error Message
301     }
302     else // If There Is No Error When Selecting Data
303     {
304         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
305     }
306
307     return 0;
308 }

```

Figure 27: selectalldivision function

```

308 // View All Candidate Information In Specific Division
309 void viewdivision(string divisionnumber)
310 {
311     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
312
313     if (divisionsizecheck(division) == 0)
314     {
315         cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
316     }
317     else
318     {
319         createcandidatedb(division); // Creates The Database If It Is Not Created
320         createcandidatetable(division); // Creates The Candidate Table
321         selectbasedondivision(division, divisionnumber); // Calls selectbasedondivision Function To Select The Candidate Information From Database And Display It
322     }
323 }

```

Figure 28: viewdivision function

```

325 // Select Candidate Data Based On Division
326 static int selectbasedondivision(const char* s, string division)
327 {
328     sqlite3* DB;
329     char* messageError;
330
331     string sql = "SELECT * FROM CandidateTable WHERE Division = '" + division + "'"; // Specify Which Data To Select
332
333     int exit = sqlite3_open(s, &DB); // Opens The Database
334
335     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
336     exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
337
338     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
339     {
340         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
341         sqlite3_free(&messageError); // Remove The Error Message
342     }
343     else // If There Is No Error When Selecting Data
344         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
345
346     return 0;
347 }

```

Figure 29: selectbasedondivision function

```

349 // View All Candidate Information In Specific Party
350 void viewparty(string partyindex)
351 {
352     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
353
354     if (divisionsizecheck(division) == 0)
355     {
356         cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
357     }
358     else
359     {
360         createcandidatedb(division); // Creates The Database If It Is Not Created
361         createcandidatetable(division); // Creates The Candidate Table
362         selectbasedonparty(division, partyindex); // Calls selectbasedonparty Function To Select The Candidate Information From Database And Display It
363     }
364 }

```

Figure 30: viewparty function

```

366 // Select Candidate Data Based on Party
367 static int selectbasedonparty(const char* s, string index)
368 {
369     sqlite3* DB;
370     char* messageError;
371     string party[] = { "Einstein", "Tesla", "Mozart" };
372     string partyname = party[stoi(index) - 1];
373
374     string sql = "SELECT * FROM CandidateTable WHERE Party = '" + partyname + "'ORDER BY Division ASC; "; // Specify Which Data To Select
375
376     int exit = sqlite3_open(s, &DB); // Opens The Database
377
378     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
379     exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
380
381     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
382     {
383         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
384         sqlite3_free(&messageError); // Remove The Error Message
385     }
386     else // If There Is No Error When Selecting Data
387         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
388
389     return 0;
390 }

```

Figure 31: selectbasedonparty function

```

392 // Select Candidate Data
393 static int selectcandidatedata(const char* s)
394 {
395     sqlite3* DB;
396     char* messageError;
397
398     string sql = "SELECT * FROM CandidateTable ORDER BY Division ASC;"; // Specify Which Data To Select
399
400     int exit = sqlite3_open(s, &DB); // Opens The Database
401
402     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
403     exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
404
405     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
406     {
407         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
408         sqlite3_free(&messageError); // Remove The Error Message
409     }
410     else // If There Is No Error When Selecting Data
411         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
412
413     return 0;
414 }

```

Figure 32: selectcandidatadata function

```

416 // Select Voter Data
417 static int selectvoterdata(const char* s)
418 {
419     sqlite3* DB;
420     char* messageError;
421
422     string sql = "SELECT * FROM VoterTable ORDER BY Division ASC;"; // Specify Which Data To Select
423
424     int exit = sqlite3_open(s, &DB); // Opens The Database
425
426     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
427     exit = sqlite3_exec(DB, sql.c_str(), callvoterdata, NULL, &messageError);
428
429     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
430     {
431         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
432         sqlite3_free(&messageError); // Remove The Error Message
433     }
434     else // If There Is No Error When Selecting Data
435         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
436
437     return 0;
438 }

```

Figure 33: selectvoterdata function

```

440 // Select Maximum and Minimum Votes Candidate Data
441 static int minmaxvotealldivision(const char* s, int divisionchoosen, int maximumvote, int minimumvote)
442 {
443     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
444
445     sqlite3* DB;
446     char* messageError;
447
448     string maxvote = "SELECT Name, CandidateID, Party, Vote FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "' AND Vote = '" + to_string(maximumvote) + "'"; // Select Max vote data
449     string minvote = "SELECT Name, CandidateID, Party, Vote FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "' AND Vote = '" + to_string(minimumvote) + "'"; // Select Min vote data
450
451     int exit = sqlite3_open(s, &DB); // Opens The Database
452
453     if (divisionsizecheck(division) == 0)
454     {
455         cout << "Sorry, There Is No Candidate!\n" << endl;
456     }
457     else
458     {
459         /* An open database, maxvote to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
460         exit = sqlite3_exec(DB, maxvote.c_str(), callcandidatedata, NULL, &messageError);
461
462         if (counttotalvote(division, divisionchoosen) != 0)
463         {
464             float percentage = (float)((maxvotecount(division, divisionchoosen) * 100) / counttotalvote(division, divisionchoosen));
465             cout << "Percentage of Vote: " << percentage << "%\n" << endl;
466         }
467         else
468         {
469             cout << "Percentage of Vote: 0%\n" << endl;
470         }
471
472         /* An open database, minvote to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
473         exit = sqlite3_exec(DB, minvote.c_str(), callcandidatedata, NULL, &messageError);
474
475         if (counttotalvote(division, divisionchoosen) != 0)
476         {
477             float percentage = (float)((minvotecount(division, divisionchoosen) / counttotalvote(division, divisionchoosen)) * 100);
478             cout << "Percentage of Vote: " << percentage << "%\n" << endl;
479         }
480         else
481         {
482             cout << "Percentage of Vote: 0%\n" << endl;
483         }
484     }
485 }
486
487
488     cout << "Total Vote: " << counttotalvote(division, divisionchoosen) << "\n" << endl;
489 }
490
491 return 0;
492 }

```

Figure 34: minmaxvotealldivision function

```

494 // Count the max votes in a division
495 static int maxvoteccount(const char* s, int divisionchoosen)
496 {
497     sqlite3* DB;
498     sqlite3_stmt* stmt{};
499
500     char* messageError;
501
502     int maxvote = 0;
503
504     int rc = sqlite3_open(s, &DB); // Opens The Database
505
506     string sql = "SELECT MAX(Vote) FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "'"; // Specify The Data To Select
507
508     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
509
510     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
511
512     for (;;)
513     {
514         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
515
516         if (rc == SQLITE_DONE) // If Reached The End Of The Data
517         {
518             break;
519         }
520
521         if (rc != SQLITE_ROW) // If There Is No Data In Database
522         {
523             maxvote = 0; // Set maxcandidate To 0
524             break;
525         }
526
527         maxvote = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
528     }
529     sqlite3_finalize(stmt); // Finalize the Data Reading
530
531     return maxvote;
532 }

```

Figure 35: maxvoteccount function

```

534 // Count the min votes in a division
535 static int minvotecount(const char* s, int divisionchoosen)
536 {
537     sqlite3* DB;
538     sqlite3_stmt* stmt{};
539
540     char* messageError;
541
542     int minvote = 0;
543
544     int rc = sqlite3_open(s, &DB); // Opens The Database
545
546     string sql = "SELECT MIN(Vote) FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "'"; // Specify The Data To Select
547
548     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
549
550     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
551
552     for (;;)
553     {
554         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
555
556         if (rc == SQLITE_DONE) // If Reached The End Of The Data
557         {
558             break;
559         }
560
561         if (rc != SQLITE_ROW) // If There Is No Data In Database
562         {
563             minvote = 0; // Set maxcandidate To 0
564             break;
565         }
566
567         minvote = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
568     }
569     sqlite3_finalize(stmt); // Finalize the Data Reading
570
571     return minvote;
572 }

```

Figure 36: minvotecount function

```

574 // Count the total votes in a division
575 static int counttotalvote(const char* s, int divisionchoosen)
576 {
577     sqlite3* DB;
578     sqlite3_stmt* stmt{};
579
580     char* messageError;
581
582     int totalvote = 0;
583
584     int rc = sqlite3_open(s, &DB); // Opens The Database
585
586     string sql = "SELECT Vote FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "'"; // Specify The Data To Select
587
588     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
589
590     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
591
592     for (;;)
593     {
594         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
595
596         if (rc == SQLITE_DONE) // If Reached The End Of The Data
597         {
598             break;
599         }
600
601         if (rc != SQLITE_ROW) // If There Is No Data In Database
602         {
603             break;
604         }
605
606         totalvote = totalvote + sqlite3_column_int(stmt, 0); // Assign Data To totalvote
607     }
608     sqlite3_finalize(stmt); // Finalize the Data Reading
609
610     return totalvote;
611 }

```

Figure 37: counttotalvote function


```

613 // Calls The Candidate Data To Be Displayed
614 static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName)
615 {
616     for (int i = 0; i < argc; i++)
617     {
618         cout << azColName[i] << ": " << argv[i] << endl; // Display The Column Name Followed By The Data
619     }
620
621     cout << endl; // Create New Line
622
623     return 0;
624 }

```

Figure 38: callcandidatedata function

```

626 // Calls The Voter Data
627 static int callvoterdata(void* NotUsed, int argc, char** argv, char** azColName)
628 {
629     for (int i = 0; i < argc; i++)
630     {
631         cout << azColName[i] << ": " << argv[i] << endl; // Display The Column Name Followed By The Data
632     }
633
634     cout << endl; // Create New Line
635
636     return 0;
637 }

```

Figure 39: callvoterdata function

```

639 // Calls The Candidate Vote Data
640 static int callvotedata(void* NotUsed, int argc, char** argv, char** azColName)
641 {
642     for (int i = 0; i < argc; i++)
643     {
644         azColName[i];
645         argv[i];
646     }
647
648     return 0;
649 }

```

Figure 40: callvotedata function

```

651 // Checks for party name used
652 static int usedpartyname(const char* s, int division)
653 {
654     sqlite3* DB;
655     sqlite3_stmt* stmt;
656
657     int partynumber;
658
659     int rc = sqlite3_open(s, &DB); // Opens The Database
660
661     string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'"; // Specify The Data To Select
662
663     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
664
665     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
666
667     for (;;)
668     {
669         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
670
671         if (rc == SQLITE_DONE) // If Reached The End Of The Data
672         {
673             break;
674         }
675
676         if (rc != SQLITE_ROW) // If There Is No Data In Database
677         {
678             partynumber = 0; // Set maxcandidate To 0
679             break;
680         }
681
682         partynumber = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
683     }
684     sqlite3_finalize(stmt); // Finalize the Data Reading
685
686     return partynumber;
687 }
688

```

Figure 41: usedpartyname function

```

690 // Check for voter's status
691 static string checkvoterstatus(const char* s, string voterid)
692 {
693     sqlite3* DB;
694     sqlite3_stmt* stmt;
695
696     string status;
697
698     int rc = sqlite3_open(s, &DB); // Opens The Database
699
700     string sql = "SELECT Status FROM VoterTable WHERE VoterID = '" + voterid + "'"; // Specify The Data To Select
701
702     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
703
704     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
705
706     for (;;)
707     {
708         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
709
710         if (rc == SQLITE_DONE) // If Reached The End Of The Data
711         {
712             break;
713         }
714
715         if (rc != SQLITE_ROW) // If There Is No Data In Database
716         {
717             break;
718         }
719
720         status = (const char*)(sqlite3_column_text(stmt, 0)); // Assign Data To status
721     }
722     sqlite3_finalize(stmt); // Finalize the Data Reading
723
724     return status;
725 }
726

```

Figure 42: checkvoterstatus function

```

727 // Store Used Candidate ID Based On Division
728 vector <string> usedcandidateidbasedondivision(const char* s, string division)
729 {
730     sqlite3* DB;
731     sqlite3_stmt* stmt;
732
733     vector <string> candidateid;
734
735     int rc = sqlite3_open(s, &DB); // Opens The Database
736
737     string sql = "SELECT CandidateID FROM CandidateTable WHERE Division = '" + division + "'";
738
739     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
740
741     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
742
743     for (;;)
744     {
745         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
746
747         if (rc == SQLITE_DONE) // If Reached The End Of The Data
748         {
749             break;
750         }
751
752         string name = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable
753
754         candidateid.push_back(name); // Insert The Data Into Vector
755
756         if (rc != SQLITE_ROW) // If There Is No Data In Database
757         {
758             break;
759         }
760     }
761     sqlite3_finalize(stmt); // Finalize the Data Reading
762
763     return candidateid;
764 }

```

Figure 43: usedcandidateidbasedondivision function

```

766 // Store Used Candidate ID
767 vector <string> usedcandidateid(const char* s)
768 {
769     sqlite3* DB;
770     sqlite3_stmt* stmt;
771
772     string id;
773
774     vector <string> candidateid;
775     int i = 0;
776
777     int rc = sqlite3_open(s, &DB); // Opens The Database
778
779     rc = sqlite3_prepare_v2(DB, "SELECT CandidateID FROM CandidateTable;", -1, &stmt, NULL); // Specify Which Data To Select
780
781     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
782
783     for (;;)
784     {
785         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
786
787         if (rc == SQLITE_DONE) // If Reached The End Of The Data
788         {
789             break;
790         }
791
792         string name = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable
793
794         candidateid.push_back(name); // Insert The Data Into Vector
795     }
796     sqlite3_finalize(stmt); // Finalize the Data Reading
797
798     return candidateid;
799 }

```

Figure 44: usedcandidateid function

```

801 // Check Size of Candidate Table
802 static int divisionsizecheck(const char* s)
803 {
804     sqlite3* DB;
805     sqlite3_stmt* stmt;
806
807     int maxcandidate = 0;
808
809     int rc = sqlite3_open(s, &DB); // Opens The Database
810
811     rc = sqlite3_prepare_v2(DB, "SELECT Count(Party) FROM CandidateTable;", -1, &stmt, NULL); // Specify The Data To Select
812
813     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
814
815     for (;;)
816     {
817         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
818
819         if (rc == SQLITE_DONE) // If Reached The End Of The Data
820         {
821             break;
822         }
823
824         if (rc != SQLITE_ROW) // If There Is No Data In Database
825         {
826             maxcandidate = 0; // Set maxcandidate To 0
827             break;
828         }
829
830         maxcandidate = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
831     }
832     sqlite3_finalize(stmt); // Finalize the Data Reading
833
834     return maxcandidate;
835 }

```

Figure 45: divisionsizecheck function

```

838 // Check Party Size
839 static int partycheck(const char* s, int division)
840 {
841     sqlite3* DB;
842     sqlite3_stmt* stmt;
843
844     int maxparty = 0;
845
846     string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'"; // Specify The Data To Select
847     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
848
849     int rc = sqlite3_open(s, &DB); // Opens The Database
850
851     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Gets The Data Following The Specified Data To Select
852
853     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
854
855     for (;;)
856     {
857         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
858
859         if (rc == SQLITE_DONE) // If Reached The End Of The Data
860         {
861             break;
862         }
863
864         maxparty = sqlite3_column_int(stmt, 0) + 1; // Assign Data To maxparty
865     }
866
867     sqlite3_finalize(stmt); // Finalize the Data Reading
868
869     return maxparty;
870 }

```

Figure 46: partycheck function

```

873 // Check Division Based On Voter ID
874 static string checkvoterdivision(const char* s, string voterid)
875 {
876     sqlite3* DB;
877     sqlite3_stmt* stmt;
878
879     string division;
880
881
882     int rc = sqlite3_open(s, &DB); // Opens The Database
883
884     string sql = "SELECT Division FROM VoterTable WHERE VoterID = '" + voterid + "'"; // Specify The Data To Select
885
886     rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select
887
888     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
889
890     for (;;)
891     {
892         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
893
894         if (rc == SQLITE_DONE) // If Reached The End Of The Data
895         {
896             break;
897         }
898
899         if (rc != SQLITE_ROW) // If There Is No Data In Database
900         {
901             break;
902         }
903
904         division = (const char*)(sqlite3_column_text(stmt, 0)); // Assign Data To status
905     }
906     sqlite3_finalize(stmt); // Finalize the Data Reading
907
908     return division;
909 }

```

Figure 47: checkvoterdivision function

```

911 // Select Candidate Information When Voter Is Voter
912 static int candidateinfovoteselection(const char* s, string division)
913 {
914     sqlite3* DB;
915     sqlite3_stmt* stmt;
916
917     char* messageError;
918
919     int exit = sqlite3_open(s, &DB); // Opens The Database
920
921     string sql = "SELECT Name, CandidateID, Party FROM CandidateTable WHERE Division = '" + division + "'"; // Specify The Data To Select
922
923     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
924     exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
925
926     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
927     {
928         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
929         sqlite3_free(&messageError); // Remove The Error Message
930     }
931     else // If There Is No Error When Selecting Data
932     {
933         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
934     }
935     return 0;
936 }

```

Figure 48: candidaeinfovoteselection function

```

937 // Store Voter ID
938 vector <string> storevotereid(const char* s)
939 {
940     sqlite3* DB;
941     sqlite3_stmt* stmt;
942
943     vector <string> voterid;
944
945     int rc = sqlite3_open(s, &DB); // Opens The Database
946
947     rc = sqlite3_prepare_v2(DB, "SELECT VoterID FROM VoterTable;", -1, &stmt, NULL); // Specify Which Data To Select
948
949     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
950
951     for (;;)
952     {
953         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
954
955         if (rc == SQLITE_DONE) // If Reached The End Of The Data
956         {
957             break;
958         }
959
960         string id = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable
961
962         voterid.push_back(id); // Insert The Data Into Vector
963     }
964     sqlite3_finalize(stmt); // Finalize the Data Reading
965
966     return voterid;
967 }

```

Figure 49: storevotereid function

```

970 static int voterstatusupdate(const char* s, string voterid)
971 {
972     sqlite3* DB;
973     char* messageError;
974
975     string sql("UPDATE VoterTable SET Status = 'Y' WHERE VoterId = '" + voterid + "'");
976
977     int exit = sqlite3_open(s, &DB);
978
979     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
980     exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);
981
982     return 0;
983 }

```

Figure 50: voterstatusupdate function

```

985 // Update Candidates Vote
986 static int votecountupdate(const char* s, string votecount, string candidateid)
987 {
988     sqlite3* DB;
989     char* messageError;
990
991     string sql("UPDATE CandidateTable SET Vote = '" + votecount + "' WHERE CandidateId = '" + candidateid + "'");
992
993     int exit = sqlite3_open(s, &DB);
994
995     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
996     exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);
997
998     return 0;
999 }

```

Figure 51: votecountupdate function


```

1089     vector<string> existingcandidateid = usedcandidateidbasedondivision(candidate, checkvoterdivision(voter, voterid)); // Set usedcandidatebasedondivision function as vector
1090
1091     if (find(existingcandidateid.begin(), existingcandidateid.end(), candidateid) != existingcandidateid.end()) // If Candidate ID Matches With Existing ID
1092     {
1093         for (int j = 0; j < size(usedcandidateidbasedondivision(candidate, checkvoterdivision(voter, voterid))); j++) // Loop To Get Candidate ID Index
1094         {
1095             if (usedcandidateidbasedondivision(candidate, checkvoterdivision(voter, voterid))[j] == candidateid) // If candidateid == usedcandidateidbasedondivision function
1096             {
1097                 voteupdate(candidate, to_string(candidatevotebasedonid(candidate, candidateid) + 1), candidateid); // call voteupdate Function To Increment Vote By 1
1098                 voterstatusupdate(voter, voterid); // Update Voter Status To Y
1099                 cout << "You Have Voted For " << candidateid << "\n" << endl; // Display Text
1100
1101                 candidateinput = true;
1102                 voteridinput = true;
1103                 break;
1104             }
1105         }
1106     }
1107
1108     else // If Candidate ID Does Not Match
1109     {
1110         cout << "This Candidate Does Not Exist!!\n" << endl; // Display Text
1111         candidateinput = true;
1112         voteridinput = true;
1113         break;
1114     }
1115 }
1116
1117 else // If Voter Status is "Y"
1118 {
1119     cout << "You Have Have Already Voted\n" << endl; // Display Text
1120     voteridinput = true;
1121 }
1122
1123 }
1124
1125 else // If Wrong Input
1126 {
1127     cout << "You Have Entered A Wrong Input, Please Try Again!!\n" << endl; // Display Text
1128     voteridinput = false;
1129 }
1130
1131 }

```

Figure 53: VoteCandidate function

```

1133 // Select Candidate Vote Count
1134 static int selectcandidatedata(const char* s, string candidateid)
1135 {
1136     sqlite3* DB;
1137     sqlite3_stmt* stmt;
1138
1139     char* messageError;
1140
1141     int voteCount;
1142
1143     string sql = "SELECT Vote FROM CandidateTable WHERE CandidateID = '" + candidateid + "'"; // Specify Which Data To Select
1144
1145     int rc = sqlite3_open(s, &DB); // Opens The Database
1146
1147     /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
1148     rc = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);
1149
1150     sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value
1151
1152     for (;;)
1153     {
1154         int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)
1155
1156         if (rc == SQLITE_DONE) // If Reached The End Of The Data
1157         {
1158             break;
1159         }
1160
1161         voteCount = sqlite3_column_int(stmt, 0) + 1; // Assign Data To maxparty
1162     }
1163
1164     sqlite3_finalize(stmt); // Finalize the Data Reading
1165
1166     if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
1167     {
1168         cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
1169         sqlite3_free(&messageError); // Remove The Error Message
1170     }
1171     else // If There Is No Error When Selecting Data
1172     {
1173         cout << "Data Selected Successfully\n" << endl; // Display A Text Message
1174     }
1175
1176     return voteCount;
1177 }

```

Figure 54: selectcandidatedata function


```

1177 // Menu For Option 4
1178 void ViewResultMenu()
1179 {
1180     bool RQuit = false;
1181     bool* QuitResult = &RQuit;
1182
1183     while (!*QuitResult) // While Still Viewing The Menu
1184     {
1185         string ResultSelection;
1186         string ResultArray[] = { "Viewing All Results From All Divisions\n", "Which Division's Information Would You Like To View?\n[1] Division 1\n[2] Division 2\n[3] Division 3\n[4] Division 4",
1187             "Going Back To Main Menu\n", "You Have Entered An Invalid Option, Please Try Again!\n" };
1188
1189         cout << "Please Select Your Result Viewing Options.\n[1] View All Results From All Divisions\n[2] View Results In A Specific Division\n[3] Back To Main Menu\n" << endl;
1190         getline(cin, ResultSelection);
1191
1192         system("cls");
1193
1194         ResultSelectionMenu(QuitResult, ResultSelection, ResultArray);
1195     }
1196 }

```

Figure 55: ViewResultMenu function

```

1198 // Menu For 2 Different Result Viewing Options In Option 4
1199 void ResultSelectionMenu(bool* Quit, string Selection, string CaseArray[])
1200 {
1201     const char* division = R"(C:\StoreData\Division.db)"; // Calls The Database
1202
1203     bool divisionselection = false;
1204     bool partyselection = false;
1205     string choosendivision;
1206     string choosenparty;
1207
1208     if (Selection == "1") // If User Choose Option 1
1209     {
1210         cout << CaseArray[0] << endl; // Display First Index From CaseArray
1211
1212         for (int i = 1; i < 5; i++) // For Every Index
1213         {
1214             minmaxvotealldivision(division, i, maxvotecount(division, i), minvotecount(division, i)); // Call minmaxvotealldivision Function
1215         }
1216         *Quit = true;
1217     }
1218     else if (Selection == "2") // If User Choose Option 1
1219     {
1220         while (divisionselection)
1221         {
1222             cout << CaseArray[1] << endl; // Display Second Index From CaseArray
1223             cin >> choosendivision;
1224
1225             system("cls"); // Clear Console
1226
1227             if (choosendivision == "1") // If User Choose Division 1
1228             {
1229                 cout << "You Have Chosen Division 1\n" << endl; // Display Text
1230                 minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function
1231
1232                 cin.clear(); // Clears Input Buffer
1233                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1234                 divisionselection = true;
1235             }
1236             else if (choosendivision == "2") // If User Choose Division 2
1237             {
1238                 cout << "You Have Chosen Division 2\n" << endl; // Display Text
1239                 minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function
1240
1241                 cin.clear(); // Clears Input Buffer
1242                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1243                 divisionselection = true;
1244             }
1245         }
1246     }
1247     else if (choosendivision == "3") // If User Choose Division 3
1248     {
1249         cout << "You Have Chosen Division 3\n" << endl; // Display Text
1250         minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function
1251
1252         cin.clear(); // Clears Input Buffer
1253         cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1254         divisionselection = true;
1255     }
1256     else if (choosendivision == "4") // If User Choose Division 4
1257     {
1258         cout << "You Have Chosen Division 4\n" << endl;
1259         minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function
1260
1261         cin.clear(); // Clears Input Buffer
1262         cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1263         divisionselection = true;
1264     }
1265     else // If User Choose Wrong Option
1266     {
1267         cout << "Invalid Option!\n" << endl;
1268         divisionselection = false;
1269     }
1270 }
1271
1272 *Quit = true;
1273
1274 else if (Selection == "3") // If User Choose Option 3
1275 {
1276     cout << CaseArray[2] << endl; // Display Third Index From CaseArray
1277     *Quit = true;
1278 }
1279 else // If User Choose Wrong Option
1280 {
1281     cout << CaseArray[3] << endl; // Display Fourth Index From CaseArray
1282     *Quit = false;
1283 }
1284 }
1285 }

```

Figure 56: ResultSelectionMenu function

```

1287 // Menu For Option 1
1288 void ViewingCandidateMenu()
1289 {
1290     bool VQuit = false;
1291     bool* QuitViewing = &VQuit;
1292
1293     while (!*QuitViewing)
1294     {
1295         string ViewingSelection;
1296         string ViewingArray[] = { "Viewing All Candidates In All Divisions\n", "Which Division Would You Like To View ?\n[1] Division 1\n[2] Division 2\n[3] Division 3\n[4] Division 4\n",
1297             "Which Party Would You Like To View?\n[1] Einstein\n[2] Tesla\n[3] Mozart\n", "Going Back To Main Menu\n", "You Have Entered An Invalid Option, Please Try Again!\n" };
1298
1299         cout << "Please select the following options by typing in the number.\n[1] View All Candidates In All Divisions\n"
1300             "[2] View The Candidates In Specific Division\n[3] View The Candidates Based On The Party\n[4] Back To Main Menu\n" << endl;
1301         getline(cin, ViewingSelection);
1302
1303         system("cls");
1304
1305         ViewingSelectionMenu(QuitViewing, ViewingSelection, ViewingArray); // Call ViewingSelectionMenu Function
1306     }
1307 }

```

Figure 57: ViewingCandidateMenu function

```

1309 // Main Menu
1310 void VoterMainMenu()
1311 {
1312     bool MMQuit = false;
1313     bool* Quit = &MMQuit;
1314
1315     while (!*Quit)
1316     {
1317         string Selection;
1318         string MainMenuArray[] = { "Viewing Candidates\n", "Registering as Voter\n", "Voting for Candidate\n", "Viewing voting results and summary\n",
1319             "Exiting Program\n", "You Have Entered An Invalid Option, Please Try Again!\n" };
1320
1321         cout << "Please select the following options by typing in the number.\n[1] View Candidates\n[2] Register Voter\n"
1322             "[3] Vote\n[4] View voting results and summary\n[5] Exit\n" << endl;
1323         getline(cin, Selection);
1324
1325         system("cls"); // Clear Console
1326
1327         MainSelectionMenu(Quit, Selection, MainMenuArray); // Call MainSelecionMenu Function
1328     }
1329 }

```

Figure 58: VoterMainMenu function

```

1331 // Menu For 3 Different Viewing Candidate Options In Option 1
1332 void ViewingSelectionMenu(bool* Quit, string Selection, string CaseArray[])
1333 {
1334     const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database
1335
1336     bool divisionselection = false;
1337     bool partyselection = false;
1338     string choosendivision;
1339     string choosenparty;
1340
1341     if (Selection == "1") // If User Choose Option 1
1342     {
1343         cout << CaseArray[0] << endl; // Display First Index Text From CaseArray
1344         viewcandidateinfo(); // Call viewcandidateinfo Function
1345         *Quit = true;
1346     }
1347     else if (Selection == "2") // If User Choose Option 2
1348     {
1349         while (!divisionselection)
1350         {
1351             cout << CaseArray[1] << endl; // Display Second Index Text From CaseArray
1352             cin >> choosendivision;
1353
1354             system("cls"); // Clear Console
1355
1356             if (choosendivision == "1") // If User Choose Division 1
1357             {
1358                 cout << "You Have Chosen Division 1\n" << endl; // Display Text
1359                 viewdivision(choosendivision); // Call viewdivision Function
1360
1361                 cin.clear(); // Clears Input Buffer
1362                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1363
1364                 divisionselection = true;
1365             }
1366             else if (choosendivision == "2") // If User Choose Division 2
1367             {
1368                 cout << "You Have Chosen Division 2\n" << endl; // Display Text
1369                 viewdivision(choosendivision); // Call viewdivision Function
1370
1371                 cin.clear(); // Clears Input Buffer
1372                 cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1373
1374                 divisionselection = true;
1375             }
1376             else if (choosendivision == "3") // If User Choose Division 3
1377             {
1378                 cout << "You Have Chosen Division 3\n" << endl; // Display Text
1379                 viewdivision(choosendivision); // Call viewdivision Function
1380             }
1381         }
1382     }
1383 }

```

```

1380
1381     cin.clear(); // Clears Input Buffer
1382     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1383
1384     divisionselection = true;
1385 }
1386 else if (choosendivision == "4") // If User Choose Division 4
1387 {
1388     cout << "You Have Chosen Division 4\n" << endl; // Display Text
1389     viewdivision(choosendivision); // Call viewdivision Function
1390
1391     cin.clear(); // Clears Input Buffer
1392     cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1393
1394     divisionselection = true;
1395 }
1396 else // If User Choose Wrong Option
1397 {
1398     cout << "Invalid Option! Try Again\n" << endl; // Display Text
1399     divisionselection = false;
1400 }
1401 }
1402 *Quit = true;
1403 }
1404 else if (Selection == "3") // If User Choose Option 3
1405 {
1406     while (!partyselection)
1407     {
1408         cout << CaseArray[2] << endl; // Display Third Index Text From CaseArray
1409         cin >> choosenparty;
1410
1411         system("cls"); // Clear Console
1412
1413         if (choosenparty == "1") // If User Choose Party Einstein
1414         {
1415             cout << "Party Einstein Selected\n" << endl; // Display Text
1416             viewparty(choosenparty); // Call viewparty Function
1417
1418             cin.clear(); // Clears Input Buffer
1419             cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1420
1421             partyselection = true;

```

```

1422     }
1423     else if (choosenparty == "2") // If User Choose Party Tesla
1424     {
1425         cout << "Party Tesla Selected\n" << endl; // Display Text
1426         viewparty(choosenparty); // Call viewparty Function
1427
1428         cin.clear(); // Clears Input Buffer
1429         cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1430
1431         partyselection = true;
1432     }
1433     else if (choosenparty == "3") // If User Choose Party Mozart
1434     {
1435         cout << "Party Mozart Selected\n" << endl; // Display Text
1436         viewparty(choosenparty); // Call viewparty Function
1437
1438         cin.clear(); // Clears Input Buffer
1439         cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line
1440
1441         partyselection = true;
1442     }
1443     else // If User Choose Wrong Option
1444     {
1445         cout << "Invalid Option! Try Again\n" << endl; // Display Text
1446         partyselection = false;
1447     }
1448 }
1449 *Quit = true;
1450 }
1451 else if (Selection == "4") // If User Choose Option 4
1452 {
1453     cout << CaseArray[3] << endl; // Display Fourth Index Text From CaseArray
1454     *Quit = true;
1455 }
1456 else // If User Choose Wrong Option
1457 {
1458     cout << CaseArray[4] << endl; // Display Fifth Index Text From CaseArray
1459     *Quit = false;
1460 }
1461 }

```

Figure 59: ViewingSelectionMenu function

```

1463 // Menu For 5 Different Option
1464 void MainSelectionMenu(bool* Quit, string Selection, string CaseArray[])
1465 {
1466     if (Selection == "1") // If User Choose Option 1
1467     {
1468         cout << CaseArray[0] << endl; // Display First Index Text From CaseArray
1469         ViewingCandidateMenu(); // Call ViewingCandidateMenu Function
1470     }
1471     else if (Selection == "2") // If User Choose Option 2
1472     {
1473         cout << CaseArray[1] << endl; // Display Second Index Text From CaseArray
1474         VoterDatabase(); // Call voterdatabase Function
1475     }
1476     else if (Selection == "3") // If User Choose Option 3
1477     {
1478         cout << CaseArray[2] << endl; // Display Third Index Text From CaseArray
1479         VoteCandidate(); // Callotecandidate Function
1480     }
1481     else if (Selection == "4") // If User Choose Option 4
1482     {
1483         cout << CaseArray[3] << endl; // Display Fourth Index Text From CaseArray
1484         ViewResultMenu(); // Call ViewResultMenu Function
1485     }
1486     else if (Selection == "5") // If User Choose Option 5
1487     {
1488         cout << CaseArray[4] << endl; // Display Fifth Index Text From CaseArray
1489         *Quit = true;
1490     }
1491     else // If User Choose Wrong Option
1492     {
1493         cout << CaseArray[5] << endl; // Display Sixth Index Text From CaseArray
1494     }
1495 }
1496
1497

```

Figure 60: MainSelectionMenu function

```

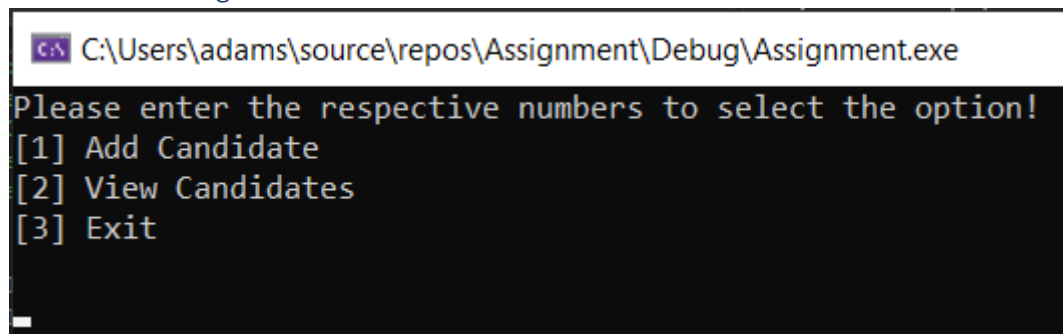
1498 int main()
1499 {
1500     VoterMainMenu();
1501
1502     return 0;
1503 }

```

Figure 61: main function

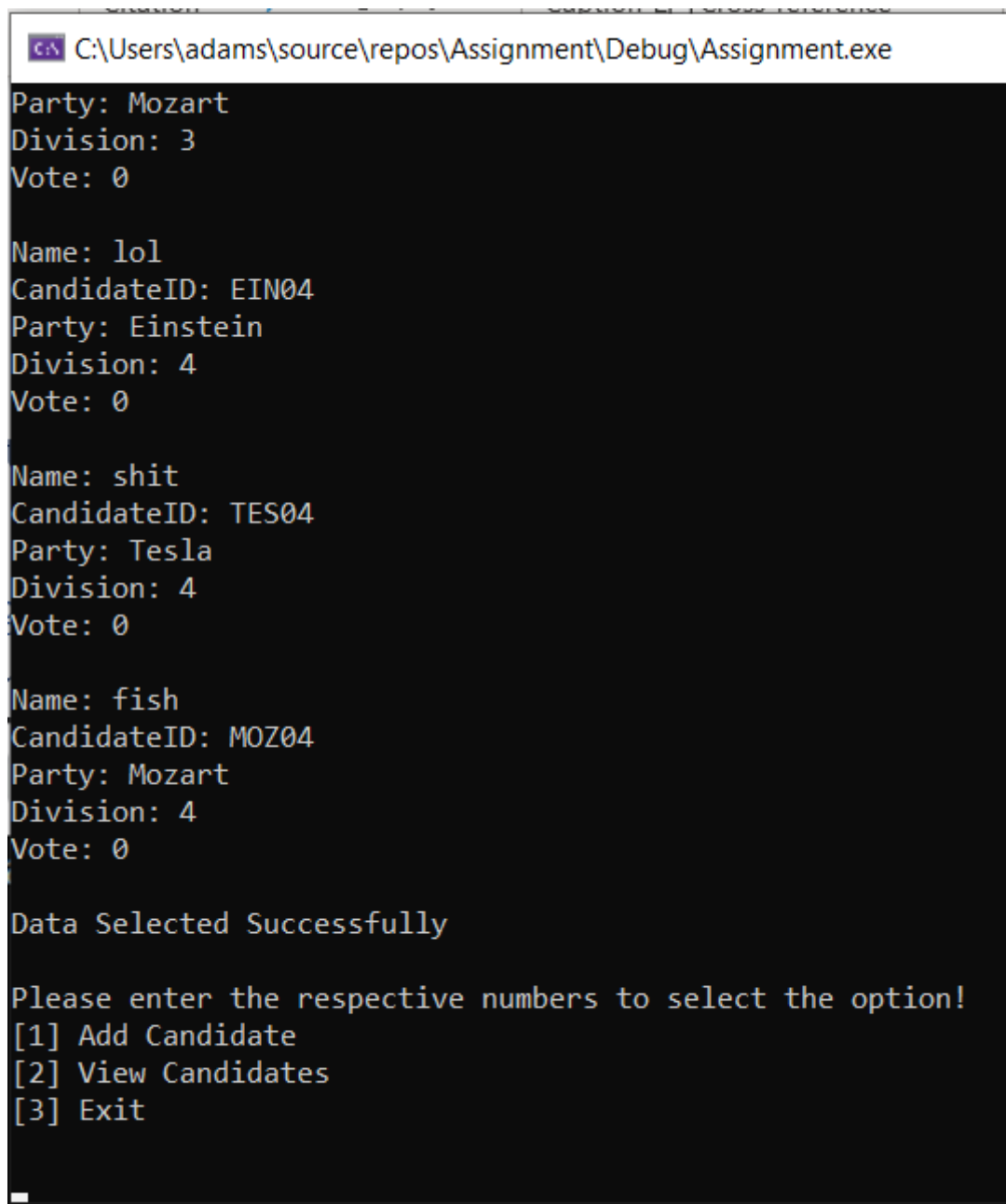
5.3 Program Walkthrough

5.3.1 Candidate Program



```
C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
Please enter the respective numbers to select the option!
[1] Add Candidate
[2] View Candidates
[3] Exit
```

Figure 62: Main Menu UI For Candidate Program



```
C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
Party: Mozart
Division: 3
Vote: 0

Name: lol
CandidateID: EIN04
Party: Einstein
Division: 4
Vote: 0

Name: shit
CandidateID: TES04
Party: Tesla
Division: 4
Vote: 0

Name: fish
CandidateID: MOZ04
Party: Mozart
Division: 4
Vote: 0

Data Selected Successfully

Please enter the respective numbers to select the option!
[1] Add Candidate
[2] View Candidates
[3] Exit
```

Figure 63: View Candidate UI

```
C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
Adding Candidate

Enter Your Name (With No Space):
```

```
C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
Data Inserted Successfully!

Please enter the respective numbers to select the option!
[1] Add Candidate
[2] View Candidates
[3] Exit
```

Figure 64: Add Candidate UI

```
Microsoft Visual Studio Debug Console

Exiting Program

C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
To automatically close the console when debugging stops,
press any key to close this window . . .
```

Figure 65: Exit Program UI

5.3.2 Voter Program

```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Please select the following options by typing in the number.
[1] View Candidates
[2] Register Voter
[3] Vote
[4] View voting results and summary
[5] Exit
```

Figure 66: Main Menu UI For Voter Program


```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Viewing Candidates

Please select the following options by typing in the number.
[1] View All Candidates In All Divisions
[2] View The Candidates In Specific Division
[3] View The Candidates Based On The Party
[4] Back To Main Menu
```

Figure 67: Viewing Candidates Menu UI

```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Which Division Would You Like To View ?
[1] Division 1
[2] Division 2
[3] Division 3
[4] Division 4
```

Figure 68: Viewing Candidates Based on Division UI

```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Which Party Would You Like To View?
[1] Einstein
[2] Tesla
[3] Mozart
```

Figure 69: Viewing Candidate Based on party UI

```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Registering as Voter

Enter Your First Name:

C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Enter Your Last Name:
_
```

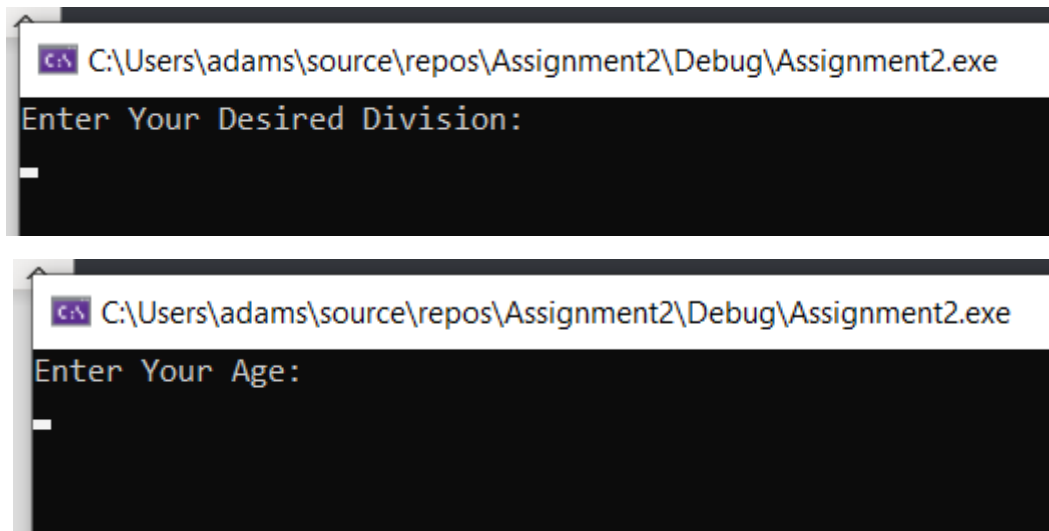


Figure 70: Register Voter Menu UI

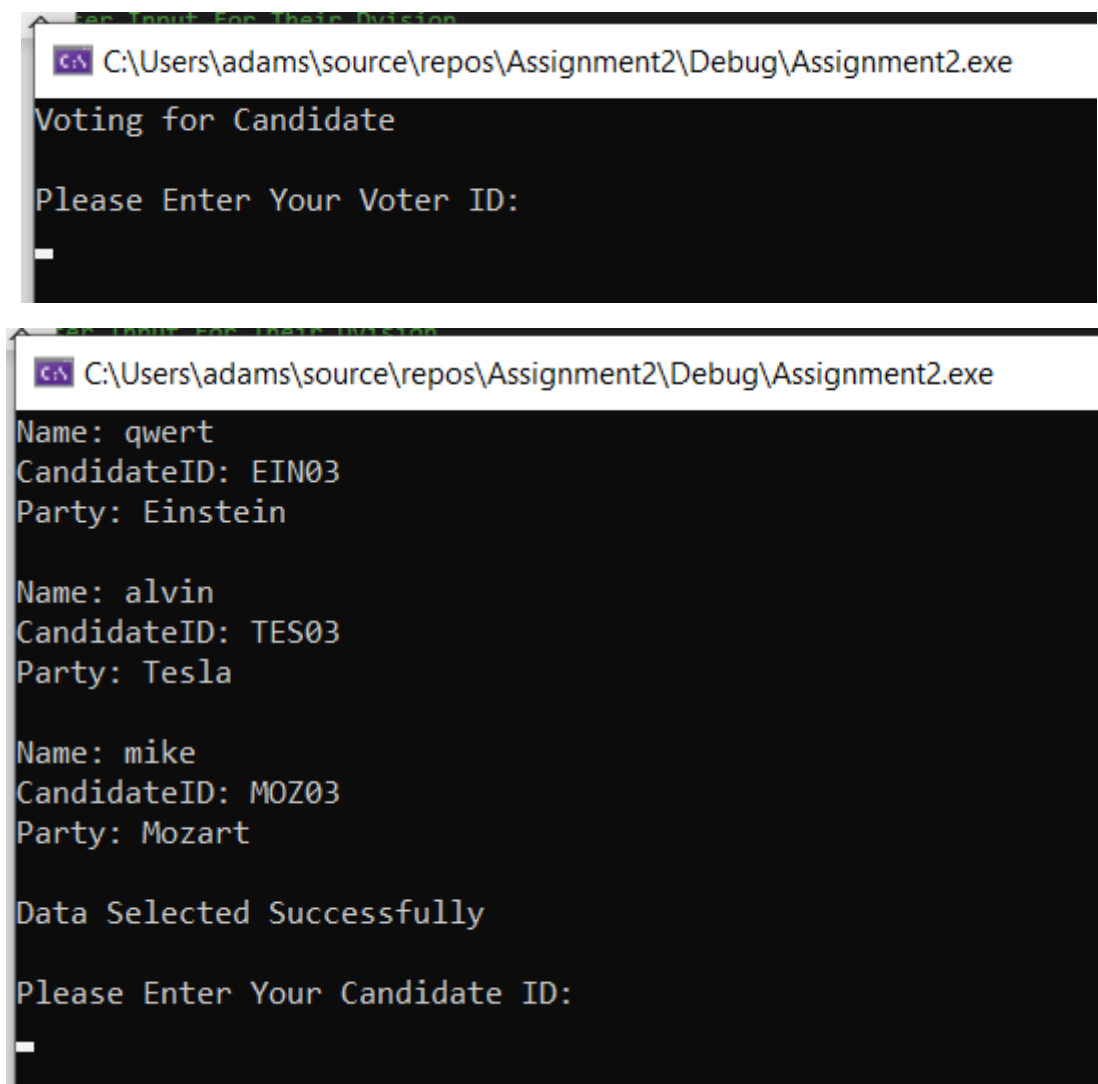


Figure 71: Vote Menu UI

```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Viewing voting results and summary

Please Select Your Result Viewing Options.
[1] View All Results From All Divisions
[2] View Results In A Specific Division
[3] Back To Main Menu
```

Figure 72: View Voting Results and Summary UI

```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Which Division's Information Would You Like To View?
[1] Division 1
[2] Division 2
[3] Division 3
[4] Division 4
```

Figure 73: View Voting Results and Summary Based on Division Menu UI

```
Microsoft Visual Studio Debug Console

Exiting Program

C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe (process 1344)
To automatically close the console when debugging stops, enable Tools->Options->
Press any key to close this window . . .
```

Figure 74: Exiting Program UI

5.4 Test Plan

5.4.1 Candidate Program

5.4.1.1 Main Menu

5.4.1.1.1 Valid Input

Choices	Expected Result
1	Go To Add Candidate Menu
2	Shows Candidate Data (Candidate Name, ID, Party, Division and Vote)
3	Exit the program

Table 1: Valid Input for Main Menu in Candidate Program

5.4.1.1.2 Invalid Input

Choices	Expected Result
Abc (any character)	Error Text show up and prompt user to enter valid input
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input

Table 2: Invalid Input for Main Menu in Candidate Program

5.4.1.2 Add Candidate Menu

5.4.1.2.1 Valid Input

Input	Expected Result
AdamSize (Any input with only characters and without white space)	Successfully add candidate Returns to Main Menu

Table 3: Valid Input for Add Candidate Menu

5.4.1.2.2 Invalid Input

Inputs	Expected Result
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input
White space	Error Text show up and prompt user to enter valid input

Table 4: Invalid Input for Add Candidate Menu

5.4.2 Voter Program

5.4.2.1 Main Menu

5.4.2.1.1 Valid Input

Choices	Expected Result
1	Go To View Candidate Menu
2	Go To Register Voter Menu
3	Go To Vote Menu
4	Go To View Voting Results and Summary Menu
5	Exit the program

Table 5: Valid Input for Main Menu in Voter Program

5.4.2.1.2 Invalid Input

Choices	Expected Result
Abc (any character)	Error Text show up and prompt user to enter valid input
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input

Table 6: Invalid Input for Main Menu in Voter Program

5.4.2.2 View Candidate Menu

5.4.2.2.1 Valid Input

Choices	Expected Result
1	View All Candidates in All Division
2	Go To View All Candidates in Specific Division
3	Go To View All Candidates Based on Party
4	Return to Main Menu

Table 7: Valid Input for View Candidate Menu

5.4.2.2.2 Invalid Input

Inputs	Expected Result
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input
White space	Error Text show up and prompt user to enter valid input

Table 8: Invalid Input for View Candidate Menu

5.4.2.3 View Candidate in Specific Division Menu

5.4.2.3.1 Valid Input

Choices	Expected Result
1	View All Candidates in Division 1
2	View All Candidates in Division 2
3	View All Candidates in Division 3
4	View All Candidates in Division 4

Table 9: Valid Input for View Candidate in Specific Division Menu

5.4.2.3.2 Invalid Input

Inputs	Expected Result
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input
White space	Error Text show up and prompt user to enter valid input

Table 10: Invalid Input for View Candidate in Specific Division Menu

5.4.2.4 View Candidate Based on Party Menu

5.4.2.4.1 Valid Input

Choices	Expected Result
1	View All Candidates in Party Einstein
2	View All Candidates in Party Tesla
3	View All Candidates in Party Mozart

Table 11: Valid Input for View Candidate Based on Party Menu

5.4.2.4.2 Invalid Input

Inputs	Expected Result
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input

! @ # (Special Characters)	Error Text show up and prompt user to enter valid input
White space	Error Text show up and prompt user to enter valid input

Table 12: Invalid Input for View Candidate Based on Party Menu

5.4.2.5 Register Voter Menu

5.4.2.5.1 Valid Input

Inputs		Expected Result
Enter Your First Name:	Max	Successfully Registered as Voter and will return to Main Menu
Enter Your Last Name:	Low	
Enter Your Desired Division:	4	
Enter Your Age:	21	
Enter Your First Name:	Max	Too Young to Register as Voter and will Return to Main Menu
Enter Your Last Name:	Low	
Enter Your Desired Division:	4	
Enter Your Age:	10	

Table 13: Valid Input for Register Voter Menu

5.4.2.5.2 Invalid Input

Inputs		Expected Result
Enter Your First Name:	! @ # (Special Characters) 123 (Integers)	Error text will appear and prompt user to retry
Enter Your Last Name:	! @ # (Special Characters) 123 (Integers)	
Enter Your Desired Division:	! @ # (Special Characters) abc (characters) 123 (Integers that is not 1 to 5)	
Enter Your Age:	! @ # (Special Characters) abc (characters) 123 (Integers that is not 1 to 5)	

Table 14: Invalid Input for Register Voter Menu

5.4.2.6 Vote Menu

5.4.2.6.1 Valid Input

Inputs		Expected Result
Please Enter Your Voter ID:	MaxLow	Show Candidates Info in the same division as the voter
Please Enter Your Candidate ID:	EIN03	Voted for EIN03 and will return to Main Menu Update Voter Status from “N” to “Y” Update Candidates Vote Count

Table 15: Valid Input for Vote Menu

5.4.2.6.2 Invalid Input

Inputs	Expected Result
--------	-----------------

Please Enter Your Voter ID:	Any Input That Is Not the Same as Voter ID	Error Text Saying Voter ID Does Not Exist and Return to Main Menu
Please Enter Your Candidate ID:	Any Input That Is Not the Same as Candidate ID	Error Text Saying Candidate ID Does Not Exist and Return to Main Menu
Please Enter Your Voter ID:	MaxLow (If Voter Already Voted Before)	Show Error Text Saying Voter Already Voted and Return to Main Menu

Table 16: Invalid Input for Vote Menu

5.4.2.7 View Voting Results and Summary Menu

5.4.2.7.1 Valid Input

Choices	Expected Result
1	View All Results from All Division
2	Go To View Results in Specific Division Menu
3	Return to Main Menu

Table 17: Valid Input for View Voting Results and Summary Menu

5.4.2.7.2 Invalid Input

Inputs	Expected Result
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input
White space	Error Text show up and prompt user to enter valid input

Table 18: Invalid Input for View Voting Results and Summary Menu

5.4.2.8 View Voting Results and Summary in Specific Division Menu

5.4.2.8.1 Valid Input

Choices	Expected Result
1	View Voting Results in Division 1
2	View Voting Results in Division 2
3	View Voting Results in Division 3
4	View Voting Results in Division 4

Table 19: Valid Input for View Voting Results and Summary in Specific Division Menu

5.4.2.8.2 Invalid Input

Inputs	Expected Result
123 (any integer that is not 1 to 5)	Error Text show up and prompt user to enter valid input
! @ # (Special Characters)	Error Text show up and prompt user to enter valid input
White space	Error Text show up and prompt user to enter valid input

Table 20: Invalid Input for View Voting Results and Summary in Specific Division Menu

6 Project Outcomes

6.1.1 Candidate Program

6.1.1.1 Main Menu

User will see three options available. The first option will be to add candidates, the second option will be to view candidates and the third option will be to exit the program.

6.1.1.2 Add Candidate Menu

Users will be able to register as a candidate by inputting their name into the program when prompted. After that, the user will return to the main menu.

6.1.1.3 View Candidate Menu

Users will be shown all the registered candidate's information such as their name, candidate ID, party, division and vote count.

6.1.2 Voter Program

6.1.2.1 Main Menu

Users will see 5 options available. The first option will be to view candidates, second option will be to register as a voter, third option will be to vote, fourth option will be to view voting results and summary and the fifth option will be to exit the program.

6.1.2.2 View Candidate Menu

Users will see 4 options. The first option is to view all candidates in all division, the second option is to view all candidate in a specific division. The user will be prompt with an input to choose which division the user would like to view. The third option is to view all candidates based on party. The user will be prompt to choose which party the user would like to view. Lastly the fourth option is to go back to the main menu.

6.1.2.3 Register Voter Menu

Users will be prompt to enter their first name, last name, desired division and age to register as a voter. If the users age is below 19 years old, the user will be returned to the main menu with a text saying that the user is not eligible to be a voter.

6.1.2.4 Vote Menu

Users will be prompt to enter their voter ID, which is their full name. If the voter ID exists, the user will be shown a list of candidates and their information who are in the same division as the voter. To vote, the user will input the candidate ID whom they would want to vote. If their input is valid, the program will show that they have voted for the candidate and return to the main menu. For invalid inputs, the program will display a text saying that the voter ID or Candidate ID does not exists and return back to the main menu. If the user has voted, the program will return to the main menu and display a text saying that the user have voted.

6.1.2.5 View Voting Results and Summary Menu

Users will be shown 3 options. The option will display the candidate who have the maximum and minimum vote by displaying their name, candidate ID, party, vote count, percentage of vote and the total of vote. The first option is to view all candidates in all division. The second option is to view all the candidates based on division where the user is prompt to choose between four options each being Division 1 to Division 4.