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/lIZ-1iOvZ4s

NOVEMBER 28, 2021
SWINBURNE UNIVERSITY OF TEHCNOLOGY SARAWAK

1 Table of Contents

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

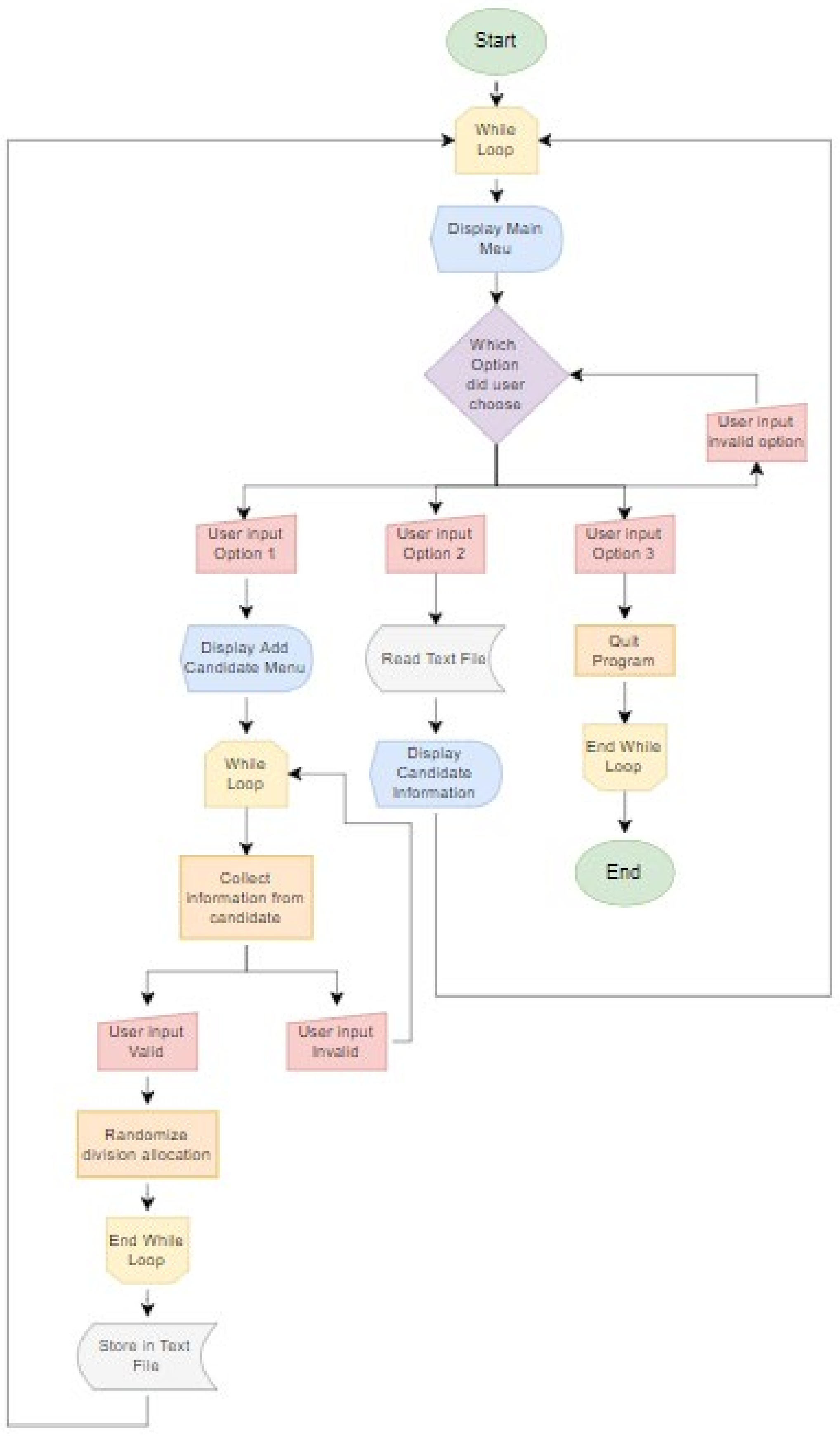


Figure 1: Flowchart for Candidate Program

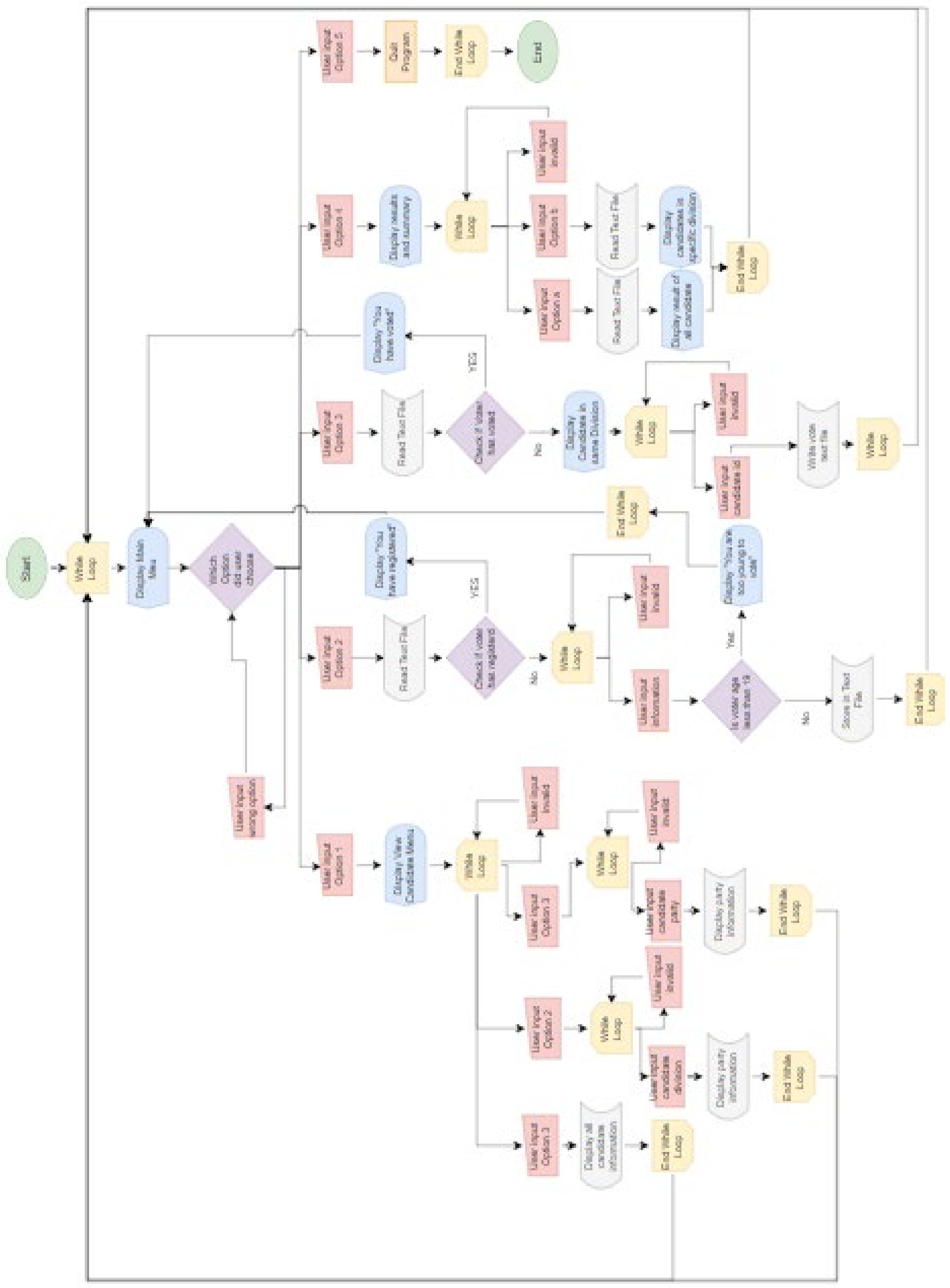


Figure 2: Flowchart for Voter Program

5 Compatibility of Design

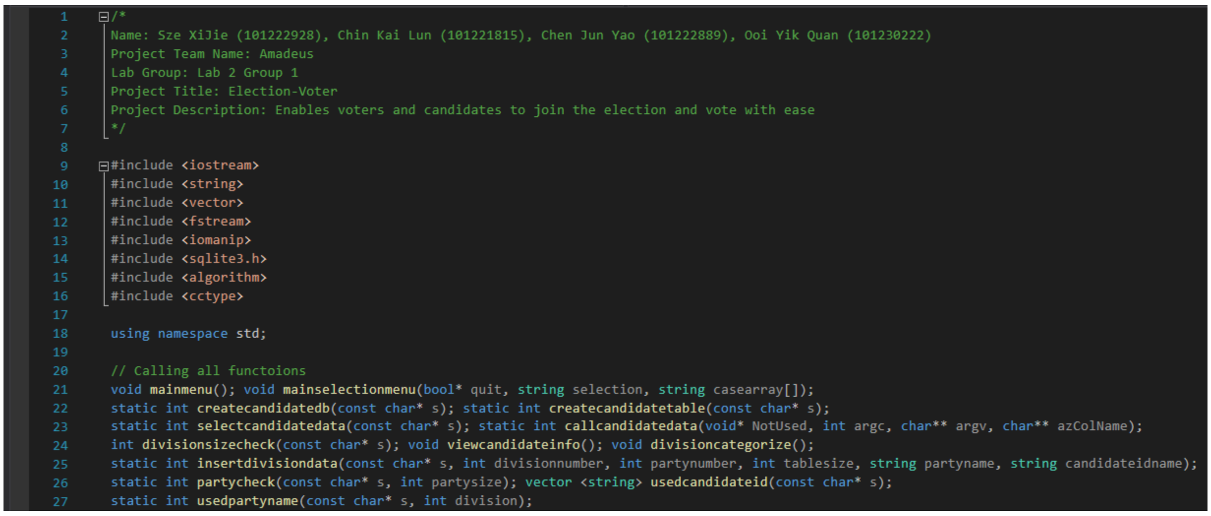
5.1 How data is Stored

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.

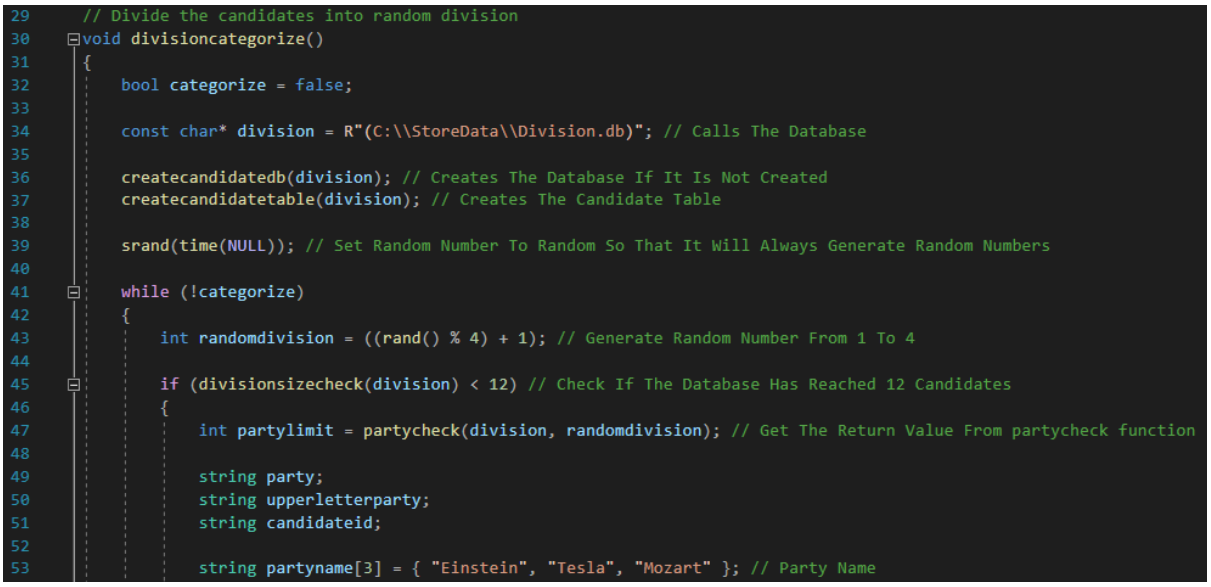
5.2 Code Explanation

5.2.1 Candidate Program

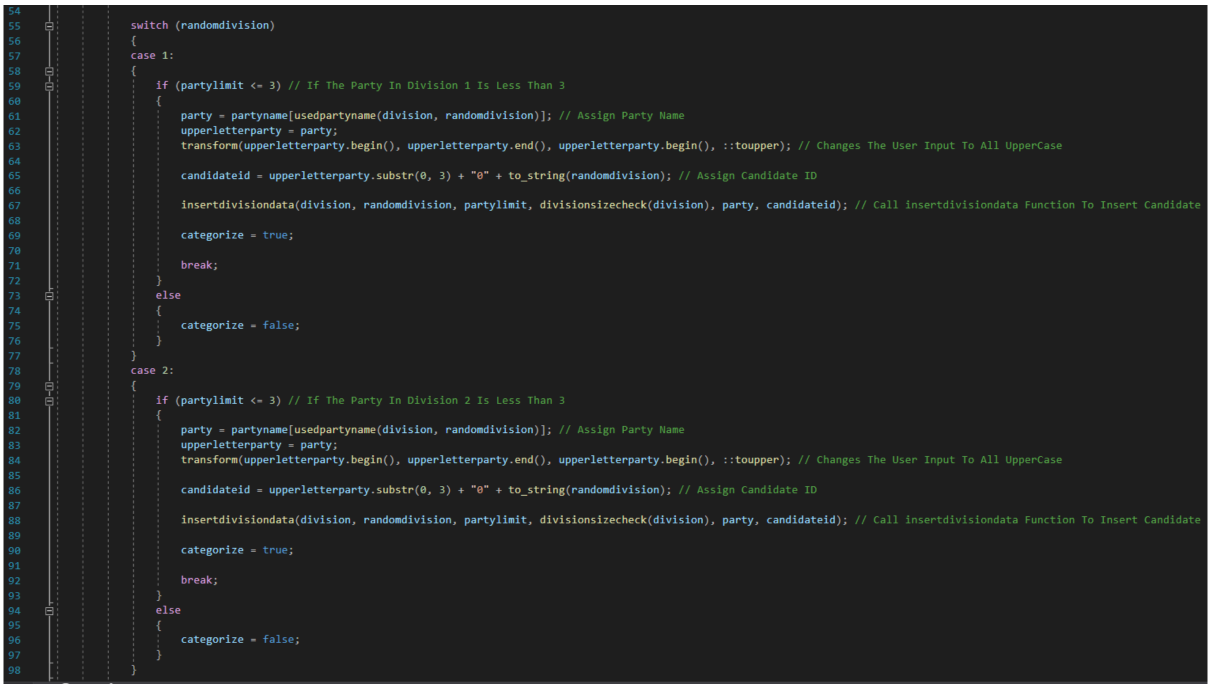


```cpp
/*
Name: Sze XiJie (101222928), Chin Kai Lun (101221815), Chen Jun Yao (101222889), Ooi Yik Quan (101230222)
Project Team Name: Amadeus
Lab Group: Lab 2 Group 1
Project Title: Election-Voter
Project Description: Enables voters and candidates to join the election and vote with ease
*/

#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <sqlite3.h>
#include <algorithm>
#include <cctype>

using namespace std;

// Calling all functoions
void mainmenu(); void mainselectionmenu(bool* quit, string selection, string casearray[]);
static int createcandidatedb(const char* s); static int createcandidatetable(const char* s);
static int selectcandidatedata(const char* s); static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName);
int divisionsizecheck(const char* s); void viewcandidateinfo(); void divisioncategorize();
static int insertdivisiondata(const char* s, int divisionnumber, int partynumber, int tablesize, string partyname, string candidateidname);
static int partycheck(const char* s, int partysize); vector <string> usedcandidateid(const char* s);
static int usedpartyname(const char* s, int division);
```

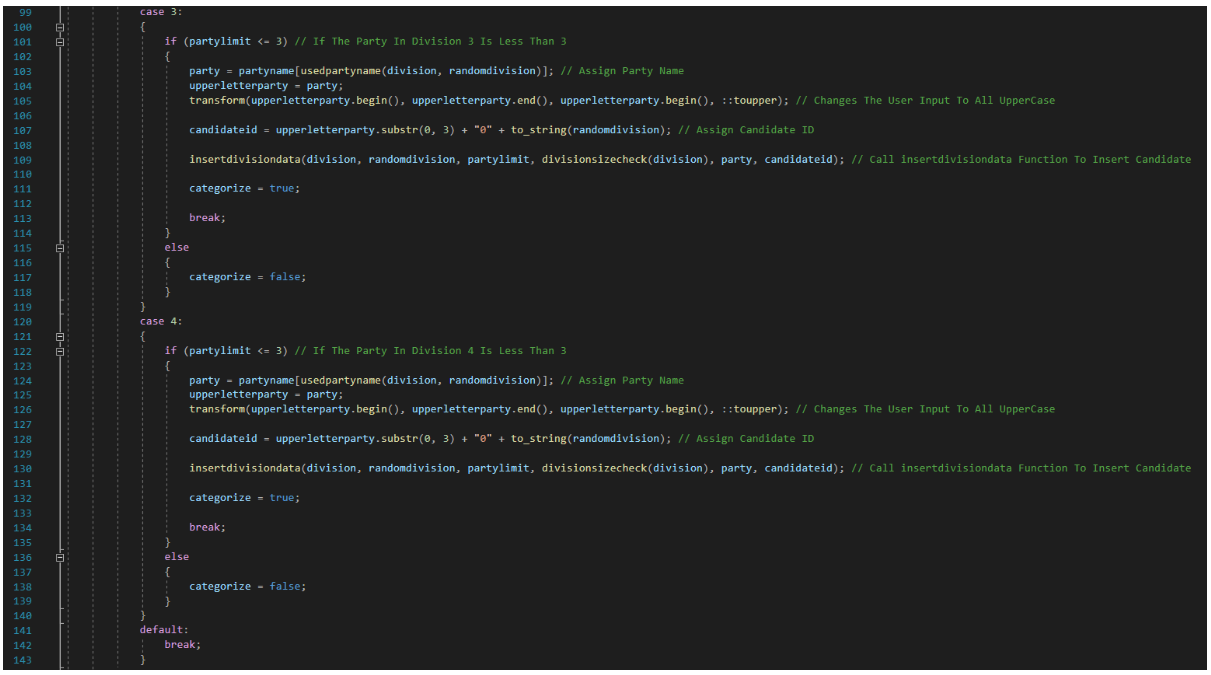
Figure 3: Program Description for Candidate Program



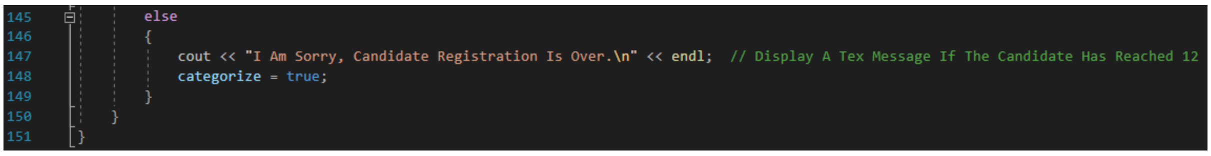
```cpp
// Divide the candidates into random division
void divisioncategorize()
{
    bool categorize = false;

    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    createcandidatedb(division); // Creates The Database If It Is Not Created
    createcandidatetable(division); // Creates The Candidate Table

    srand(time(NULL)); // Set Random Number To Random So That It Will Always Generate Random Numbers

    while (!categorize)
    {
        int randomdivision = ((rand() % 4) + 1); // Generate Random Number From 1 To 4

        if (divisionsizecheck(division) < 12) // Check If The Database Has Reached 12 Candidates
        {
            int partylimit = partycheck(division, randomdivision); // Get The Return Value From partycheck function

            string party;
            string upperletterparty;
            string candidateid;

            string partyname[3] = { "Einstein", "Tesla", "Mozart" }; // Party Name
```



```cpp

            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;
                }
            }
```

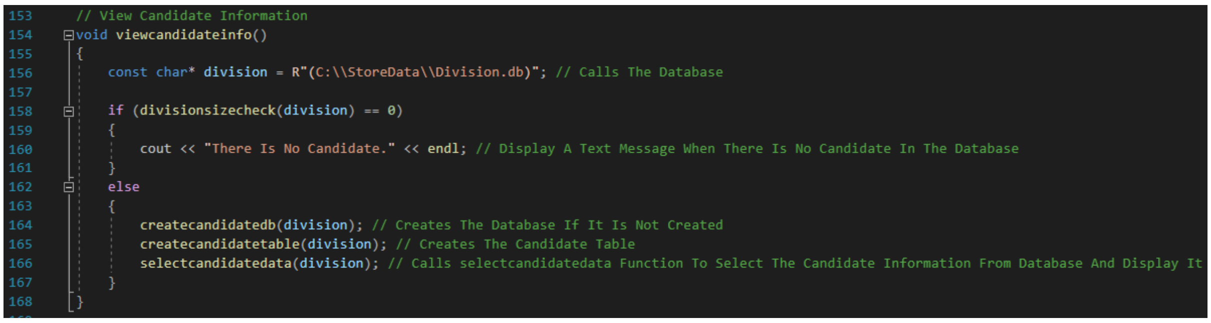


```cpp
            case 3:
            {
                if (partylimit <= 3) // If The Party In Division 3 Is Less Than 3
                {
                    party = partyname[usedpartyname(division, randomdivision)]; // Assign Party Name
                    upperletterparty = party;
                    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;
            }
```



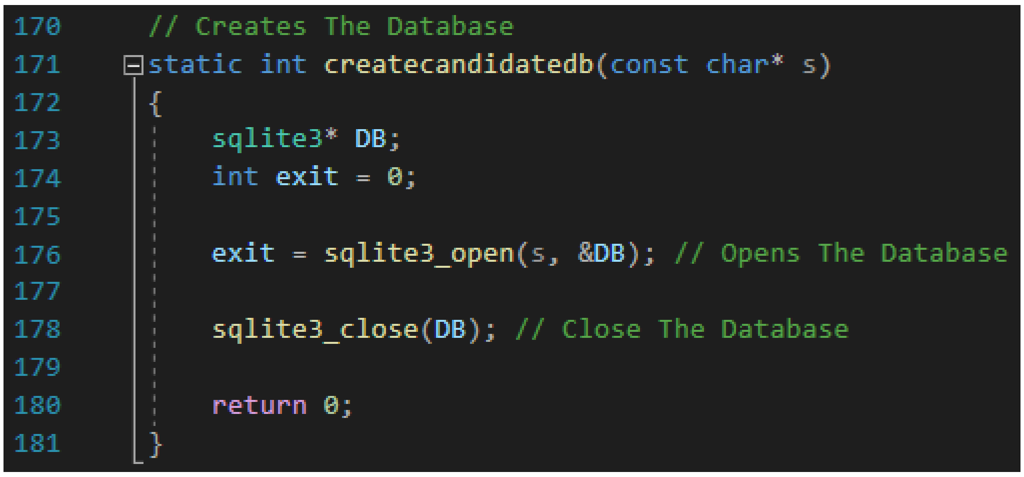
```cpp
        else
        {
            cout << "I Am Sorry, Candidate Registration Is Over.\n" << endl;  // Display A Tex Message If The Candidate Has Reached 12
            categorize = true;
        }
    }
}
```

Figure 4: divisioncategorize function



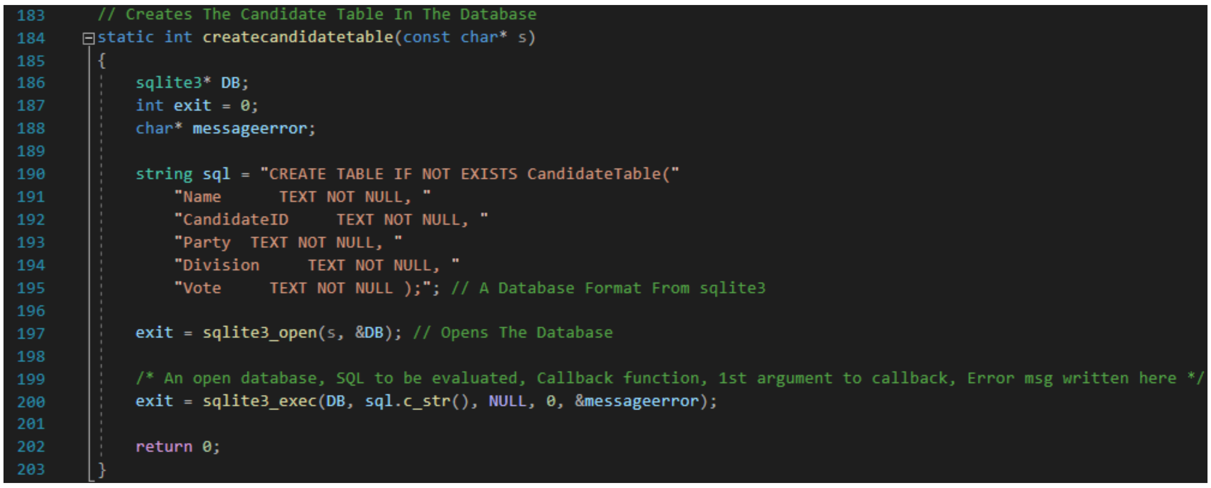
```cpp
// View Candidate Information
void viewcandidateinfo()
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    if (divisionsizecheck(division) == 0)
    {
        cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
    }
    else
    {
        createcandidatedb(division); // Creates The Database If It Is Not Created
        createcandidatetable(division); // Creates The Candidate Table
        selectcandidatedata(division); // Calls selectcandidatedata Function To Select The Candidate Information From Database And Display It
    }
}
```

Figure 5: viewcandidateinfo function



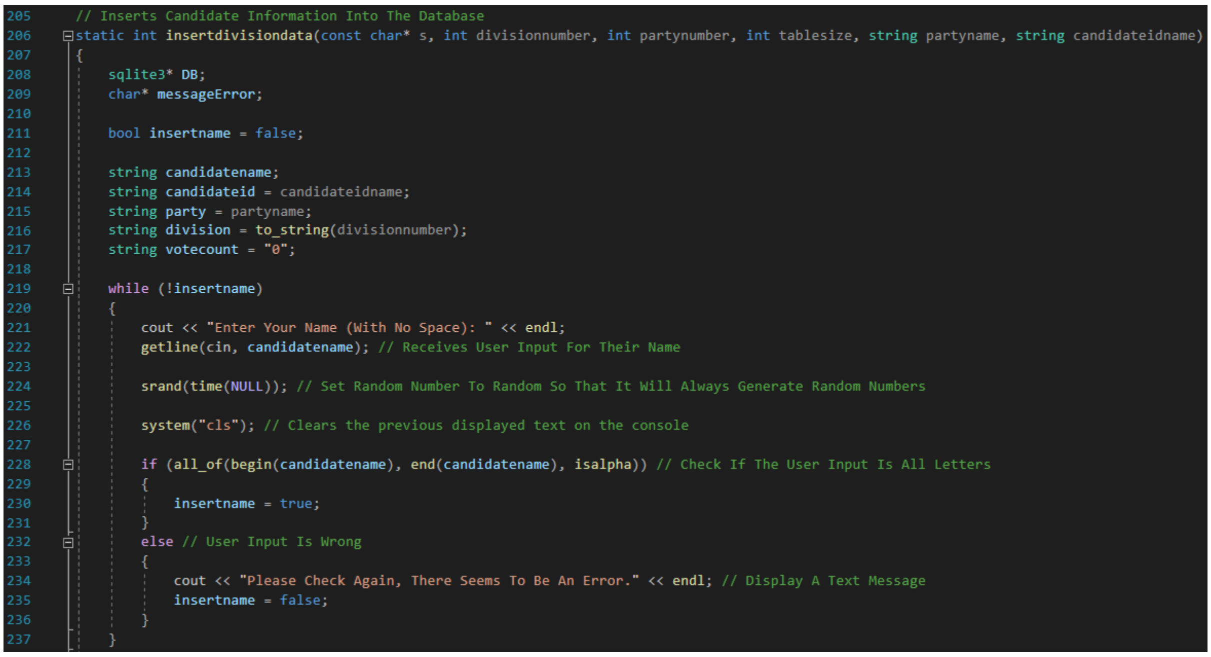
```cpp
// Creates The Database
static int createcandidatedb(const char* s)
{
    sqlite3* DB;
    int exit = 0;

    exit = sqlite3_open(s, &DB); // Opens The Database

    sqlite3_close(DB); // Close The Database

    return 0;
}
```

Figure 6: createcandidatedb function

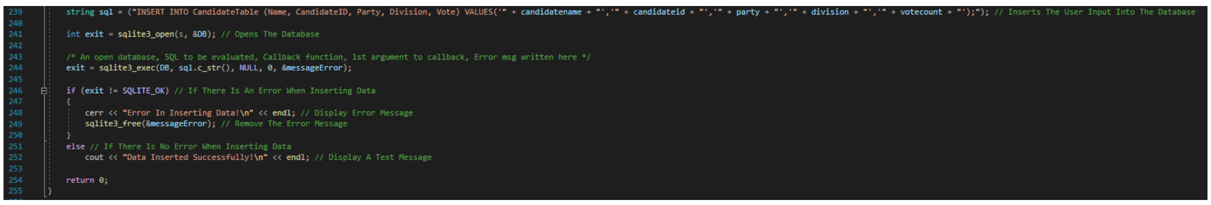


```cpp
// Creates The Candidate Table In The Database
static int createcandidatetable(const char* s)
{
    sqlite3* DB;
    int exit = 0;
    char* messageerror;

    string sql = "CREATE TABLE IF NOT EXISTS CandidateTable("
        "Name       TEXT NOT NULL, "
        "CandidateID      TEXT NOT NULL, "
        "Party  TEXT NOT NULL, "
        "Division      TEXT NOT NULL, "
        "Vote      TEXT NOT NULL );"; // A Database Format From sqlite3

    exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageerror);

    return 0;
}
```

Figure 7: createcandidatetable function

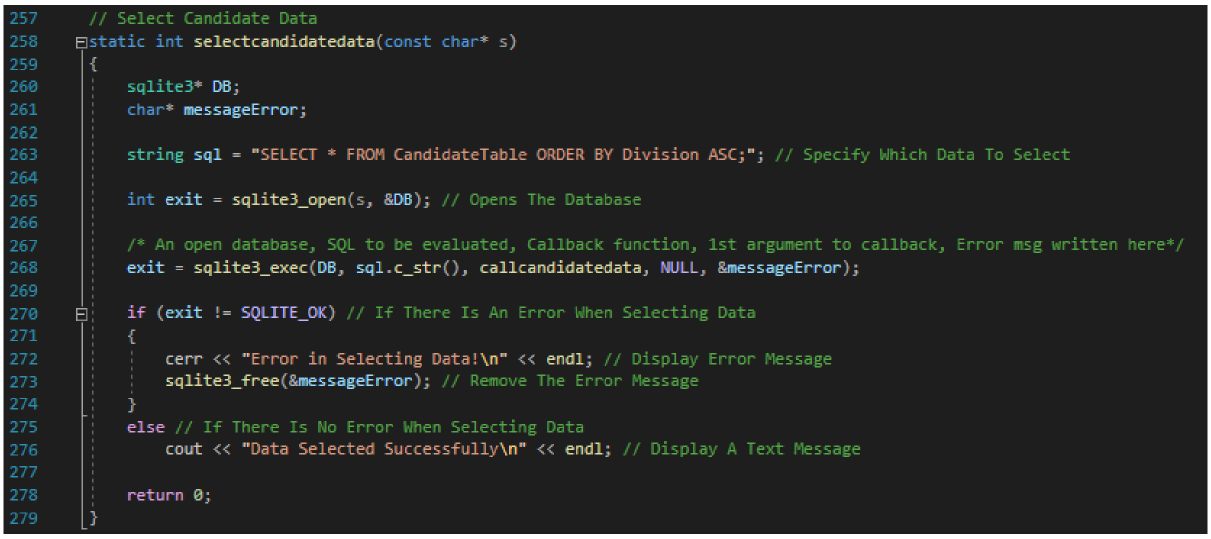


```cpp
// Inserts Candidate Information Into The Database
static int insertdivisiondata(const char* s, int divisionnumber, int partynumber, int tablesize, string partyname, string candidateidname)
{
    sqlite3* DB;
    char* messageError;

    bool insertname = false;

    string candidatename;
    string candidateid = candidateidname;
    string party = partyname;
    string division = to_string(divisionnumber);
    string votecount = "0";

    while (!insertname)
    {
        cout << "Enter Your Name (With No Space): " << endl;
        getline(cin, candidatename); // Receives User Input For Their Name

        srand(time(NULL)); // Set Random Number To Random So That It Will Always Generate Random Numbers

        system("cls"); // Clears the previous displayed text on the console

        if (all_of(begin(candidatename), end(candidatename), isalpha)) // Check If The User Input Is All Letters
        {
            insertname = true;
        }
        else // User Input Is Wrong
        {
            cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
            insertname = false;
        }
    }
```



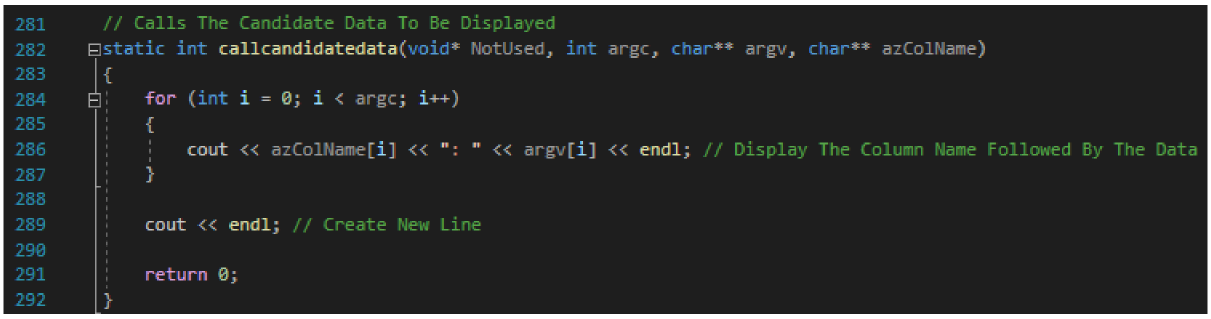
```cpp
    string sql = ("INSERT INTO CandidateTable (Name, CandidateID, Party, Division, Vote) VALUES('" + candidatename + "','" + candidateid + "','" + party + "','" + division + "','" + votecount + "');"); // Inserts The User Input Into The Database

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Inserting Data
    {
        cerr << "Error In Inserting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Inserting Data
        cout << "Data Inserted Successfully!\n" << endl; // Display A Text Message

    return 0;
}
```

Figure 8: insertdivisiondata function



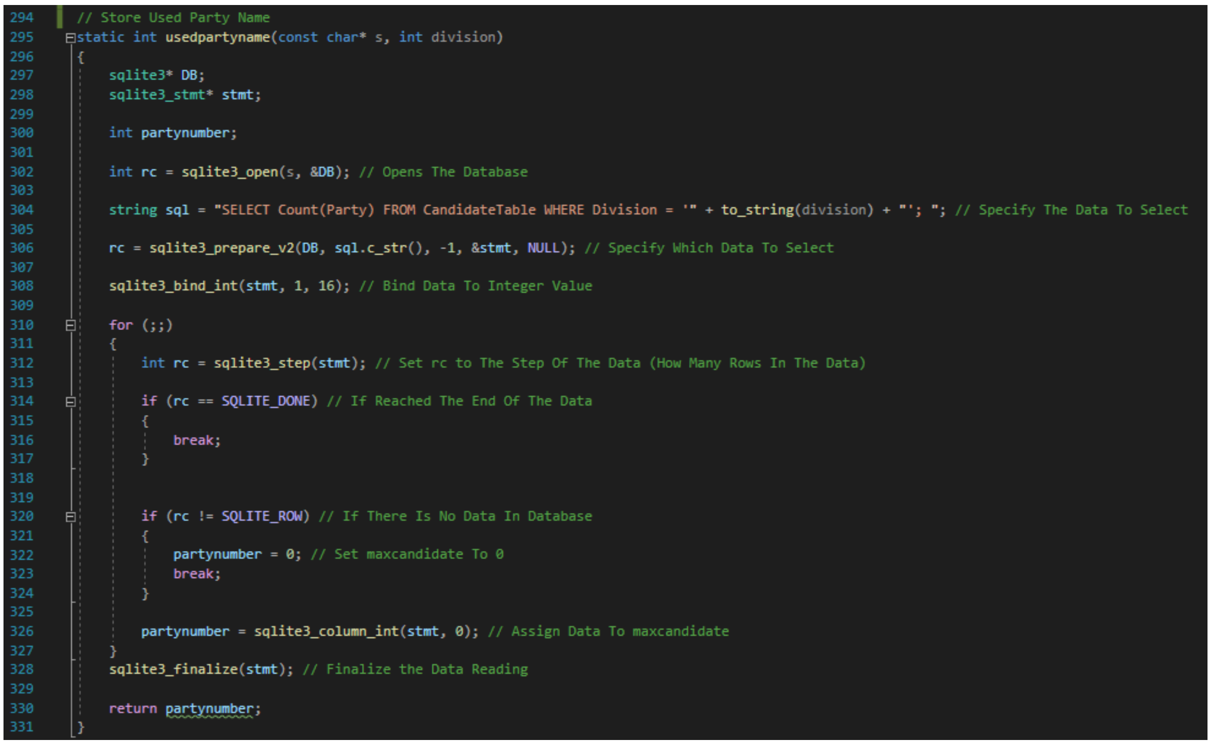
```cpp
// Select Candidate Data
static int selectcandidatedata(const char* s)
{
    sqlite3* DB;
    char* messageError;

    string sql = "SELECT * FROM CandidateTable ORDER BY Division ASC;"; // Specify Which Data To Select

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

Figure 9: selectcandidatedata function



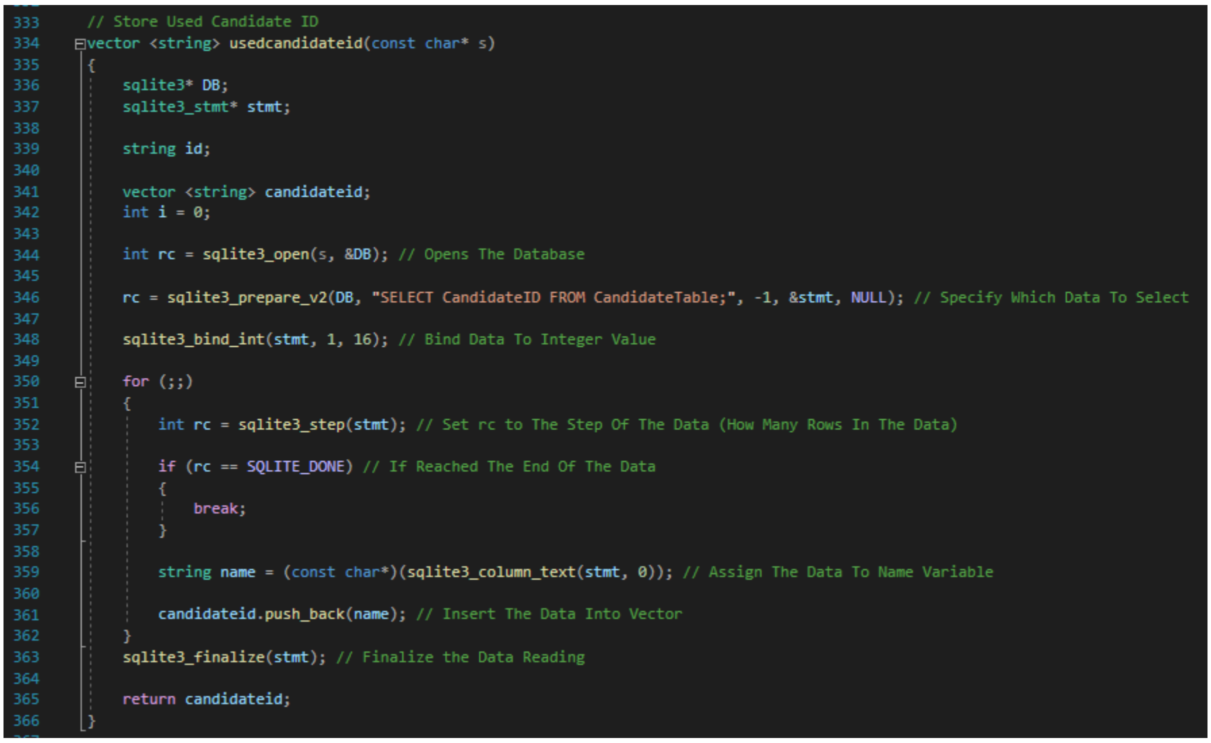
```cpp
// Calls The Candidate Data To Be Displayed
static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName)
{
    for (int i = 0; i < argc; i++)
    {
        cout << azColName[i] << ": " << argv[i] << endl; // Display The Column Name Followed By The Data
    }

    cout << endl; // Create New Line

    return 0;
}
```

Figure 10: callcandidatedata function



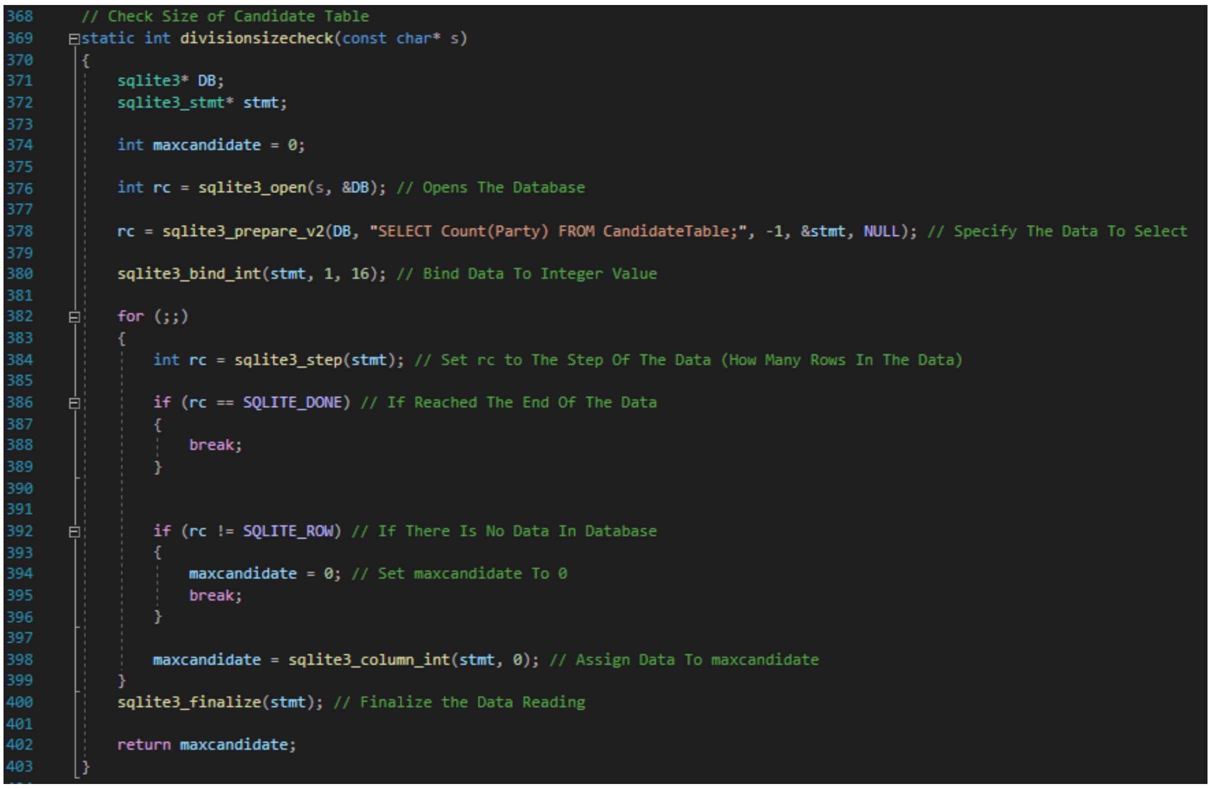
```cpp
// Store Used Party Name
static int usedpartyname(const char* s, int division)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    int partynumber;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }


        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            partynumber = 0; // Set maxcandidate To 0
            break;
        }

        partynumber = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return partynumber;
}
```

Figure 11: usedpartyname function



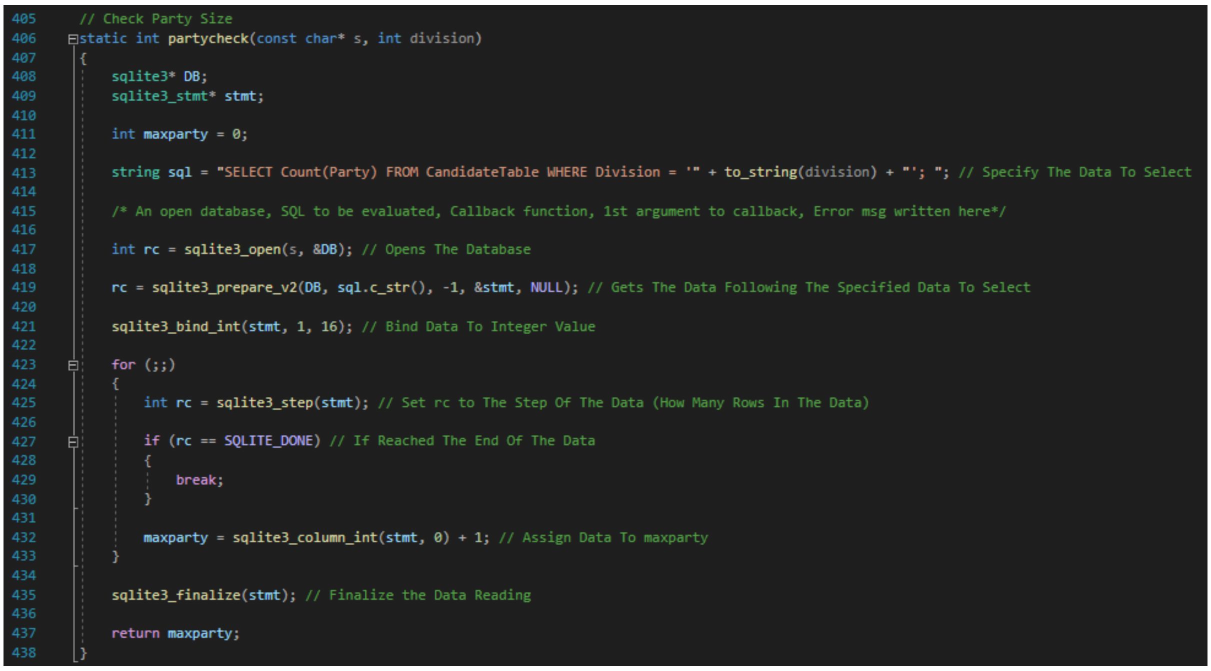
```cpp
// Store Used Candidate ID
vector <string> usedcandidateid(const char* s)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    string id;

    vector <string> candidateid;
    int i = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, "SELECT CandidateID FROM CandidateTable;", -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        string name = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable

        candidateid.push_back(name); // Insert The Data Into Vector
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return candidateid;
}
```

Figure 12: usedcandidateid function



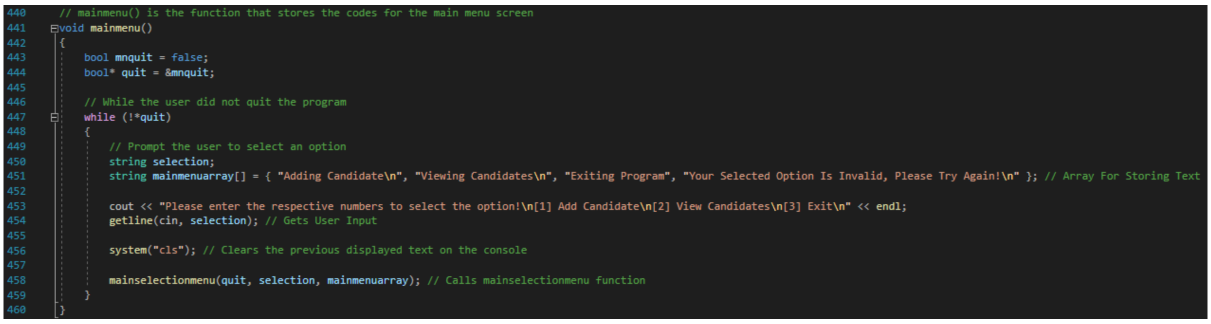
```cpp
// Check Size of Candidate Table
static int divisionsizecheck(const char* s)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    int maxcandidate = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, "SELECT Count(Party) FROM CandidateTable;", -1, &stmt, NULL); // Specify The Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }


        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            maxcandidate = 0; // Set maxcandidate To 0
            break;
        }

        maxcandidate = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return maxcandidate;
}
```

Figure 13: divisionsizecheck function



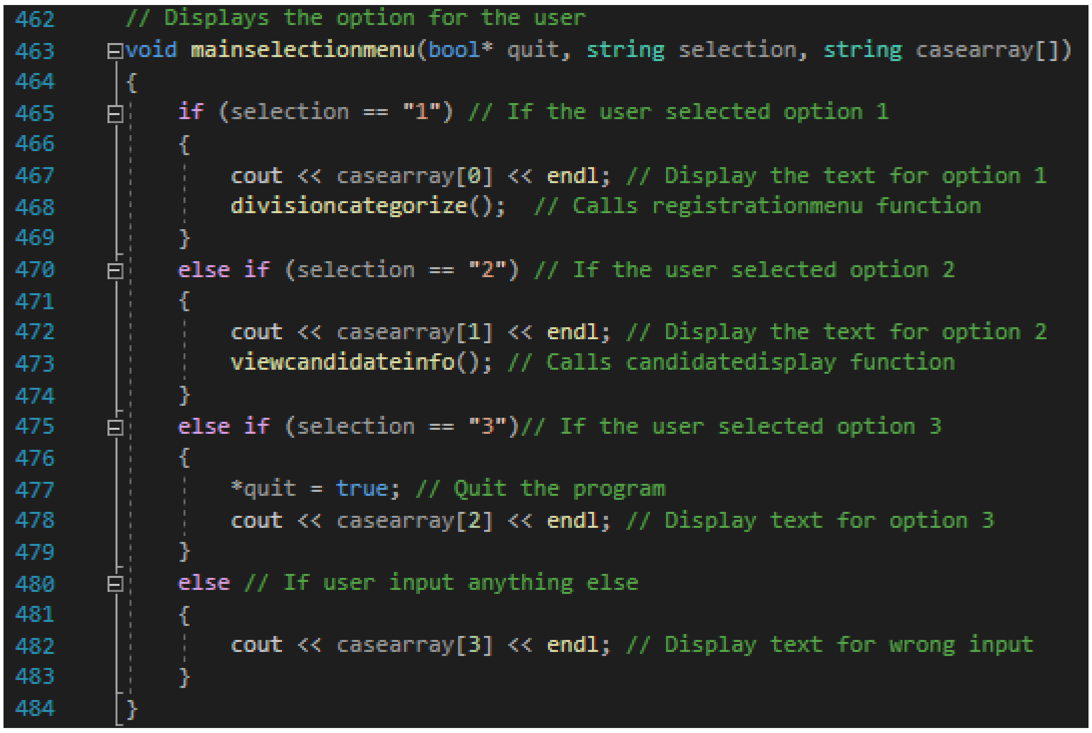
```cpp
// Check Party Size
static int partycheck(const char* s, int division)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    int maxparty = 0;

    string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'; "; // Specify The Data To Select

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Gets The Data Following The Specified Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        maxparty = sqlite3_column_int(stmt, 0) + 1; // Assign Data To maxparty
    }

    sqlite3_finalize(stmt); // Finalize the Data Reading

    return maxparty;
}
```

Figure 14: partycheck function



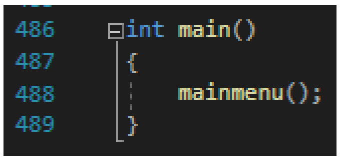
```cpp
// mainmenu() is the function that stores the codes for the main menu screen
void mainmenu()
{
    bool mnquit = false;
    bool* quit = &mnquit;

    // While the user did not quit the program
    while (!*quit)
    {
        // Prompt the user to select an option
        string selection;
        string mainmenuarray[] = { "Adding Candidate\n", "Viewing Candidates\n", "Exiting Program", "Your Selected Option Is Invalid, Please Try Again!\n" }; // Array For Storing Text

        cout << "Please enter the respective numbers to select the option!\n[1] Add Candidate\n[2] View Candidates\n[3] Exit\n" << endl;
        getline(cin, selection); // Gets User Input

        system("cls"); // Clears the previous displayed text on the console

        mainselectionmenu(quit, selection, mainmenuarray); // Calls mainselectionmenu function
    }
}
```

Figure 15: mainmenu function



```cpp
    // Displays the option for the user
void mainselectionmenu(bool* quit, string selection, string casearray[])
{
    if (selection == "1") // If the user selected option 1
    {
        cout << casearray[0] << endl; // Display the text for option 1
        divisioncategorize();  // Calls registrationmenu function
    }
    else if (selection == "2") // If the user selected option 2
    {
        cout << casearray[1] << endl; // Display the text for option 2
        viewcandidateinfo(); // Calls candidatedisplay function
    }
    else if (selection == "3")// If the user selected option 3
    {
        *quit = true; // Quit the program
        cout << casearray[2] << endl; // Display text for option 3
    }
    else // If user input anything else
    {
        cout << casearray[3] << endl; // Display text for wrong input
    }
}
```

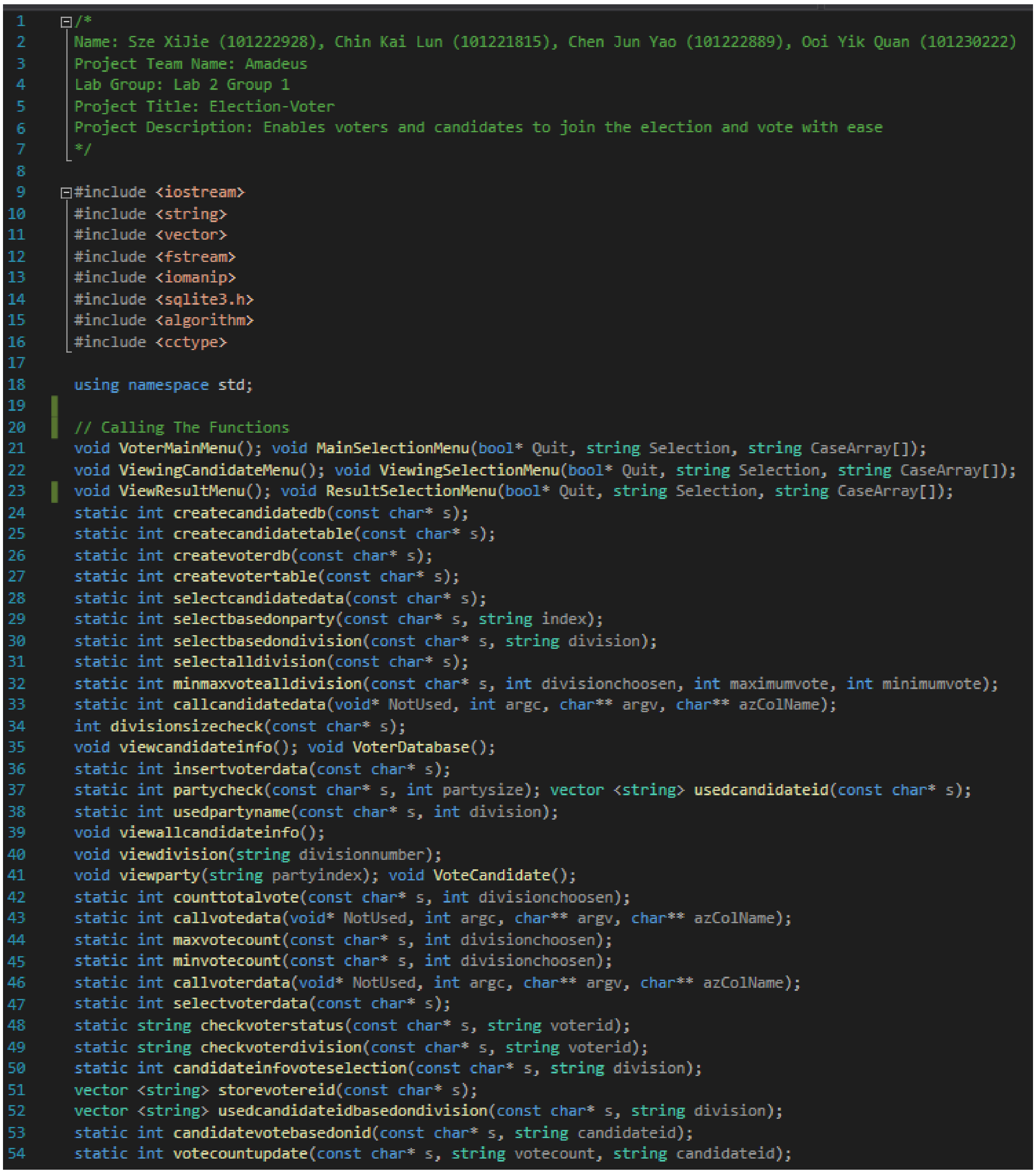
Figure 16: mainselectionmenu function



```cpp
int main()
{
    mainmenu();
}
```

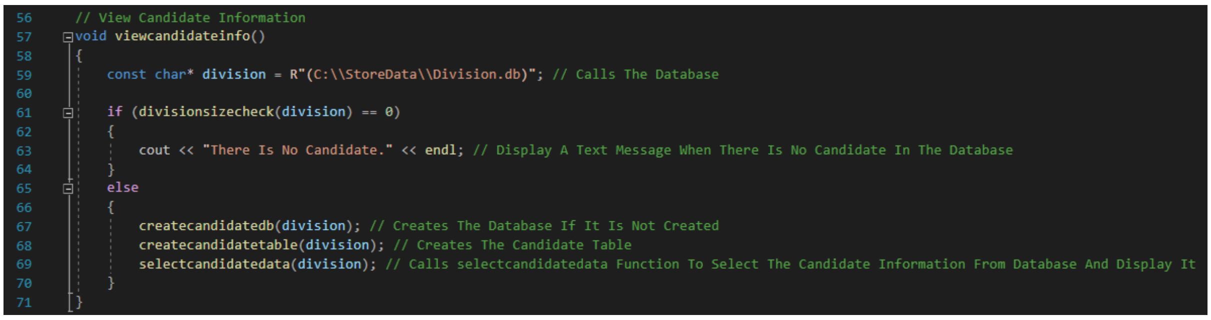
Figure 17: main function

5.2.2 Voter Program



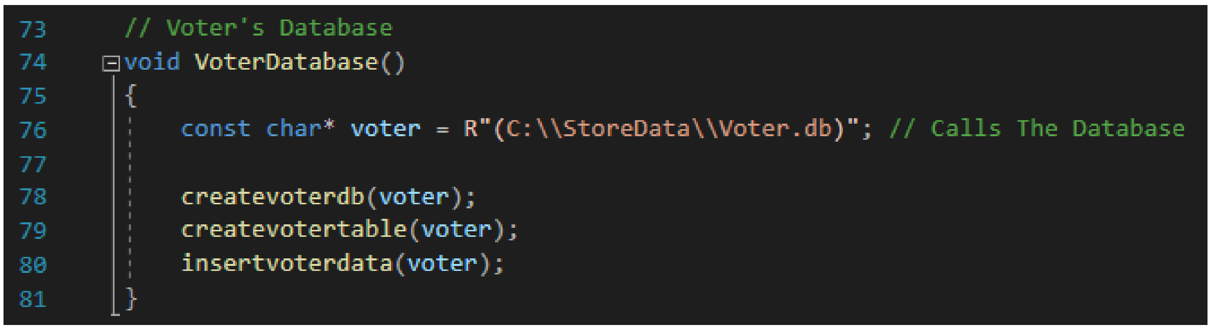
```cpp
/*
Name: Sze XiJie (101222928), Chin Kai Lun (101221815), Chen Jun Yao (101222889), Ooi Yik Quan (101230222)
Project Team Name: Amadeus
Lab Group: Lab 2 Group 1
Project Title: Election-Voter
Project Description: Enables voters and candidates to join the election and vote with ease
*/

#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <sqlite3.h>
#include <algorithm>
#include <cctype>

using namespace std;

// Calling The Functions
void VoterMainMenu(); void MainSelectionMenu(bool* Quit, string Selection, string CaseArray[]);
void ViewingCandidateMenu(); void ViewingSelectionMenu(bool* Quit, string Selection, string CaseArray[]);
void ViewResultMenu(); void ResultSelectionMenu(bool* Quit, string Selection, string CaseArray[]);
static int createcandidatedb(const char* s);
static int createcandidatetable(const char* s);
static int createvoterdb(const char* s);
static int createvotertable(const char* s);
static int selectcandidatedata(const char* s);
static int selectbasedonparty(const char* s, string index);
static int selectbasedondivision(const char* s, string division);
static int selectalldivision(const char* s);
static int minmaxvotealldivision(const char* s, int divisionchoosen, int maximumvote, int minimumvote);
static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName);
int divisionsizecheck(const char* s);
void viewcandidateinfo(); void VoterDatabase();
static int insertvoterdata(const char* s);
static int partycheck(const char* s, int partysize); vector <string> usedcandidateid(const char* s);
static int usedpartyname(const char* s, int division);
void viewallcandidateinfo();
void viewdivision(string divisionnumber);
void viewparty(string partyindex); void VoteCandidate();
static int counttotalvote(const char* s, int divisionchoosen);
static int callvotedata(void* NotUsed, int argc, char** argv, char** azColName);
static int maxvotecount(const char* s, int divisionchoosen);
static int minvotecount(const char* s, int divisionchoosen);
static int callvoterdata(void* NotUsed, int argc, char** argv, char** azColName);
static int selectvoterdata(const char* s);
static string checkvoterstatus(const char* s, string voterid);
static string checkvoterdivision(const char* s, string voterid);
static int candidateinfovoteselection(const char* s, string division);
vector <string> storevotereid(const char* s);
vector <string> usedcandidateidbasedondivision(const char* s, string division);
static int candidatevotebasedonid(const char* s, string candidateid);
static int votecountupdate(const char* s, string votecount, string candidateid);
```

Figure 18: Program Description for Voter Program



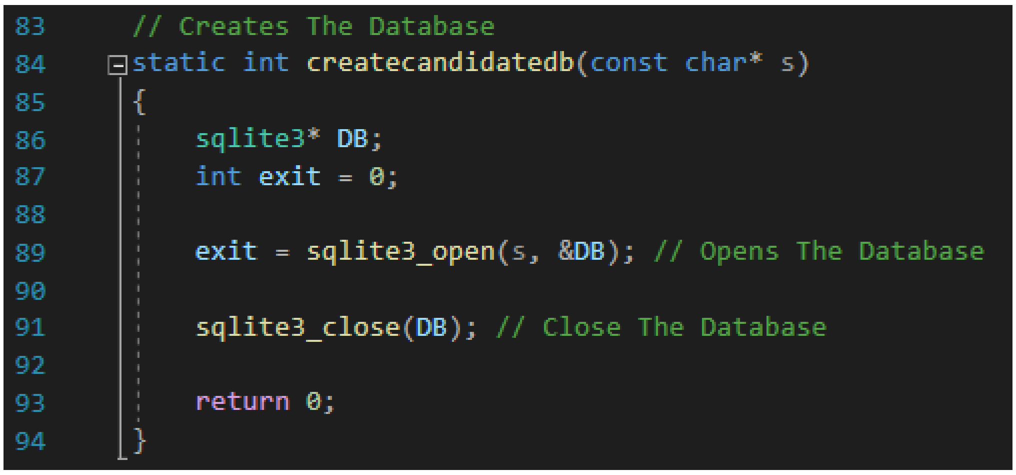
```cpp
// View Candidate Information
void viewcandidateinfo()
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    if (divisionsizecheck(division) == 0)
    {
        cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
    }
    else
    {
        createcandidatedb(division); // Creates The Database If It Is Not Created
        createcandidatetable(division); // Creates The Candidate Table
        selectcandidatedata(division); // Calls selectcandidatedata Function To Select The Candidate Information From Database And Display It
    }
}
```

Figure 19: viewcandidaeinfo function



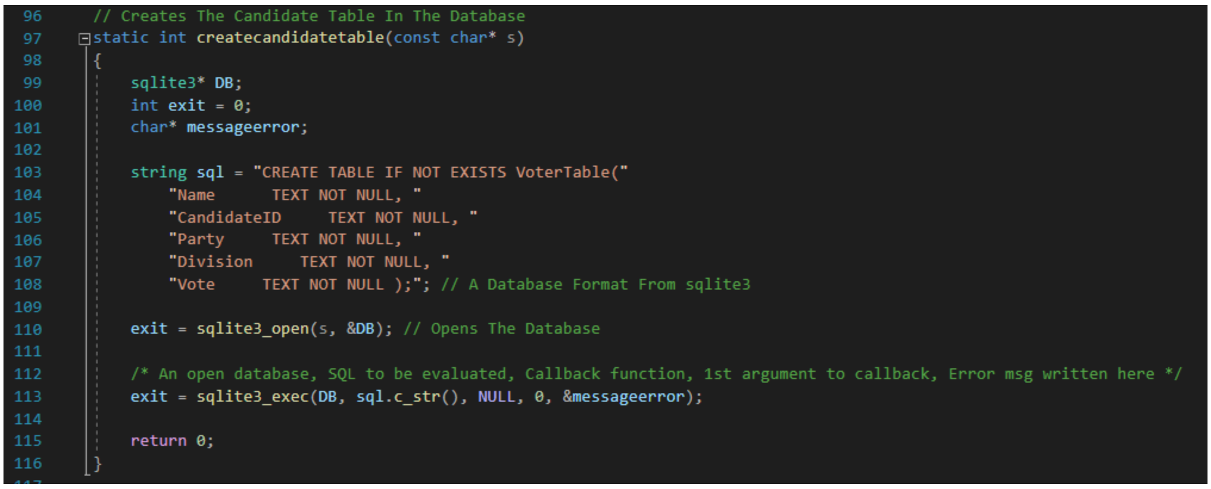
```cpp
// Voter's Database
void VoterDatabase()
{
    const char* voter = R"(C:\\StoreData\\Voter.db)"; // Calls The Database

    createvoterdb(voter);
    createvotertable(voter);
    insertvoterdata(voter);
}
```

Figure 20: VoterDatabase function



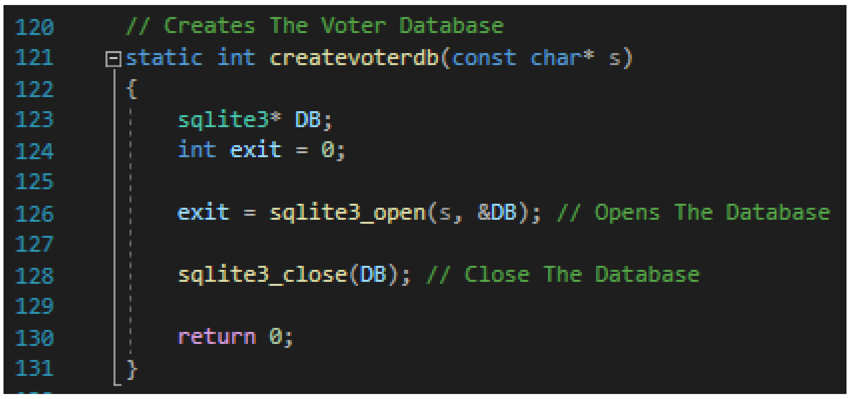
```cpp
// Creates The Database
static int createcandidatedb(const char* s)
{
    sqlite3* DB;
    int exit = 0;

    exit = sqlite3_open(s, &DB); // Opens The Database

    sqlite3_close(DB); // Close The Database

    return 0;
}
```

Figure 21: createcandidatedb function



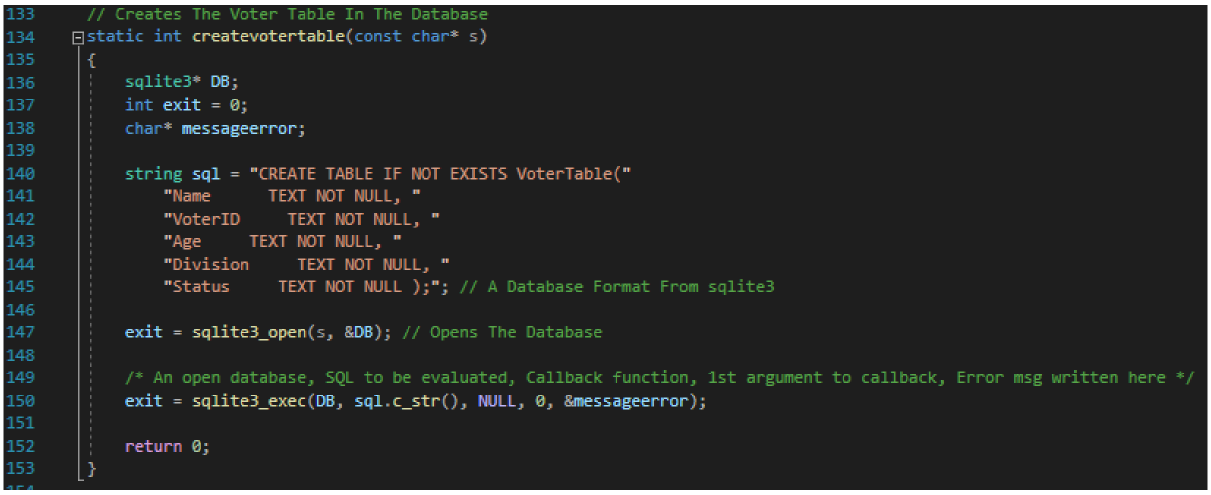
```cpp
// Creates The Candidate Table In The Database
static int createcandidatetable(const char* s)
{
    sqlite3* DB;
    int exit = 0;
    char* messageerror;

    string sql = "CREATE TABLE IF NOT EXISTS VoterTable("
        "Name      TEXT NOT NULL, "
        "CandidateID     TEXT NOT NULL, "
        "Party     TEXT NOT NULL, "
        "Division     TEXT NOT NULL, "
        "Vote     TEXT NOT NULL );"; // A Database Format From sqlite3

    exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageerror);

    return 0;
}
```

Figure 22: createcandidatetable function



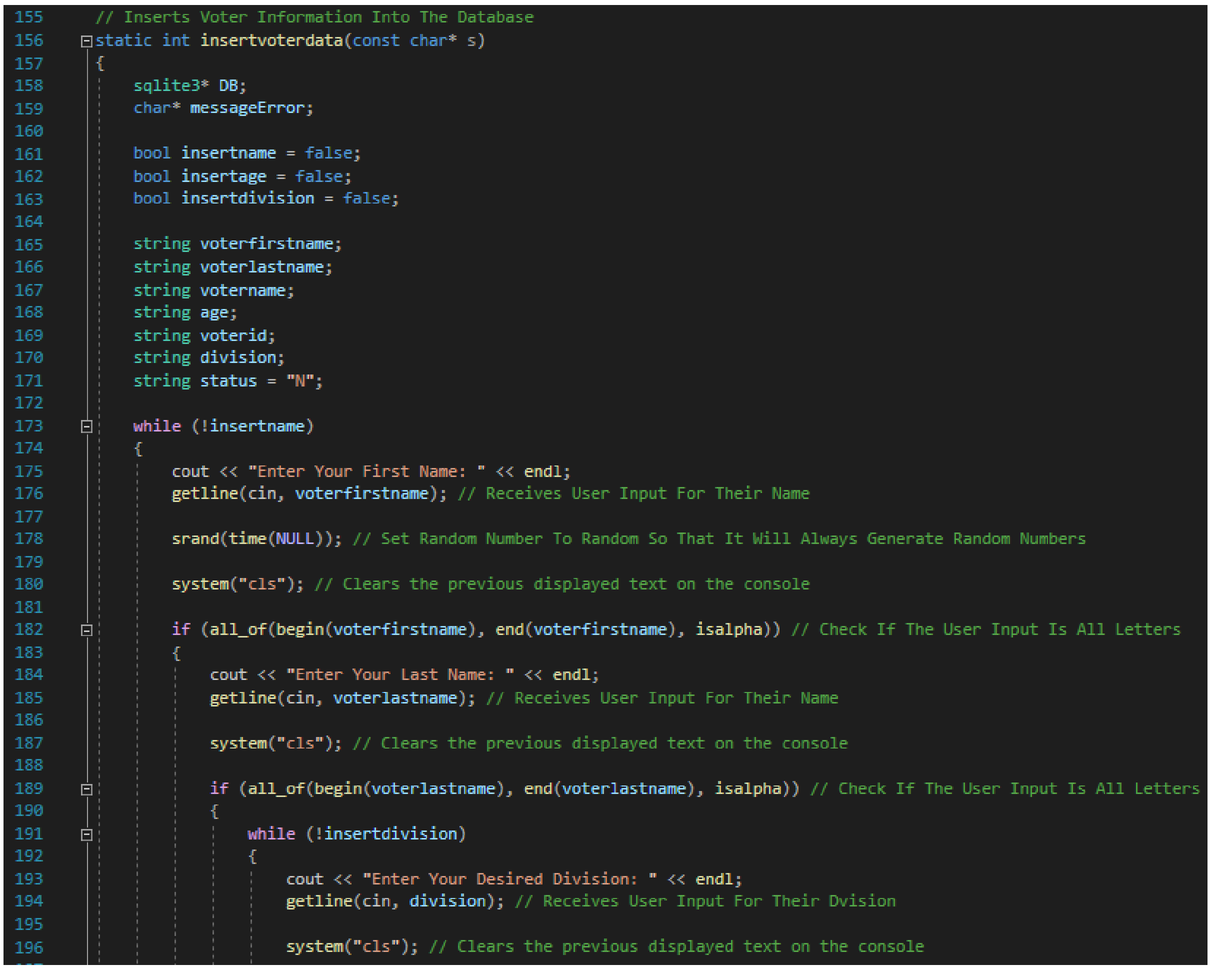
```cpp
// Creates The Voter Database
static int createvoterdb(const char* s)
{
    sqlite3* DB;
    int exit = 0;

    exit = sqlite3_open(s, &DB); // Opens The Database

    sqlite3_close(DB); // Close The Database

    return 0;
}
```

Figure 23: createvoterdb function

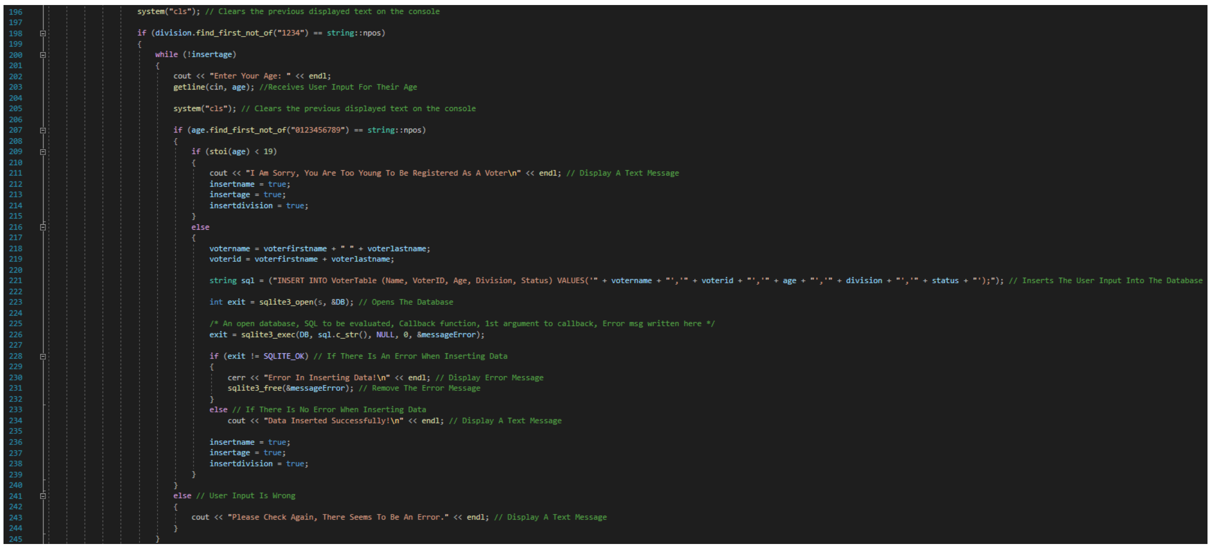


```cpp
// Creates The Voter Table In The Database
static int createvotertable(const char* s)
{
    sqlite3* DB;
    int exit = 0;
    char* messageerror;

    string sql = "CREATE TABLE IF NOT EXISTS VoterTable("
        "Name      TEXT NOT NULL, "
        "VoterID     TEXT NOT NULL, "
        "Age     TEXT NOT NULL, "
        "Division     TEXT NOT NULL, "
        "Status     TEXT NOT NULL );"; // A Database Format From sqlite3

    exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageerror);

    return 0;
}
```

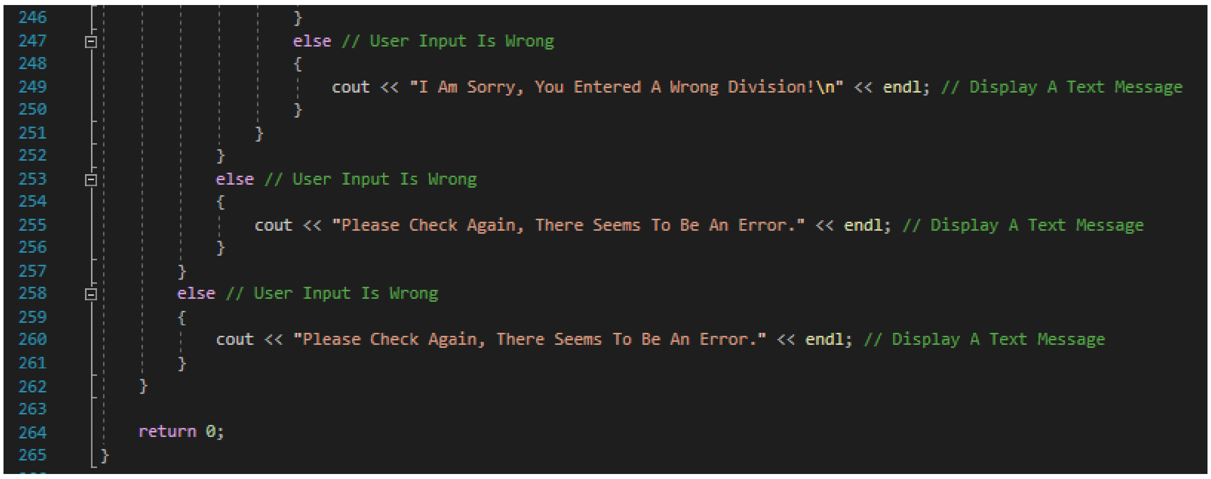
Figure 24:createvotertable function



```cpp
// Inserts Voter Information Into The Database
static int insertvoterdata(const char* s)
{
    sqlite3* DB;
    char* messageError;

    bool insertname = false;
    bool insertage = false;
    bool insertdivision = false;

    string voterfirstname;
    string voterlastname;
    string votername;
    string age;
    string voterid;
    string division;
    string status = "N";

    while (!insertname)
    {
        cout << "Enter Your First Name: " << endl;
        getline(cin, voterfirstname); // Receives User Input For Their Name

        srand(time(NULL)); // Set Random Number To Random So That It Will Always Generate Random Numbers

        system("cls"); // Clears the previous displayed text on the console

        if (all_of(begin(voterfirstname), end(voterfirstname), isalpha)) // Check If The User Input Is All Letters
        {
            cout << "Enter Your Last Name: " << endl;
            getline(cin, voterlastname); // Receives User Input For Their Name

            system("cls"); // Clears the previous displayed text on the console

            if (all_of(begin(voterlastname), end(voterlastname), isalpha)) // Check If The User Input Is All Letters
            {
                while (!insertdivision)
                {
                    cout << "Enter Your Desired Division: " << endl;
                    getline(cin, division); // Receives User Input For Their Dvision

                    system("cls"); // Clears the previous displayed text on the console
```

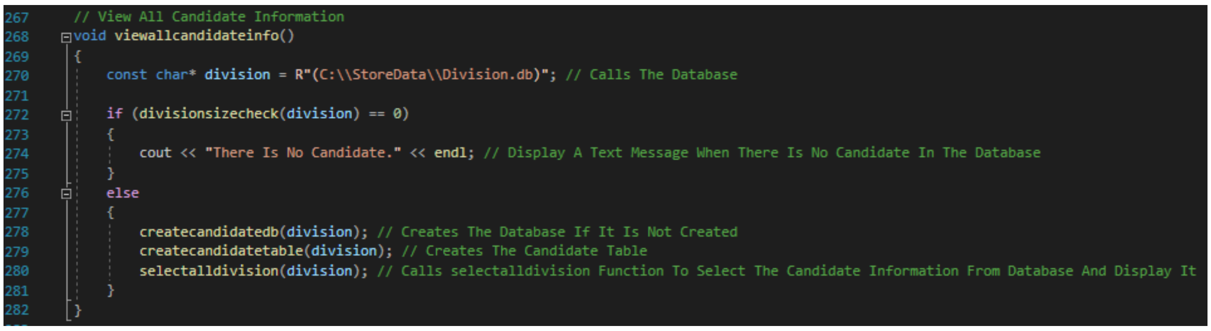


```cpp
                    system("cls"); // Clears the previous displayed text on the console

                    if (division.find_first_not_of("1234") == string::npos)
                    {
                        while (!insertage)
                        {
                            cout << "Enter Your Age: " << endl;
                            getline(cin, age); //Receives User Input For Their Age

                            system("cls"); // Clears the previous displayed text on the console

                            if (age.find_first_not_of("0123456789") == string::npos)
                            {
                                if (stoi(age) < 19)
                                {
                                    cout << "I Am Sorry, You Are Too Young To Be Registered As A Voter\n" << endl; // Display A Text Message
                                    insertname = true;
                                    insertage = true;
                                    insertdivision = true;
                                }
                                else
                                {
                                    votername = voterfirstname + " " + voterlastname;
                                    voterid = voterfirstname + voterlastname;

                                    string sql = ("INSERT INTO VoterTable (Name, VoterID, Age, Division, Status) VALUES('" + votername + "','" + voterid + "','" + age + "','" + division + "','" + status + "');"); // Inserts The User Input Into The Database

                                    int exit = sqlite3_open(s, &DB); // Opens The Database

                                    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
                                    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);

                                    if (exit != SQLITE_OK) // If There Is An Error When Inserting Data
                                    {
                                        cerr << "Error In Inserting Data!\n" << endl; // Display Error Message
                                        sqlite3_free(&messageError); // Remove The Error Message
                                    }
                                    else // If There Is No Error When Inserting Data
                                        cout << "Data Inserted Successfully!\n" << endl; // Display A Text Message

                                    insertname = true;
                                    insertage = true;
                                    insertdivision = true;
                                }
                            }
                            else // User Input Is Wrong
                            {
                                cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
                            }
                        }
```



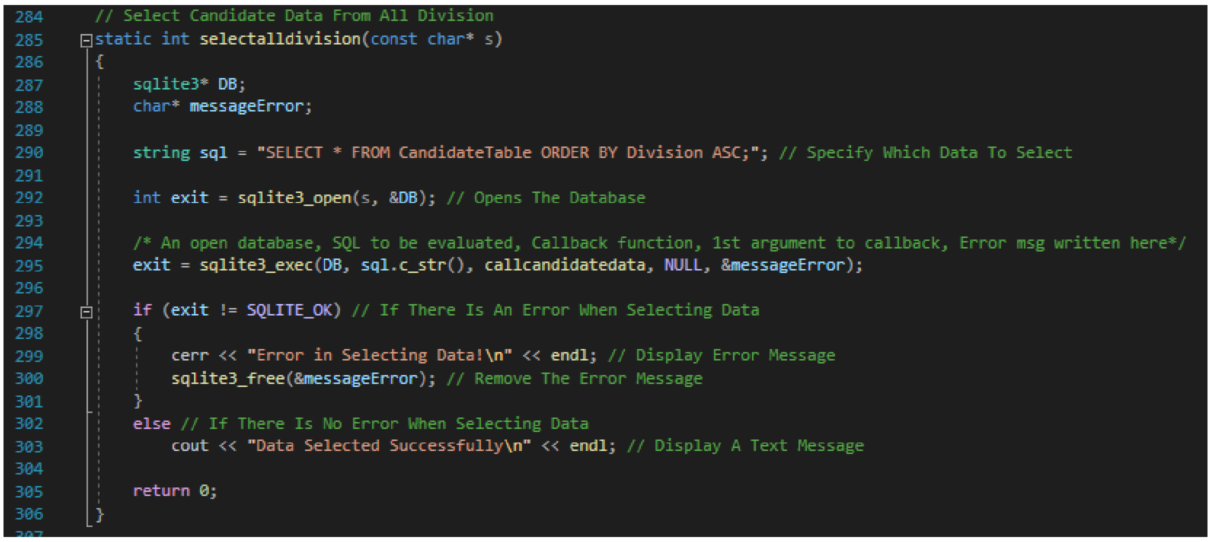
```
                    }
                    else // User Input Is Wrong
                    {
                        cout << "I Am Sorry, You Entered A Wrong Division!\n" << endl; // Display A Text Message
                    }
                }
            }
            else // User Input Is Wrong
            {
                cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
            }
        }
        else // User Input Is Wrong
        {
            cout << "Please Check Again, There Seems To Be An Error." << endl; // Display A Text Message
        }
    }

    return 0;
}
```

Figure 25: insertvoterdata function



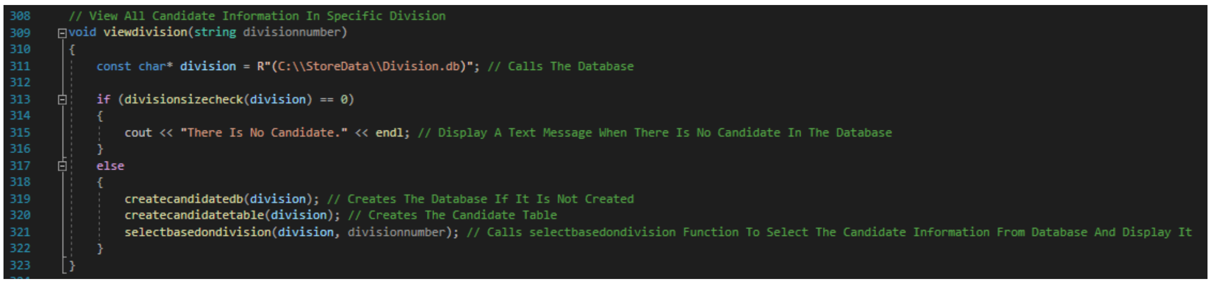
```
// View All Candidate Information
void viewallcandidateinfo()
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    if (divisionsizecheck(division) == 0)
    {
        cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
    }
    else
    {
        createcandidatedb(division); // Creates The Database If It Is Not Created
        createcandidatetable(division); // Creates The Candidate Table
        selectalldivision(division); // Calls selectalldivision Function To Select The Candidate Information From Database And Display It
    }
}
```

Figure 26: viewallcandidateinfo function



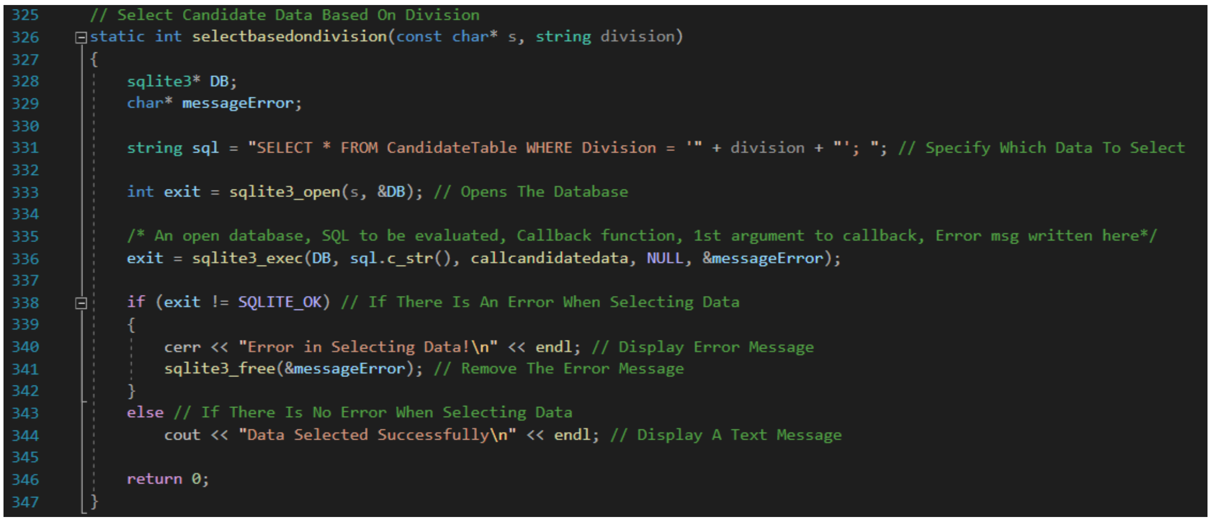
```
// Select Candidate Data From All Division
static int selectalldivision(const char* s)
{
    sqlite3* DB;
    char* messageError;

    string sql = "SELECT * FROM CandidateTable ORDER BY Division ASC;"; // Specify Which Data To Select

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

Figure 27: selectalldivision function



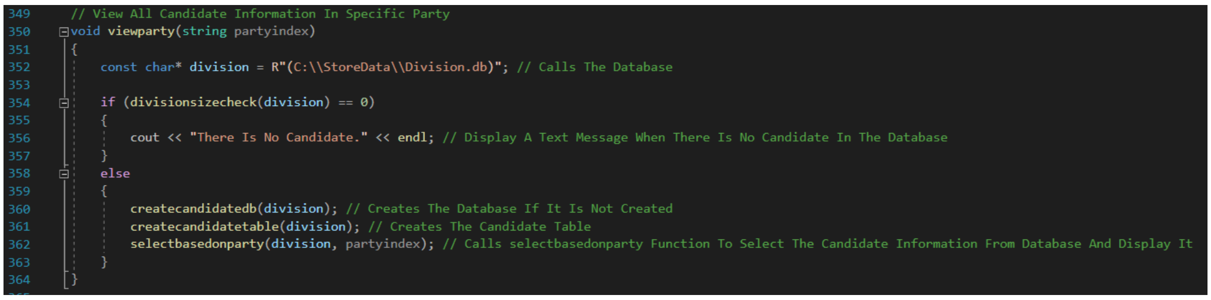
```cpp
// View All Candidate Information In Specific Division
void viewdivision(string divisionnumber)
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    if (divisionsizecheck(division) == 0)
    {
        cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
    }
    else
    {
        createcandidatedb(division); // Creates The Database If It Is Not Created
        createcandidatetable(division); // Creates The Candidate Table
        selectbasedondivision(division, divisionnumber); // Calls selectbasedondivision Function To Select The Candidate Information From Database And Display It
    }
}
```

Figure 28: viewdivision function



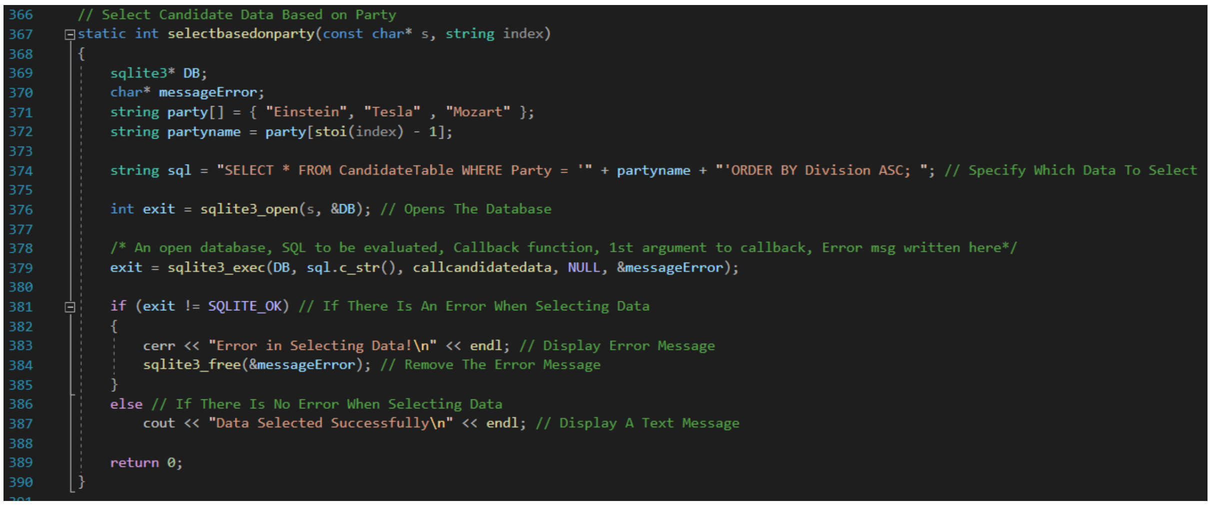
```cpp
// Select Candidate Data Based On Division
static int selectbasedondivision(const char* s, string division)
{
    sqlite3* DB;
    char* messageError;

    string sql = "SELECT * FROM CandidateTable WHERE Division = '" + division + "'; "; // Specify Which Data To Select

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

Figure 29: selectbasedondivision function



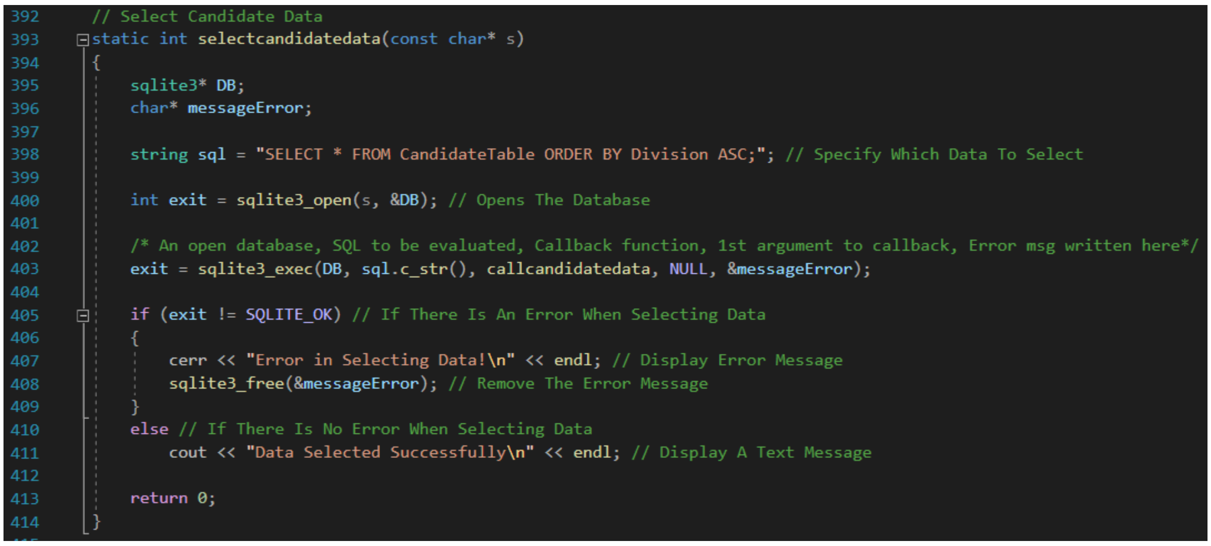
```cpp
// View All Candidate Information In Specific Party
void viewparty(string partyindex)
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    if (divisionsizecheck(division) == 0)
    {
        cout << "There Is No Candidate." << endl; // Display A Text Message When There Is No Candidate In The Database
    }
    else
    {
        createcandidatedb(division); // Creates The Database If It Is Not Created
        createcandidatetable(division); // Creates The Candidate Table
        selectbasedonparty(division, partyindex); // Calls selectbasedonparty Function To Select The Candidate Information From Database And Display It
    }
}
```

Figure 30: viewparty function



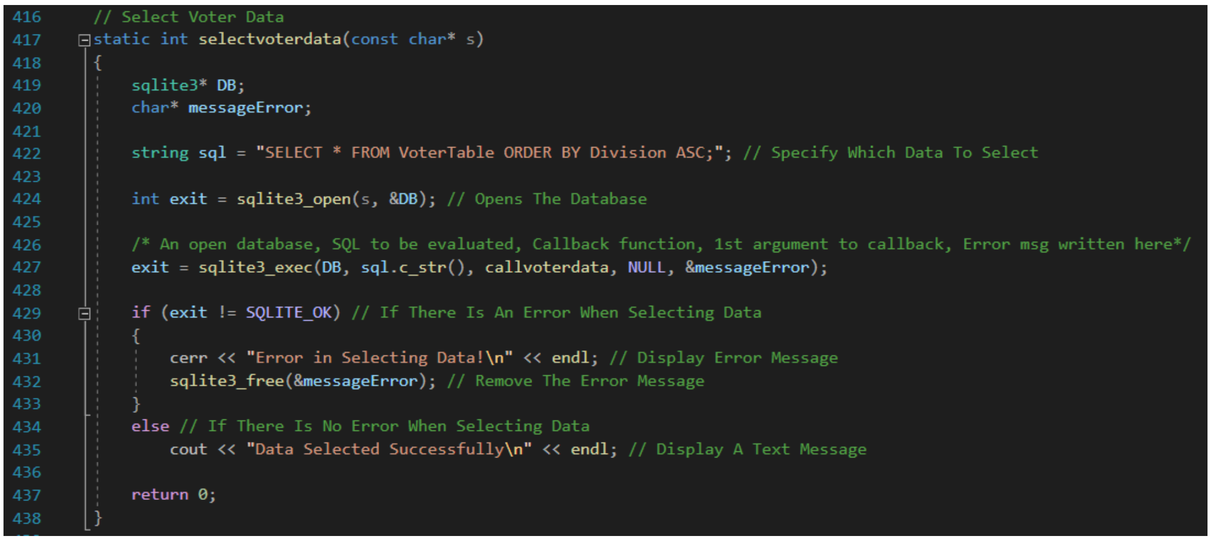
```cpp
// Select Candidate Data Based on Party
static int selectbasedonparty(const char* s, string index)
{
    sqlite3* DB;
    char* messageError;
    string party[] = { "Einstein", "Tesla" , "Mozart" };
    string partyname = party[stoi(index) - 1];

    string sql = "SELECT * FROM CandidateTable WHERE Party = '" + partyname + "'ORDER BY Division ASC; "; // Specify Which Data To Select

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

Figure 31: selectbasedonparty function



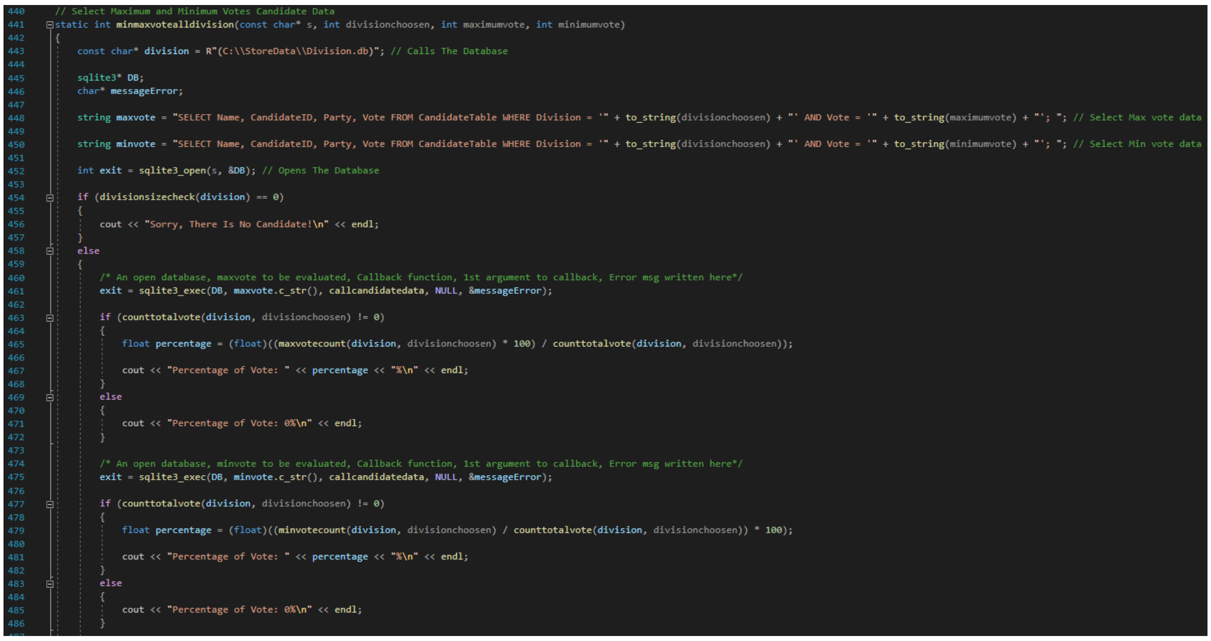
```cpp
// Select Candidate Data
static int selectcandidatedata(const char* s)
{
    sqlite3* DB;
    char* messageError;

    string sql = "SELECT * FROM CandidateTable ORDER BY Division ASC;"; // Specify Which Data To Select

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

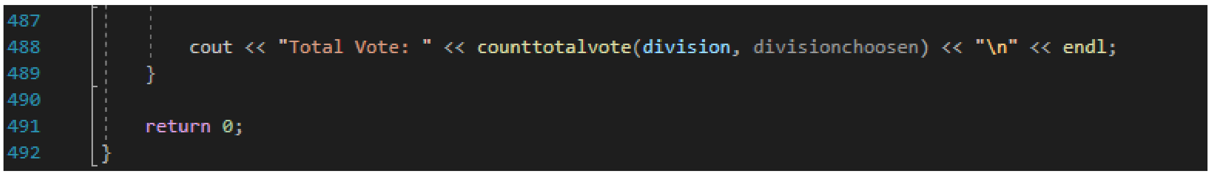
Figure 32: selectcandidatadata function



```cpp
// Select Voter Data
static int selectvoterdata(const char* s)
{
    sqlite3* DB;
    char* messageError;

    string sql = "SELECT * FROM VoterTable ORDER BY Division ASC;"; // Specify Which Data To Select

    int exit = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callvoterdata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

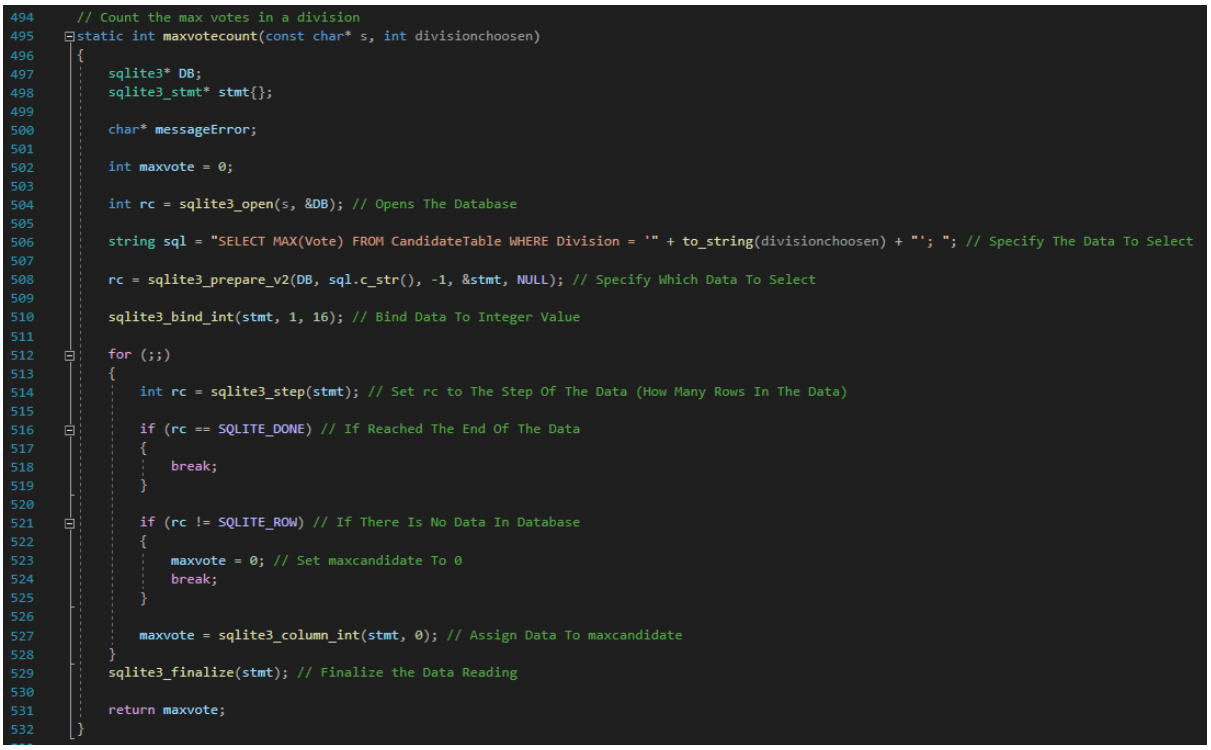
Figure 33: selectvoterdata function





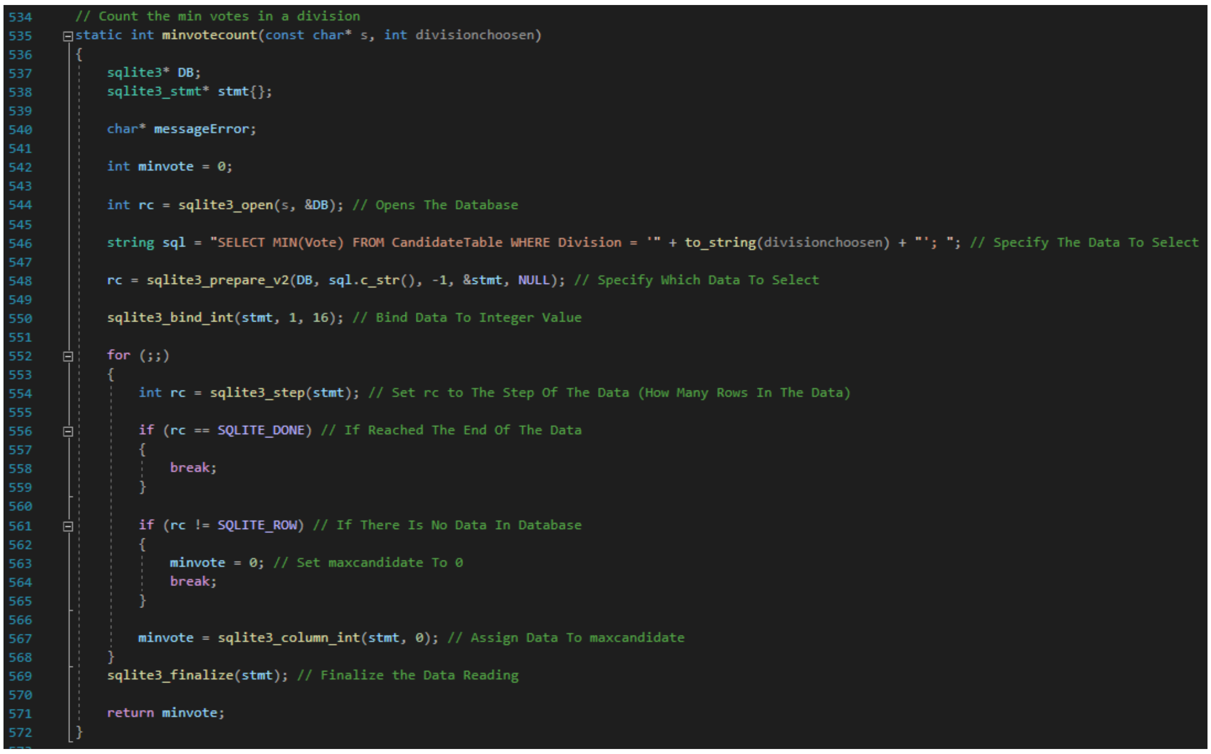
```cpp
// Select Maximum and Minimum Votes Candidate Data
static int minmaxvotealldivision(const char* s, int divisionchoosen, int maximumvote, int minimumvote)
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    sqlite3* DB;
    char* messageError;

    string maxvote = "SELECT Name, CandidateID, Party, Vote FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "' AND Vote = '" + to_string(maximumvote) + "'; "; // Select Max vote data

    string minvote = "SELECT Name, CandidateID, Party, Vote FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "' AND Vote = '" + to_string(minimumvote) + "'; "; // Select Min vote data

    int exit = sqlite3_open(s, &DB); // Opens The Database

    if (divisionsizecheck(division) == 0)
    {
        cout << "Sorry, There Is No Candidate!\n" << endl;
    }
    else
    {
        /* An open database, maxvote to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
        exit = sqlite3_exec(DB, maxvote.c_str(), callcandidatedata, NULL, &messageError);

        if (counttotalvote(division, divisionchoosen) != 0)
        {
            float percentage = (float)((maxvotecount(division, divisionchoosen) * 100) / counttotalvote(division, divisionchoosen));

            cout << "Percentage of Vote: " << percentage << "%\n" << endl;
        }
        else
        {
            cout << "Percentage of Vote: 0%\n" << endl;
        }

        /* An open database, minvote to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
        exit = sqlite3_exec(DB, minvote.c_str(), callcandidatedata, NULL, &messageError);

        if (counttotalvote(division, divisionchoosen) != 0)
        {
            float percentage = (float)((minvotecount(division, divisionchoosen) / counttotalvote(division, divisionchoosen)) * 100);

            cout << "Percentage of Vote: " << percentage << "%\n" << endl;
        }
        else
        {
            cout << "Percentage of Vote: 0%\n" << endl;
        }

        cout << "Total Vote: " << counttotalvote(division, divisionchoosen) << "\n" << endl;
    }

    return 0;
}
```

Figure 34: minmaxvotealldivision function



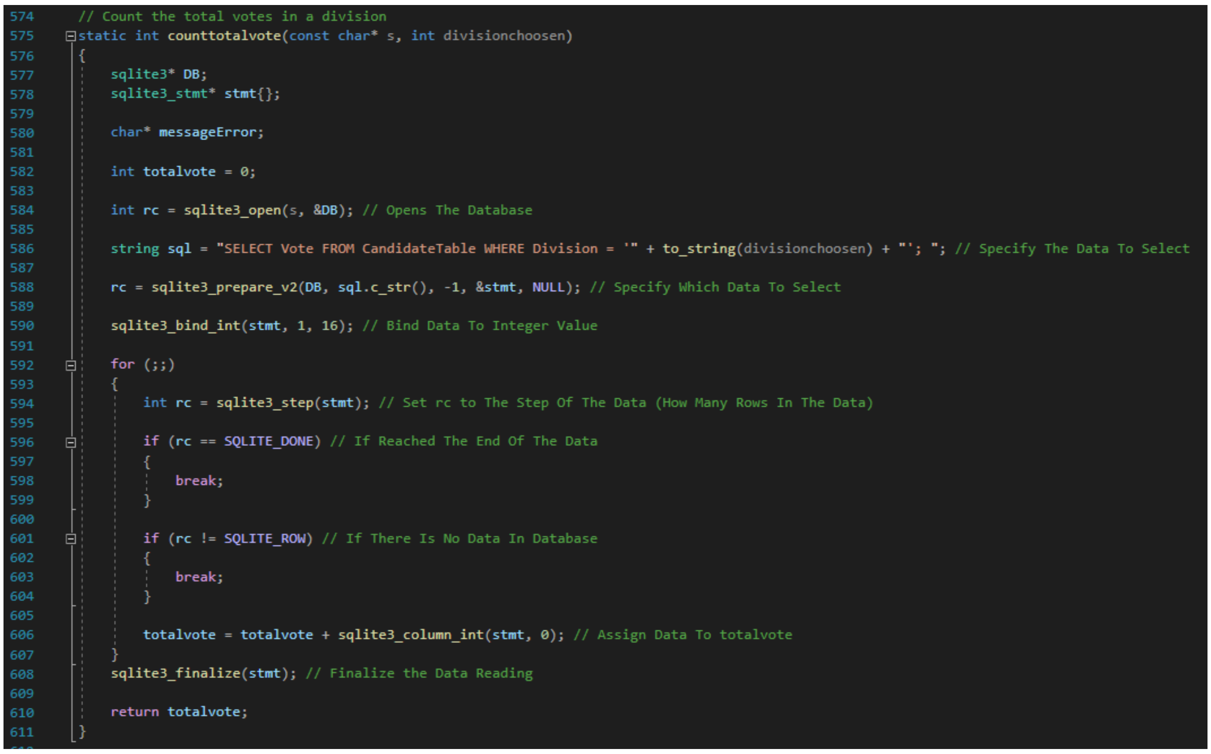
```cpp
// Count the max votes in a division
static int maxvotecount(const char* s, int divisionchoosen)
{
    sqlite3* DB;
    sqlite3_stmt* stmt{};

    char* messageError;

    int maxvote = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT MAX(Vote) FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            maxvote = 0; // Set maxcandidate To 0
            break;
        }

        maxvote = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return maxvote;
}
```

Figure 35: maxvotecount function



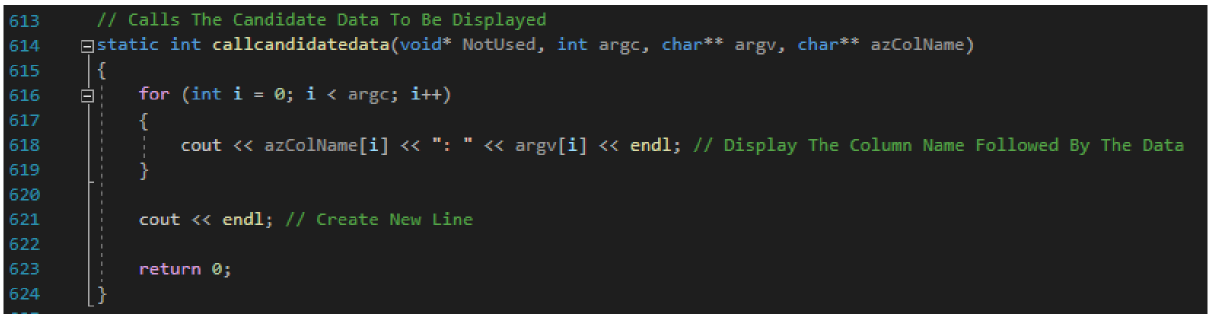
```
// Count the min votes in a division
static int minvotecount(const char* s, int divisionchoosen)
{
    sqlite3* DB;
    sqlite3_stmt* stmt{};

    char* messageError;

    int minvote = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT MIN(Vote) FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            minvote = 0; // Set maxcandidate To 0
            break;
        }

        minvote = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return minvote;
}
```

Figure 36: minvotecount function



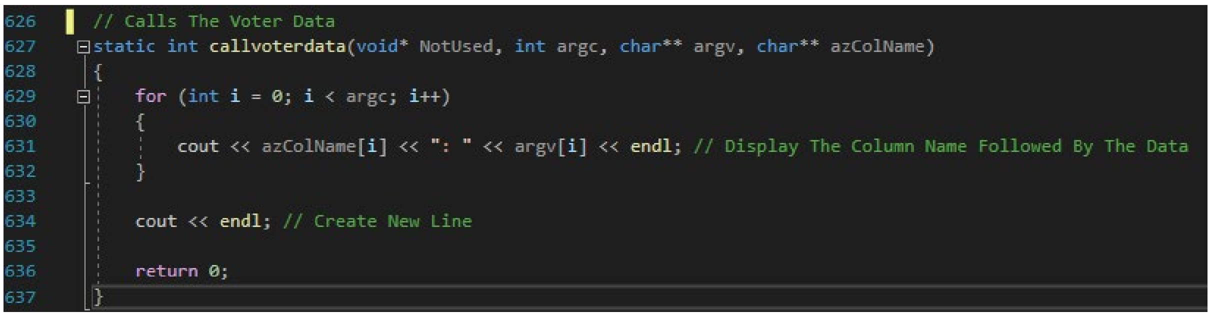
```
// Count the total votes in a division
static int counttotalvote(const char* s, int divisionchoosen)
{
    sqlite3* DB;
    sqlite3_stmt* stmt{};

    char* messageError;

    int totalvote = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Vote FROM CandidateTable WHERE Division = '" + to_string(divisionchoosen) + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            break;
        }

        totalvote = totalvote + sqlite3_column_int(stmt, 0); // Assign Data To totalvote
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return totalvote;
}
```

Figure 37: counttotalvote function



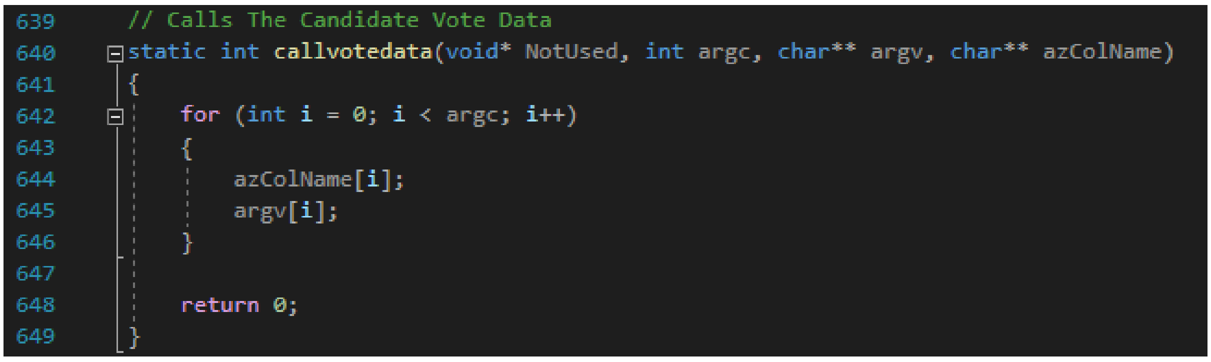
```cpp
// Calls The Candidate Data To Be Displayed
static int callcandidatedata(void* NotUsed, int argc, char** argv, char** azColName)
{
    for (int i = 0; i < argc; i++)
    {
        cout << azColName[i] << ": " << argv[i] << endl; // Display The Column Name Followed By The Data
    }

    cout << endl; // Create New Line

    return 0;
}
```

Figure 38: callcandidatedata function



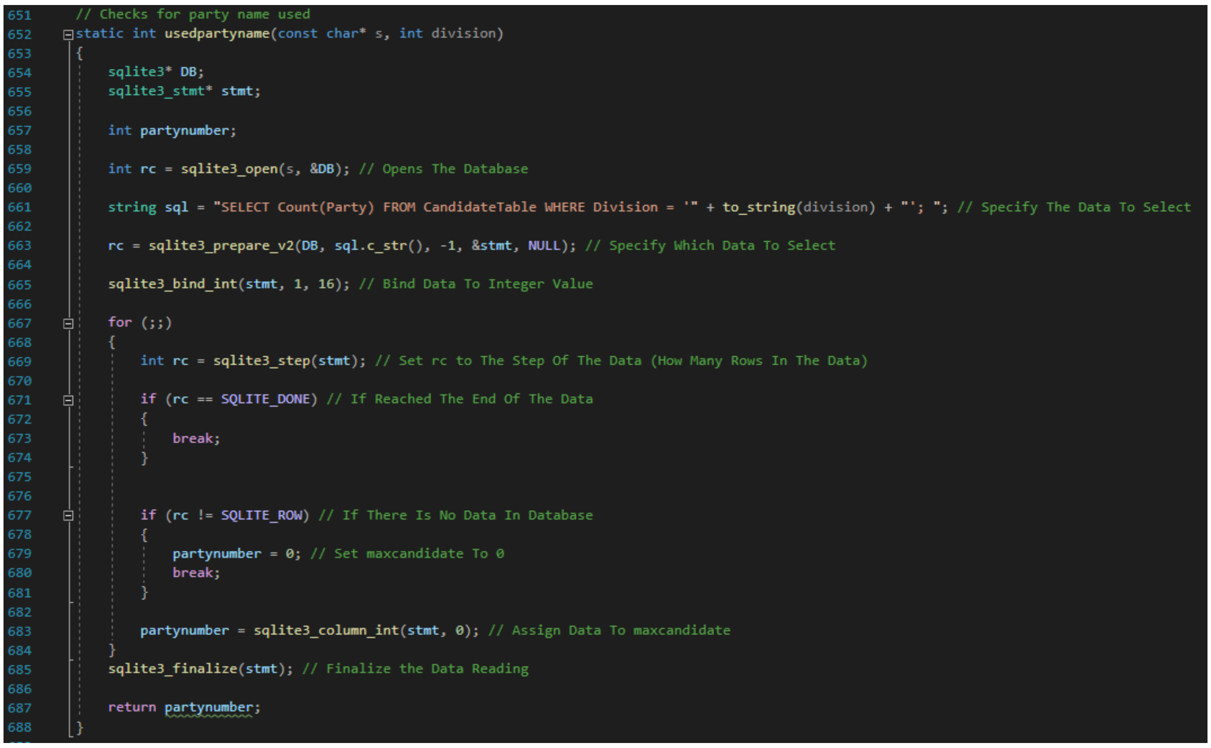
```cpp
// Calls The Voter Data
static int callvoterdata(void* NotUsed, int argc, char** argv, char** azColName)
{
    for (int i = 0; i < argc; i++)
    {
        cout << azColName[i] << ": " << argv[i] << endl; // Display The Column Name Followed By The Data
    }

    cout << endl; // Create New Line

    return 0;
}
```

Figure 39: callvoterdata function



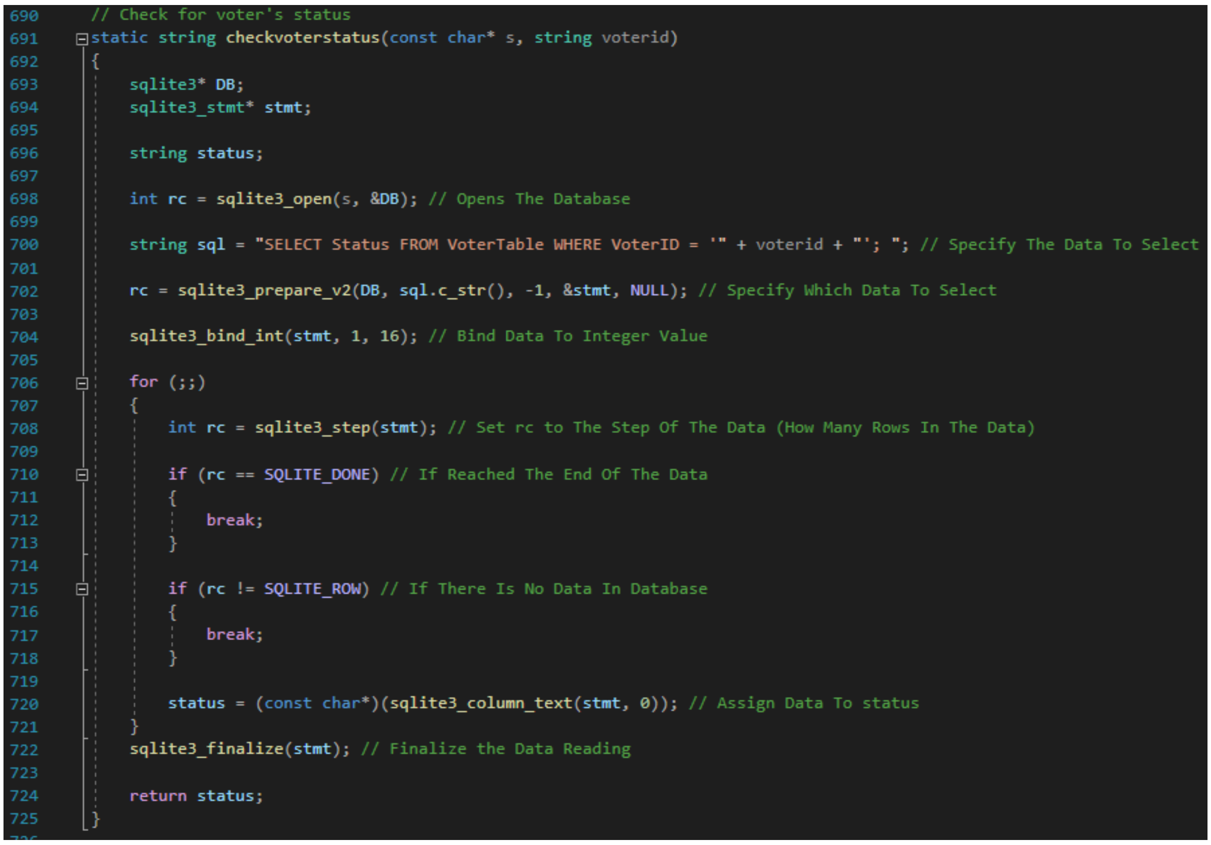
```cpp
// Calls The Candidate Vote Data
static int callvotedata(void* NotUsed, int argc, char** argv, char** azColName)
{
    for (int i = 0; i < argc; i++)
    {
        azColName[i];
        argv[i];
    }

    return 0;
}
```

Figure 40: callvotedata function



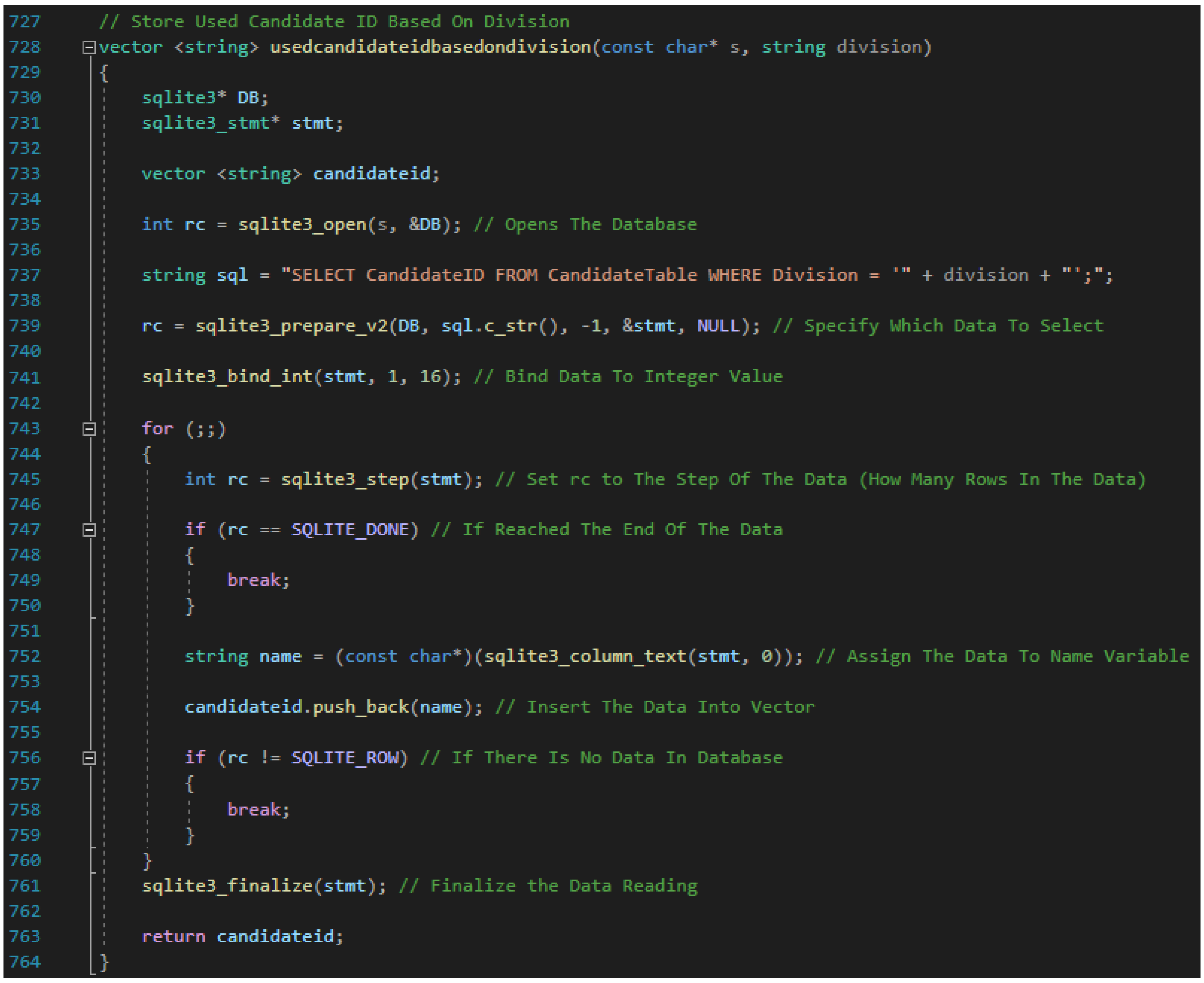
```
// Checks for party name used
static int usedpartyname(const char* s, int division)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    int partynumber;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }


        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            partynumber = 0; // Set maxcandidate To 0
            break;
        }

        partynumber = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return partynumber;
}
```

Figure 41: usedpartyname function



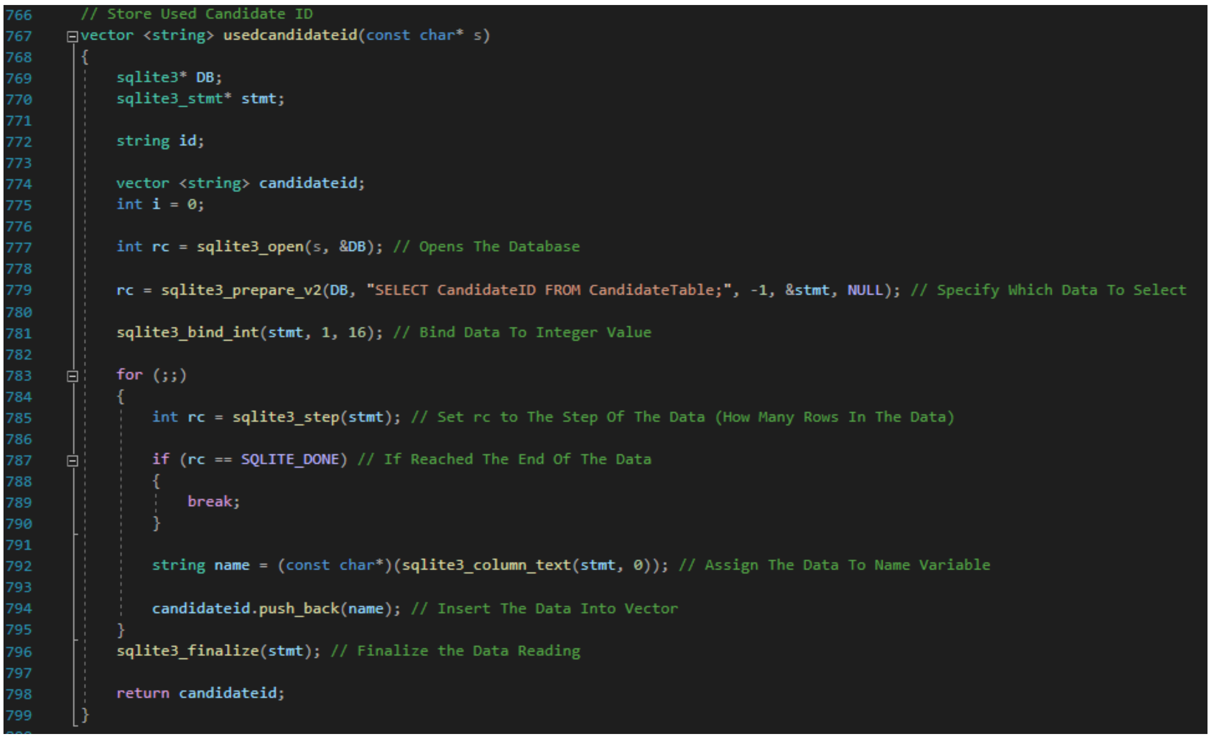
```
// Check for voter's status
static string checkvoterstatus(const char* s, string voterid)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    string status;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Status FROM VoterTable WHERE VoterID = '" + voterid + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            break;
        }

        status = (const char*)(sqlite3_column_text(stmt, 0)); // Assign Data To status
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return status;
}
```

Figure 42: checkvoterstatus function



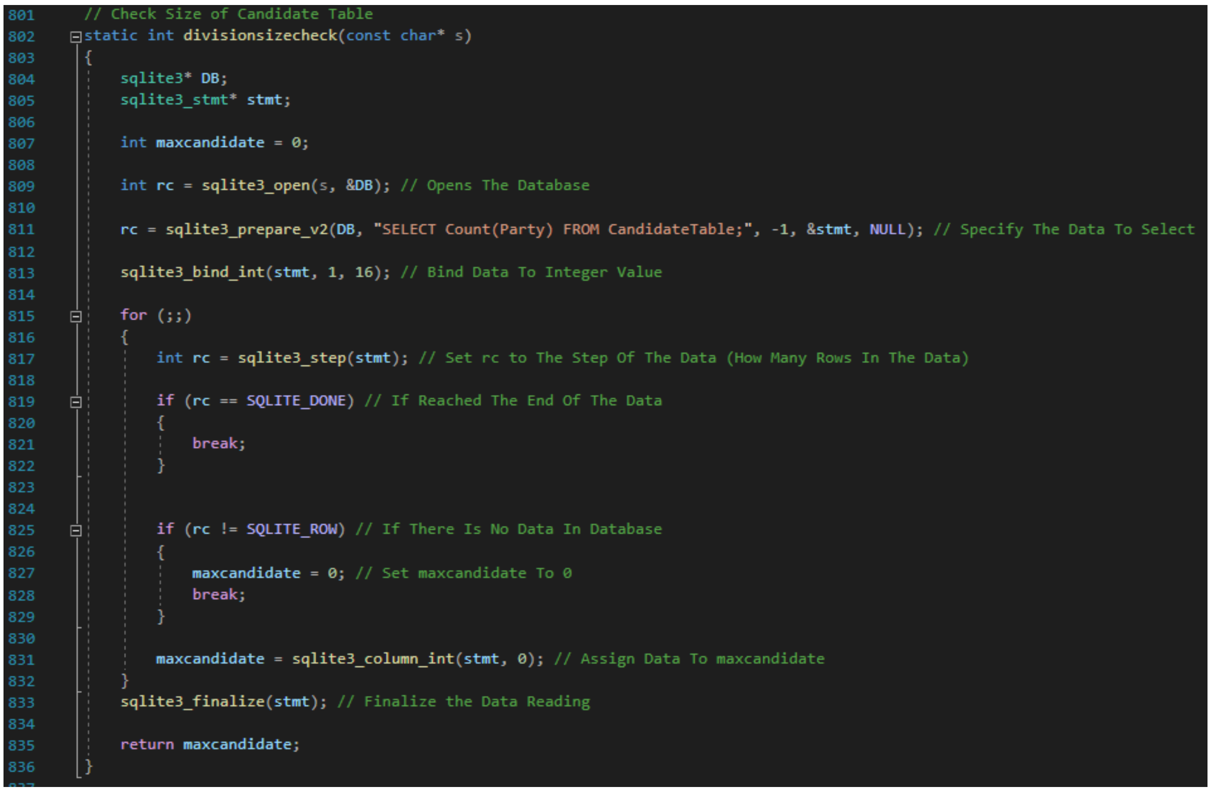
```cpp
    // Store Used Candidate ID Based On Division
vector <string> usedcandidateidbasedondivision(const char* s, string division)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    vector <string> candidateid;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT CandidateID FROM CandidateTable WHERE Division = '" + division + "';";

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        string name = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable

        candidateid.push_back(name); // Insert The Data Into Vector

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            break;
        }
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return candidateid;
}
```

Figure 43: usedcandidateidbasedondivision function



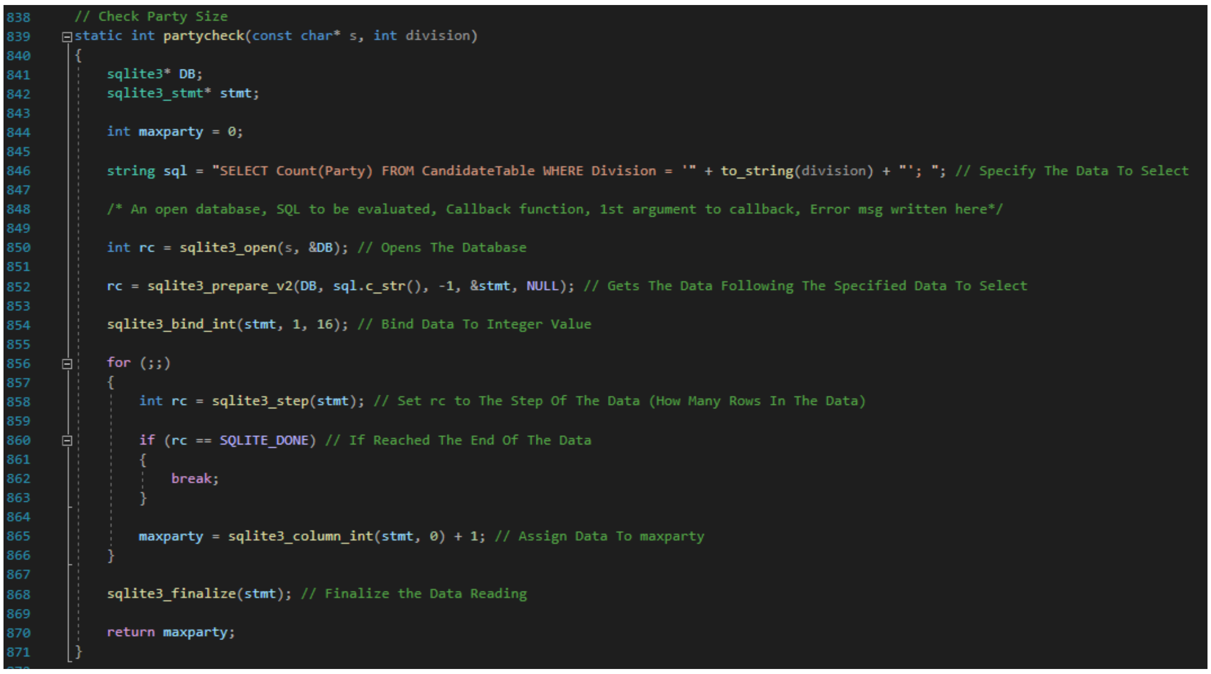
```cpp
  // Store Used Candidate ID
vector <string> usedcandidateid(const char* s)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    string id;

    vector <string> candidateid;
    int i = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, "SELECT CandidateID FROM CandidateTable;", -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        string name = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable

        candidateid.push_back(name); // Insert The Data Into Vector
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return candidateid;
}
```

Figure 44: usedcandidateid function



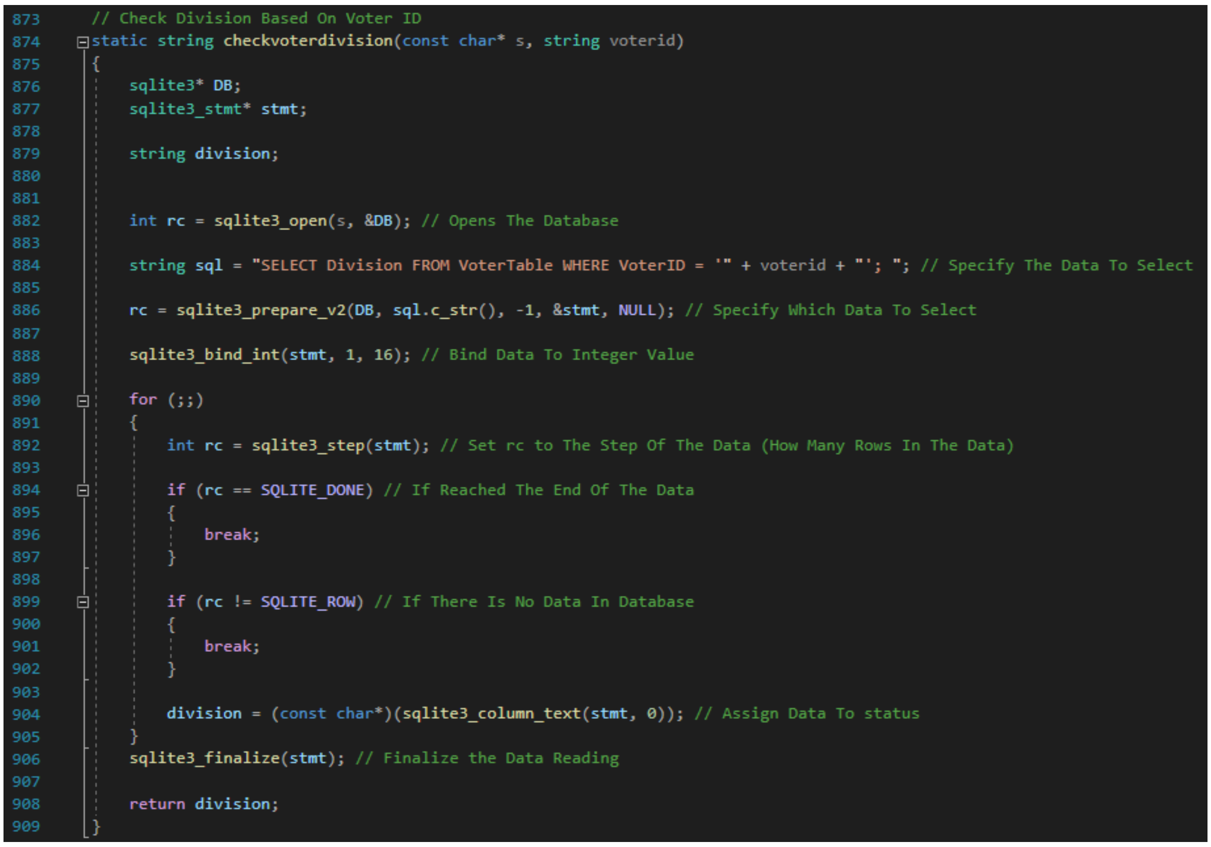
```cpp
// Check Size of Candidate Table
static int divisionsizecheck(const char* s)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    int maxcandidate = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, "SELECT Count(Party) FROM CandidateTable;", -1, &stmt, NULL); // Specify The Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }


        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            maxcandidate = 0; // Set maxcandidate To 0
            break;
        }

        maxcandidate = sqlite3_column_int(stmt, 0); // Assign Data To maxcandidate
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return maxcandidate;
}
```

Figure 45: divisionsizecheck function



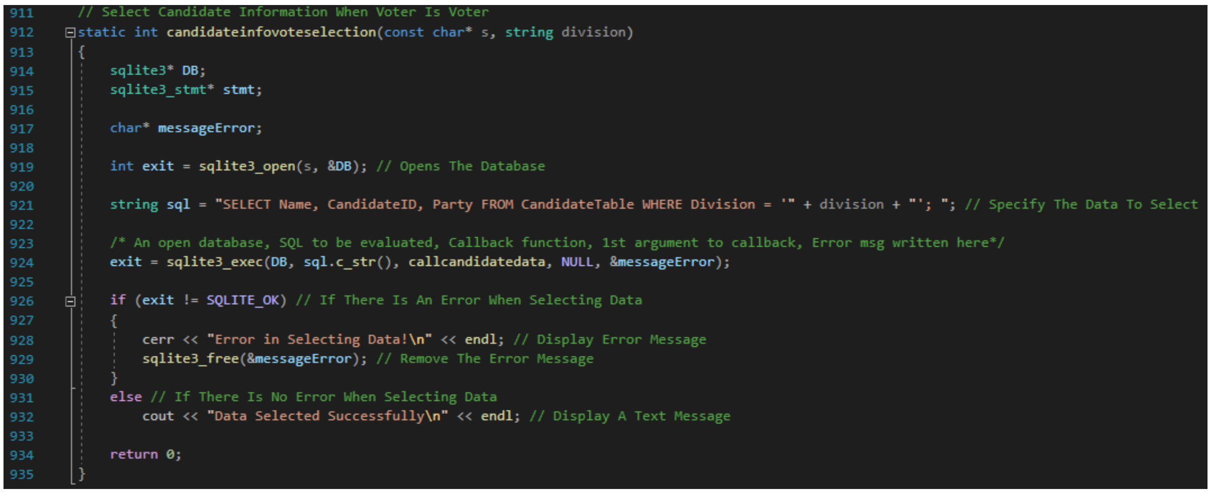
```cpp
// Check Party Size
static int partycheck(const char* s, int division)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    int maxparty = 0;

    string sql = "SELECT Count(Party) FROM CandidateTable WHERE Division = '" + to_string(division) + "'; "; // Specify The Data To Select

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Gets The Data Following The Specified Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        maxparty = sqlite3_column_int(stmt, 0) + 1; // Assign Data To maxparty
    }

    sqlite3_finalize(stmt); // Finalize the Data Reading

    return maxparty;
}
```

Figure 46: partycheck function



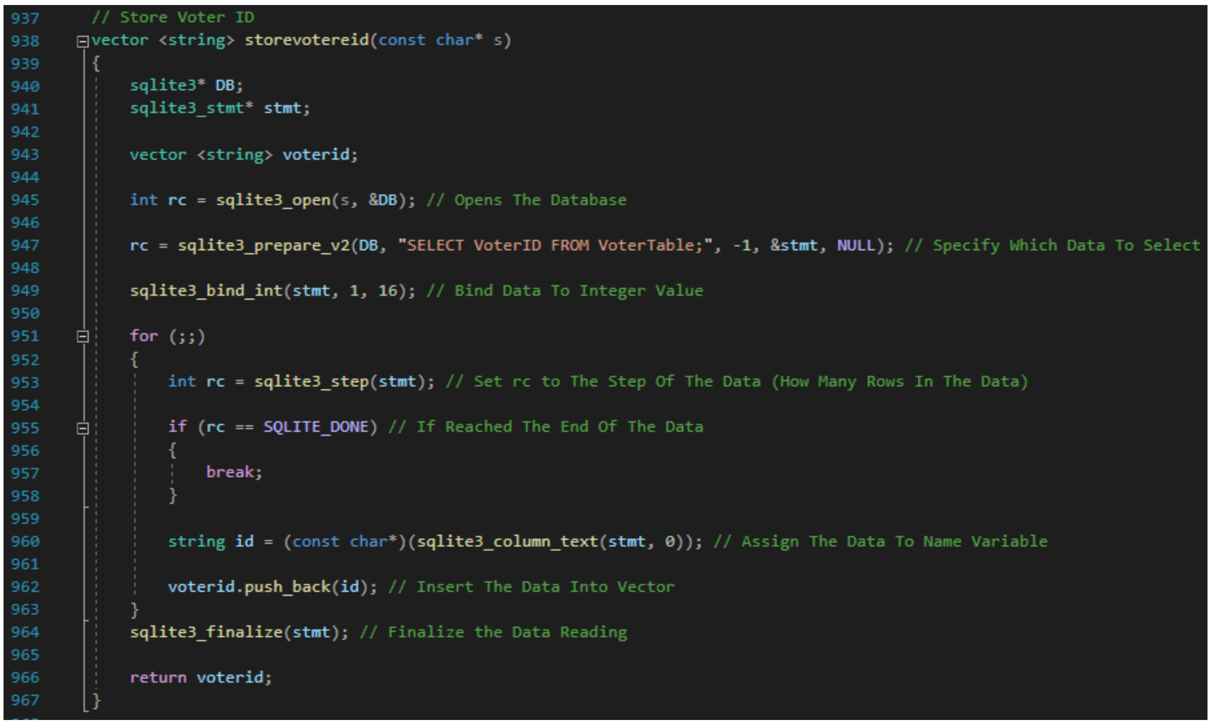
```cpp
// Check Division Based On Voter ID
static string checkvoterdivision(const char* s, string voterid)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    string division;


    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Division FROM VoterTable WHERE VoterID = '" + voterid + "'; "; // Specify The Data To Select

    rc = sqlite3_prepare_v2(DB, sql.c_str(), -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            break;
        }

        division = (const char*)(sqlite3_column_text(stmt, 0)); // Assign Data To status
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return division;
}
```

Figure 47: checkvoterdivision function



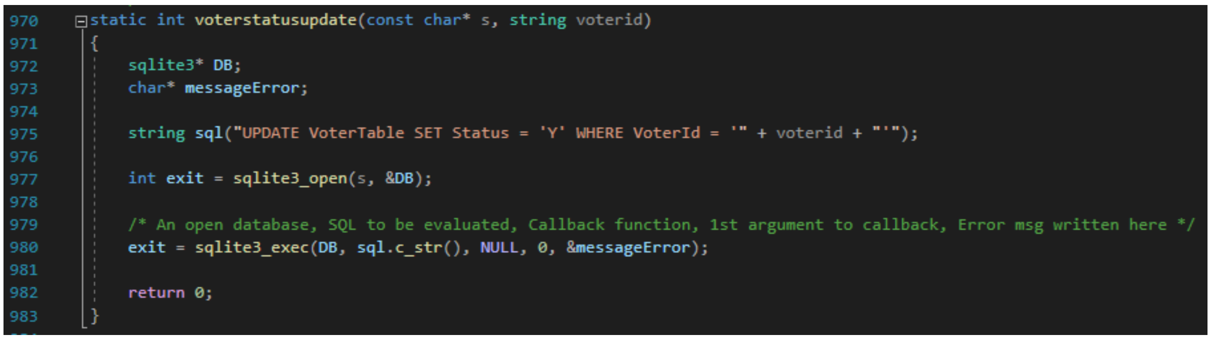
```cpp
// Select Candidate Information When Voter Is Voter
static int candidateinfovoteselection(const char* s, string division)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    char* messageError;

    int exit = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Name, CandidateID, Party FROM CandidateTable WHERE Division = '" + division + "'; "; // Specify The Data To Select

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    exit = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return 0;
}
```

Figure 48: candidaeinfovoteselection function



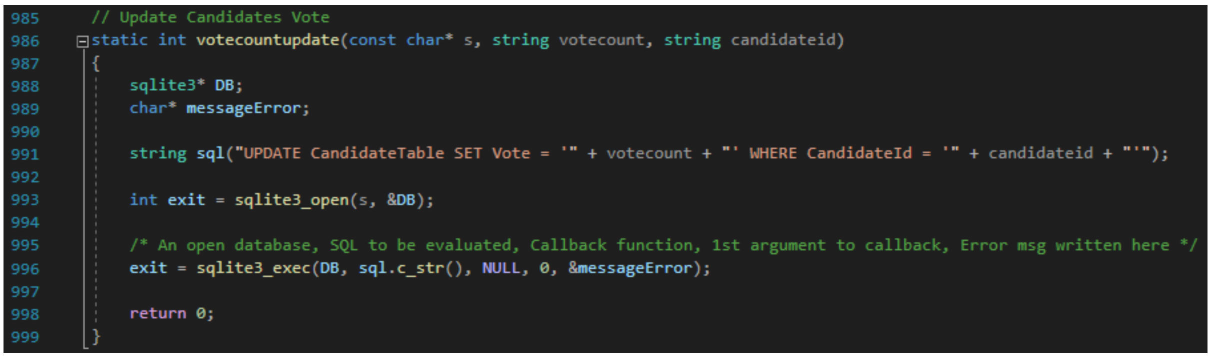
```cpp
    // Store Voter ID
vector <string> storevotereid(const char* s)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    vector <string> voterid;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    rc = sqlite3_prepare_v2(DB, "SELECT VoterID FROM VoterTable;", -1, &stmt, NULL); // Specify Which Data To Select

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        string id = (const char*)(sqlite3_column_text(stmt, 0)); // Assign The Data To Name Variable

        voterid.push_back(id); // Insert The Data Into Vector
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return voterid;
}
```

Figure 49: storevoterid function



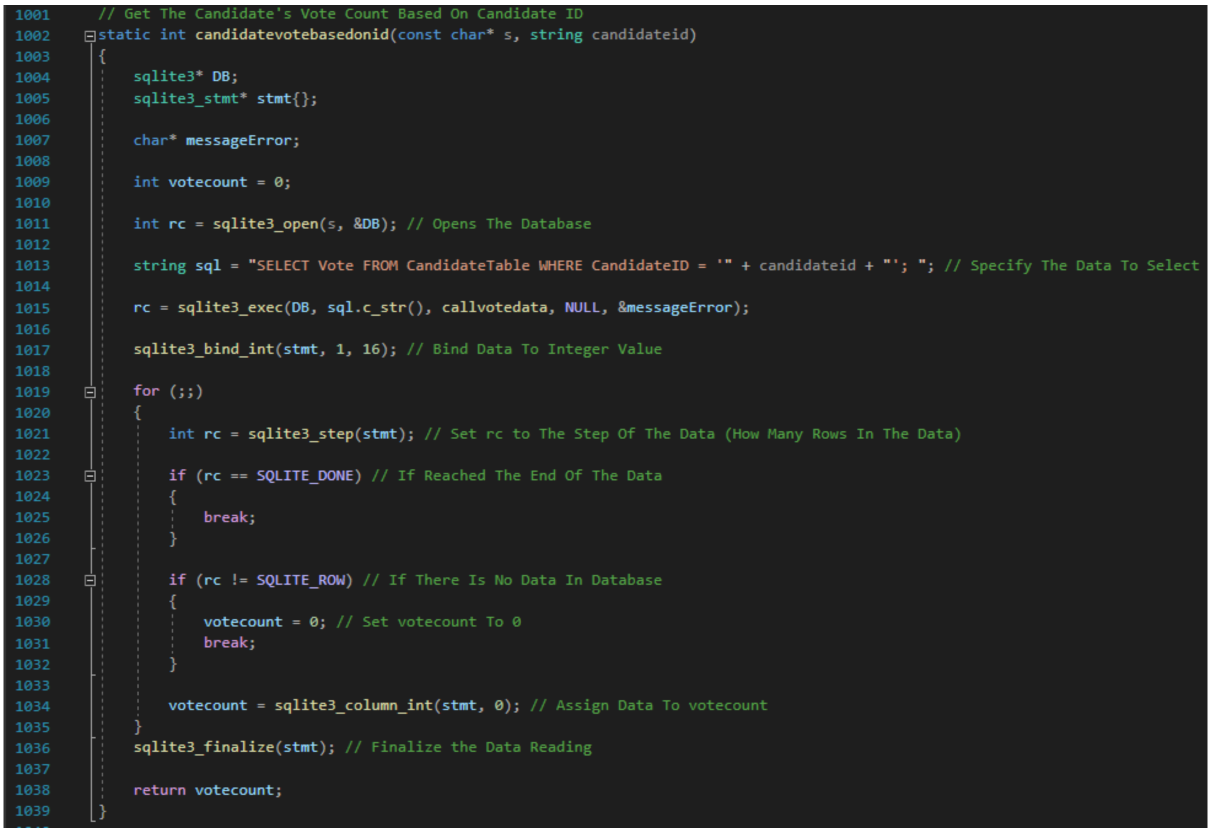
```cpp
static int voterstatusupdate(const char* s, string voterid)
{
    sqlite3* DB;
    char* messageError;

    string sql("UPDATE VoterTable SET Status = 'Y' WHERE VoterId = '" + voterid + "'");

    int exit = sqlite3_open(s, &DB);

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);

    return 0;
}
```

Figure 50: voterstatusupdate function



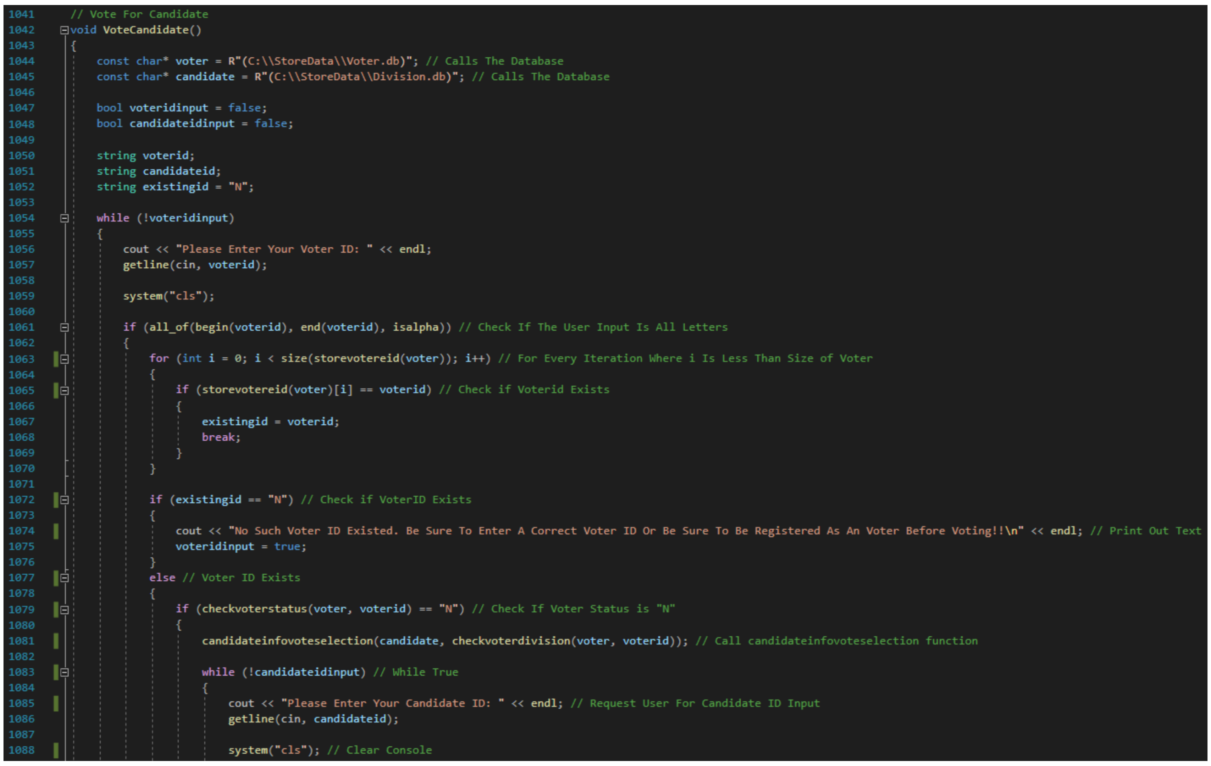
```cpp
    // Update Candidates Vote
static int votecountupdate(const char* s, string votecount, string candidateid)
{
    sqlite3* DB;
    char* messageError;

    string sql("UPDATE CandidateTable SET Vote = '" + votecount + "' WHERE CandidateId = '" + candidateid + "'");

    int exit = sqlite3_open(s, &DB);

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here */
    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);

    return 0;
}
```

Figure 51: votecountupdate function

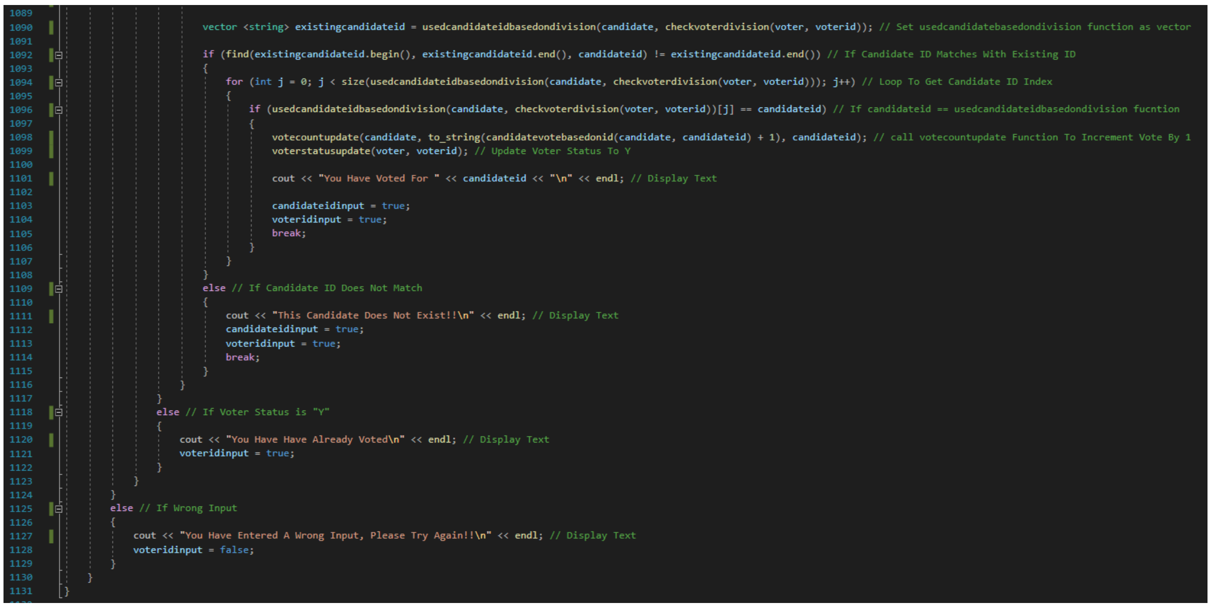


```cpp
// Get The Candidate's Vote Count Based On Candidate ID
static int candidatevotebasedonid(const char* s, string candidateid)
{
    sqlite3* DB;
    sqlite3_stmt* stmt{};

    char* messageError;

    int votecount = 0;

    int rc = sqlite3_open(s, &DB); // Opens The Database

    string sql = "SELECT Vote FROM CandidateTable WHERE CandidateID = '" + candidateid + "'; "; // Specify The Data To Select

    rc = sqlite3_exec(DB, sql.c_str(), callvotedata, NULL, &messageError);

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        if (rc != SQLITE_ROW) // If There Is No Data In Database
        {
            votecount = 0; // Set votecount To 0
            break;
        }

        votecount = sqlite3_column_int(stmt, 0); // Assign Data To votecount
    }
    sqlite3_finalize(stmt); // Finalize the Data Reading

    return votecount;
}
```

Figure 52: candidatevotebasedonid function

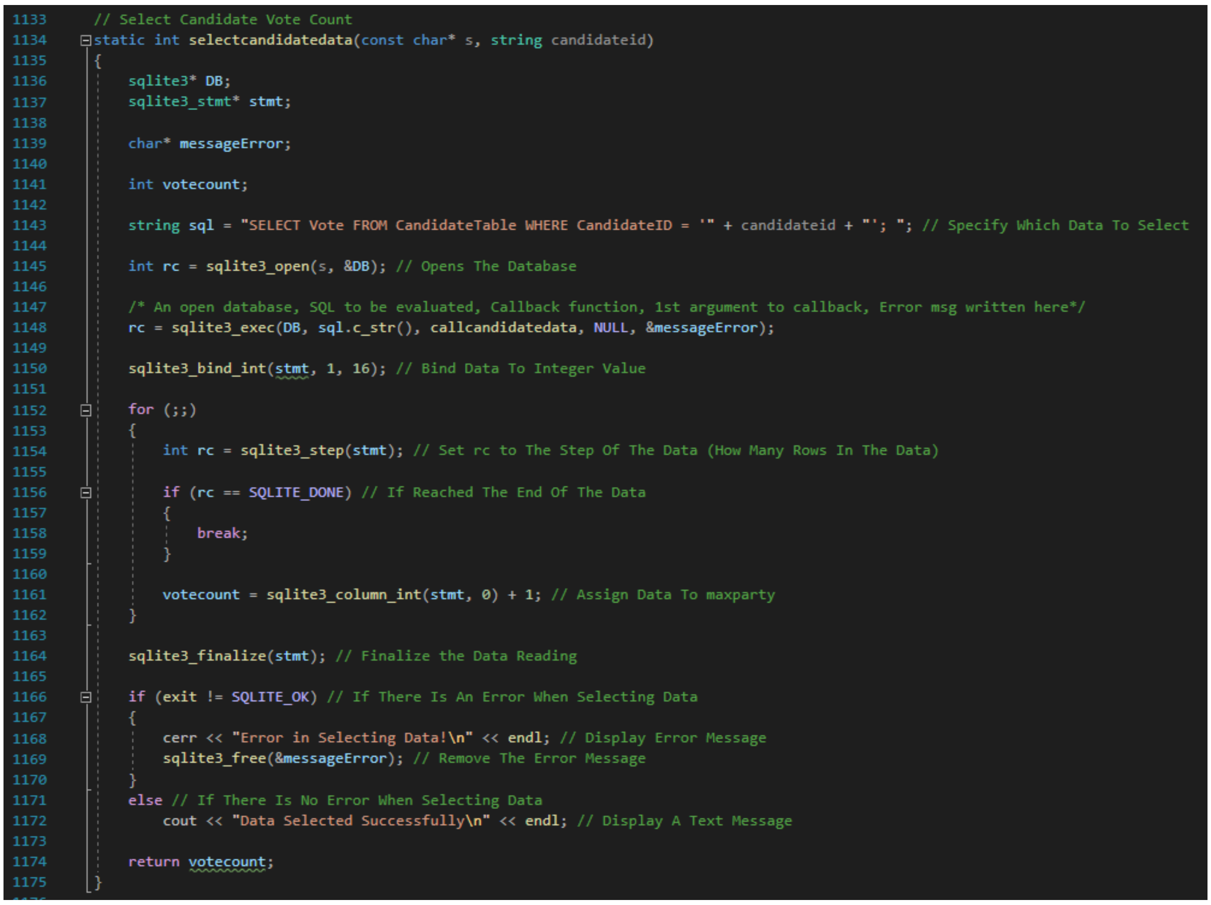


```cpp
// Vote For Candidate
void VoteCandidate()
{
    const char* voter = R"(C:\\StoreData\\Voter.db)"; // Calls The Database
    const char* candidate = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    bool voteridinput = false;
    bool candidateidinput = false;

    string voterid;
    string candidateid;
    string existingid = "N";

    while (!voteridinput)
    {
        cout << "Please Enter Your Voter ID: " << endl;
        getline(cin, voterid);

        system("cls");

        if (all_of(begin(voterid), end(voterid), isalpha)) // Check If The User Input Is All Letters
        {
            for (int i = 0; i < size(storevotereid(voter)); i++) // For Every Iteration Where i Is Less Than Size of Voter
            {
                if (storevotereid(voter)[i] == voterid) // Check if Voterid Exists
                {
                    existingid = voterid;
                    break;
                }
            }

            if (existingid == "N") // Check if VoterID Exists
            {
                cout << "No Such Voter ID Existed. Be Sure To Enter A Correct Voter ID Or Be Sure To Be Registered As An Voter Before Voting!!\n" << endl; // Print Out Text
                voteridinput = true;
            }
            else // Voter ID Exists
            {
                if (checkvoterstatus(voter, voterid) == "N") // Check If Voter Status is "N"
                {
                    candidateinfovoteselection(candidate, checkvoterdivision(voter, voterid)); // Call candidateinfovoteselection function

                    while (!candidateidinput) // While True
                    {
                        cout << "Please Enter Your Candidate ID: " << endl; // Request User For Candidate ID Input
                        getline(cin, candidateid);

                        system("cls"); // Clear Console
```



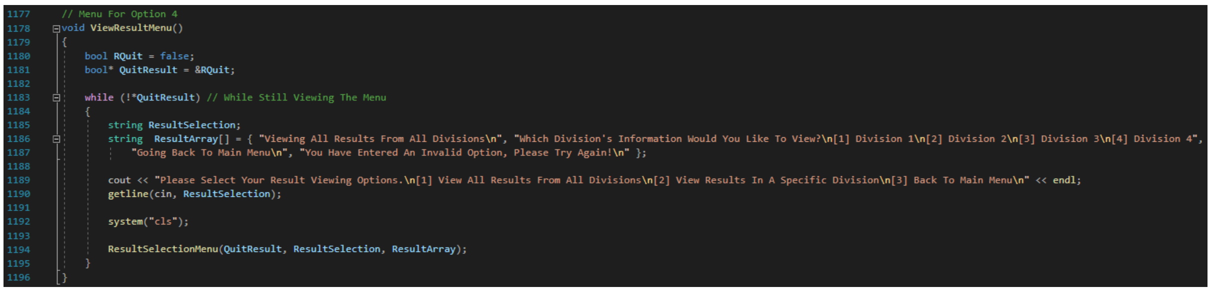
```cpp

                        vector <string> existingcandidateid = usedcandidateidbasedondivision(candidate, checkvoterdivision(voter, voterid)); // Set usedcandidatebasedondivision function as vector

                        if (find(existingcandidateid.begin(), existingcandidateid.end(), candidateid) != existingcandidateid.end()) // If Candidate ID Matches With Existing ID
                        {
                            for (int j = 0; j < size(usedcandidateidbasedondivision(candidate, checkvoterdivision(voter, voterid))); j++) // Loop To Get Candidate ID Index
                            {
                                if (usedcandidateidbasedondivision(candidate, checkvoterdivision(voter, voterid))[j] == candidateid) // If candidateid == usedcandidateidbasedondivision fucntion
                                {
                                    votecountupdate(candidate, to_string(candidatevotebasedonid(candidate, candidateid) + 1), candidateid); // call votecountupdate Function To Increment Vote By 1
                                    voterstatusupdate(voter, voterid); // Update Voter Status To Y

                                    cout << "You Have Voted For " << candidateid << "\n" << endl; // Display Text

                                    candidateidinput = true;
                                    voteridinput = true;
                                    break;
                                }
                            }
                        }
                        else // If Candidate ID Does Not Match
                        {
                            cout << "This Candidate Does Not Exist!!\n" << endl; // Display Text
                            candidateidinput = true;
                            voteridinput = true;
                            break;
                        }
                    }
                }
                else // If Voter Status is "Y"
                {
                    cout << "You Have Have Already Voted\n" << endl; // Display Text
                    voteridinput = true;
                }
            }
        }
        else // If Wrong Input
        {
            cout << "You Have Entered A Wrong Input, Please Try Again!!\n" << endl; // Display Text
            voteridinput = false;
        }
    }
}
```

Figure 53: VoteCandidate function



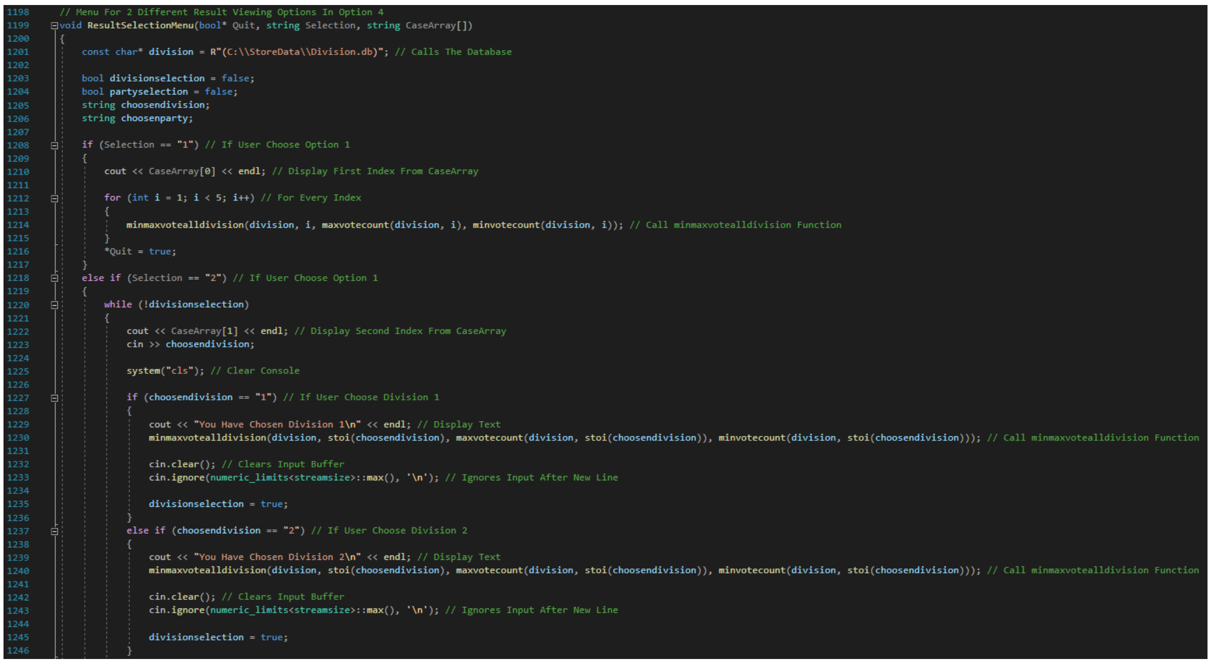
```cpp
// Select Candidate Vote Count
static int selectcandidatedata(const char* s, string candidateid)
{
    sqlite3* DB;
    sqlite3_stmt* stmt;

    char* messageError;

    int votecount;

    string sql = "SELECT Vote FROM CandidateTable WHERE CandidateID = '" + candidateid + "'; "; // Specify Which Data To Select

    int rc = sqlite3_open(s, &DB); // Opens The Database

    /* An open database, SQL to be evaluated, Callback function, 1st argument to callback, Error msg written here*/
    rc = sqlite3_exec(DB, sql.c_str(), callcandidatedata, NULL, &messageError);

    sqlite3_bind_int(stmt, 1, 16); // Bind Data To Integer Value

    for (;;)
    {
        int rc = sqlite3_step(stmt); // Set rc to The Step Of The Data (How Many Rows In The Data)

        if (rc == SQLITE_DONE) // If Reached The End Of The Data
        {
            break;
        }

        votecount = sqlite3_column_int(stmt, 0) + 1; // Assign Data To maxparty
    }

    sqlite3_finalize(stmt); // Finalize the Data Reading

    if (exit != SQLITE_OK) // If There Is An Error When Selecting Data
    {
        cerr << "Error in Selecting Data!\n" << endl; // Display Error Message
        sqlite3_free(&messageError); // Remove The Error Message
    }
    else // If There Is No Error When Selecting Data
        cout << "Data Selected Successfully\n" << endl; // Display A Text Message

    return votecount;
}
```

Figure 54: selectcandidatedata function

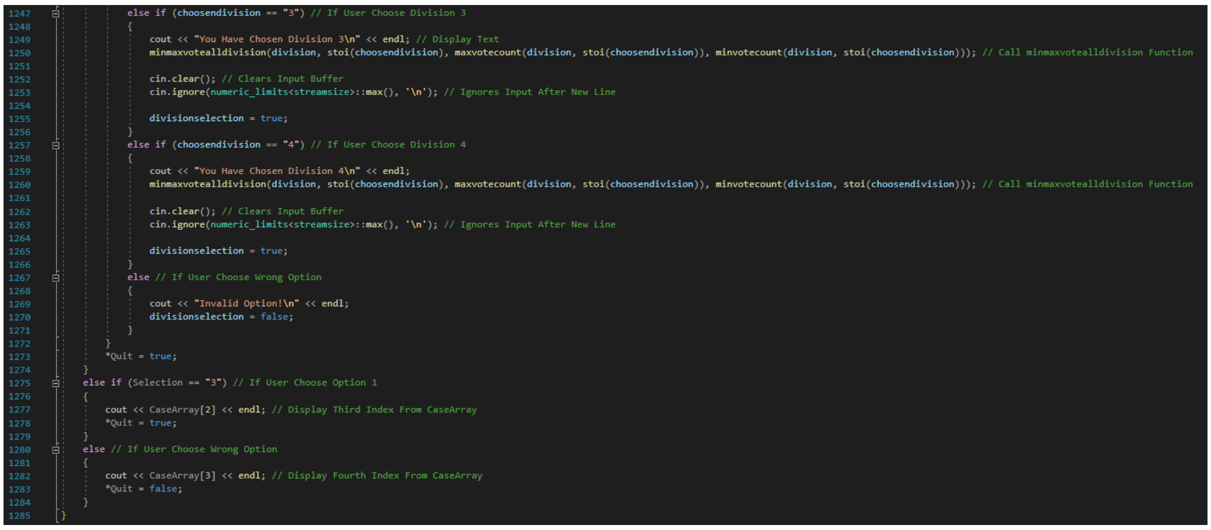


```cpp
// Menu For Option 4
void ViewResultMenu()
{
    bool RQuit = false;
    bool* QuitResult = &RQuit;

    while (!*QuitResult) // While Still Viewing The Menu
    {
        string ResultSelection;
        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",
            "Going Back To Main Menu\n", "You Have Entered An Invalid Option, Please Try Again!\n" };

        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;
        getline(cin, ResultSelection);

        system("cls");

        ResultSelectionMenu(QuitResult, ResultSelection, ResultArray);
    }
}
```

Figure 55: ViewResultMenu function

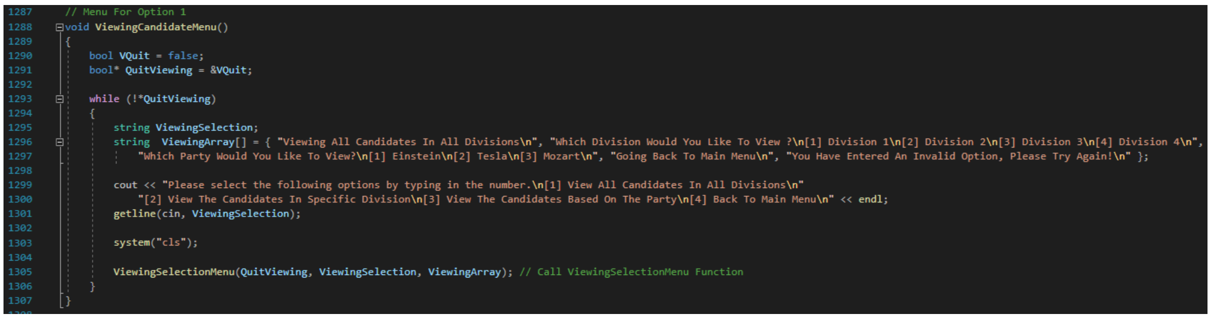


```cpp
// Menu For 2 Different Result Viewing Options In Option 4
void ResultSelectionMenu(bool* Quit, string Selection, string CaseArray[])
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    bool divisionselection = false;
    bool partyselection = false;
    string choosendivision;
    string choosenparty;

    if (Selection == "1") // If User Choose Option 1
    {
        cout << CaseArray[0] << endl; // Display First Index From CaseArray

        for (int i = 1; i < 5; i++) // For Every Index
        {
            minmaxvotealldivision(division, i, maxvotecount(division, i), minvotecount(division, i)); // Call minmaxvotealldivision Function
        }
        *Quit = true;
    }
    else if (Selection == "2") // If User Choose Option 1
    {
        while (!divisionselection)
        {
            cout << CaseArray[1] << endl; // Display Second Index From CaseArray
            cin >> choosendivision;

            system("cls"); // Clear Console

            if (choosendivision == "1") // If User Choose Division 1
            {
                cout << "You Have Chosen Division 1\n" << endl; // Display Text
                minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else if (choosendivision == "2") // If User Choose Division 2
            {
                cout << "You Have Chosen Division 2\n" << endl; // Display Text
                minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
```



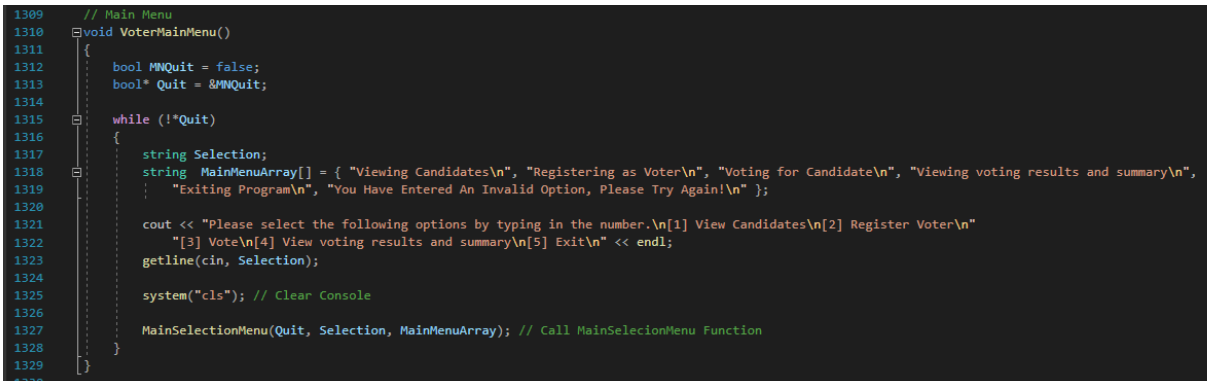
```cpp
            else if (choosendivision == "3") // If User Choose Division 3
            {
                cout << "You Have Chosen Division 3\n" << endl; // Display Text
                minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else if (choosendivision == "4") // If User Choose Division 4
            {
                cout << "You Have Chosen Division 4\n" << endl;
                minmaxvotealldivision(division, stoi(choosendivision), maxvotecount(division, stoi(choosendivision)), minvotecount(division, stoi(choosendivision))); // Call minmaxvotealldivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else // If User Choose Wrong Option
            {
                cout << "Invalid Option!\n" << endl;
                divisionselection = false;
            }
        }
        *Quit = true;
    }
    else if (Selection == "3") // If User Choose Option 1
    {
        cout << CaseArray[2] << endl; // Display Third Index From CaseArray
        *Quit = true;
    }
    else // If User Choose Wrong Option
    {
        cout << CaseArray[3] << endl; // Display Fourth Index From CaseArray
        *Quit = false;
    }
}
```

Figure 56: ResultSelectionMenu function



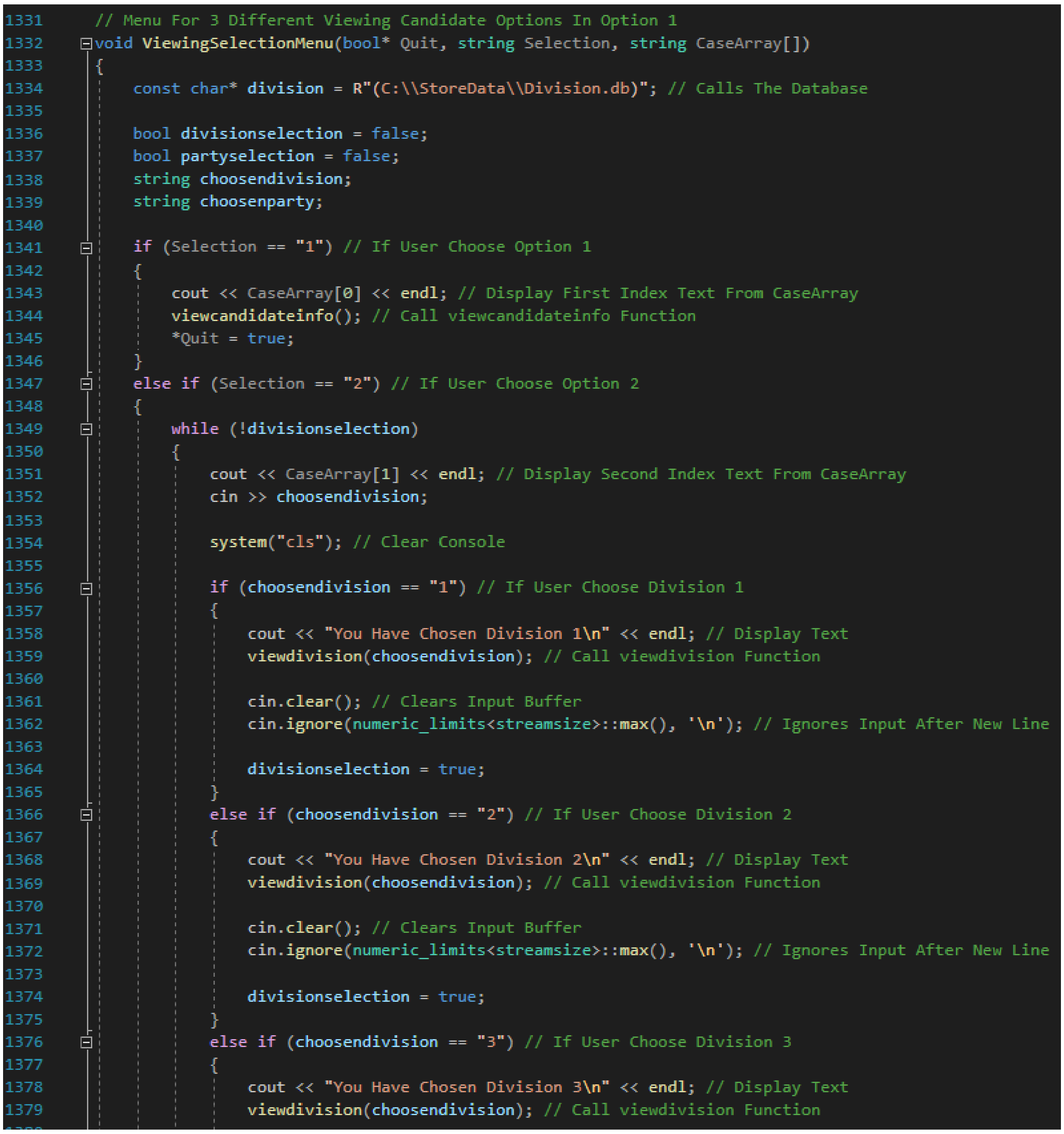
```cpp
// Menu For Option 1
void ViewingCandidateMenu()
{
    bool VQuit = false;
    bool* QuitViewing = &VQuit;

    while (!*QuitViewing)
    {
        string ViewingSelection;
        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",
            "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" };

        cout << "Please select the following options by typing in the number.\n[1] View All Candidates In All Divisions\n"
            "[2] View The Candidates In Specific Division\n[3] View The Candidates Based On The Party\n[4] Back To Main Menu\n" << endl;
        getline(cin, ViewingSelection);

        system("cls");

        ViewingSelectionMenu(QuitViewing, ViewingSelection, ViewingArray); // Call ViewingSelectionMenu Function
    }
}
```

Figure 57: ViewingCandidateMenu function

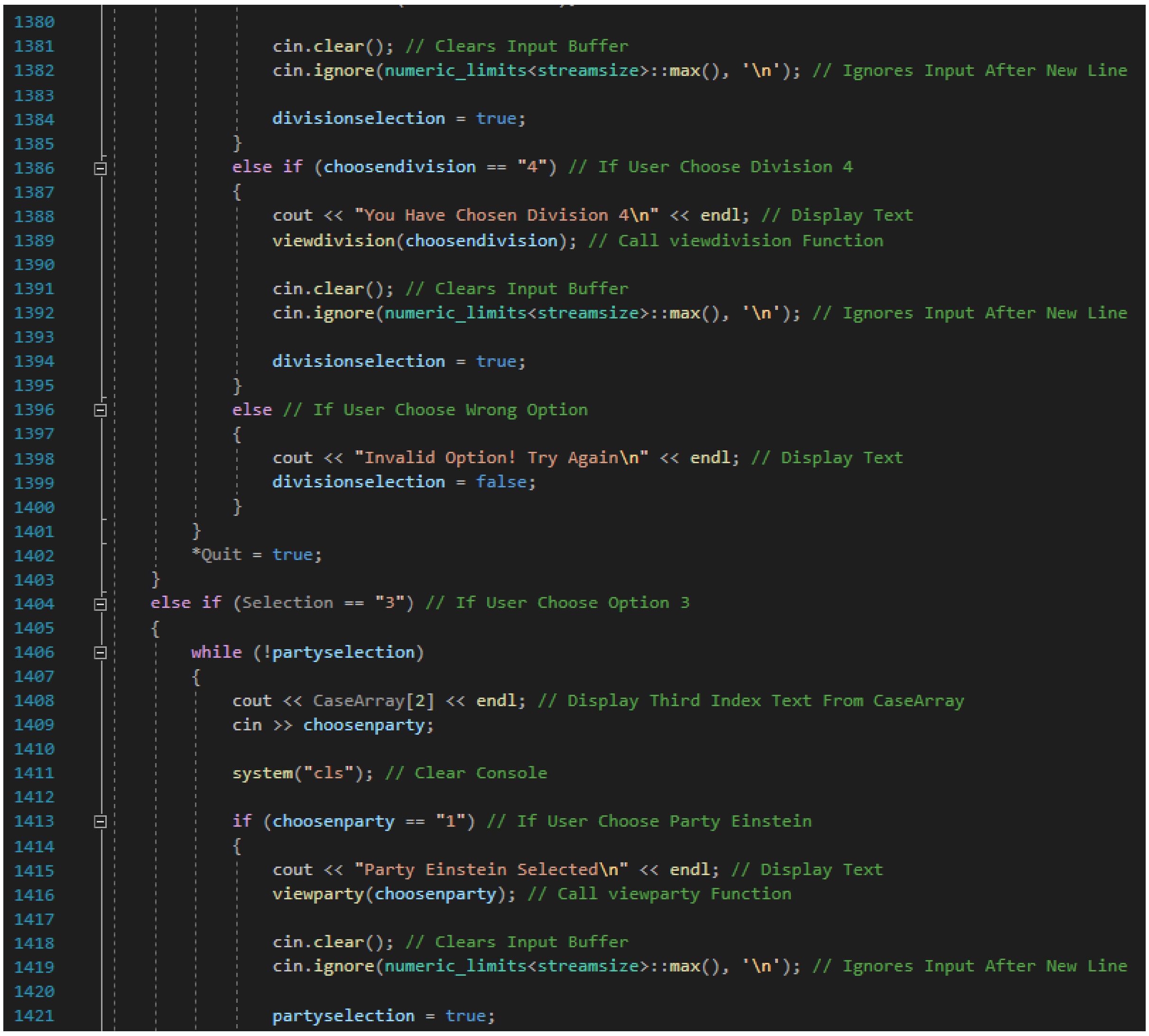


```cpp
// Main Menu
void VoterMainMenu()
{
    bool MNQuit = false;
    bool* Quit = &MNQuit;

    while (!*Quit)
    {
        string Selection;
        string  MainMenuArray[] = { "Viewing Candidates\n", "Registering as Voter\n", "Voting for Candidate\n", "Viewing voting results and summary\n",
            "Exiting Program\n", "You Have Entered An Invalid Option, Please Try Again!\n" };

        cout << "Please select the following options by typing in the number.\n[1] View Candidates\n[2] Register Voter\n"
            "[3] Vote\n[4] View voting results and summary\n[5] Exit\n" << endl;
        getline(cin, Selection);

        system("cls"); // Clear Console

        MainSelectionMenu(Quit, Selection, MainMenuArray); // Call MainSelecionMenu Function
    }
}
```

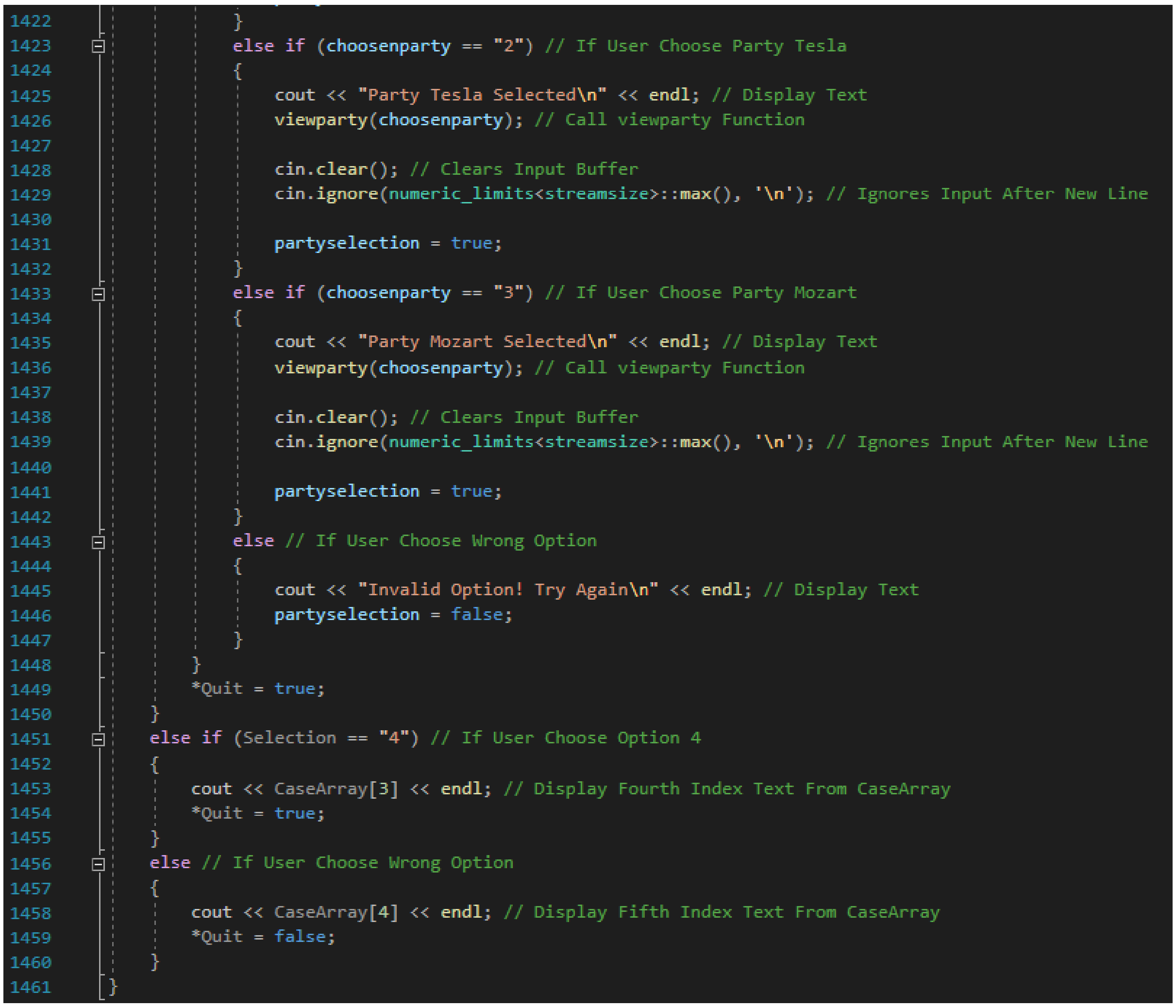
Figure 58: VoterMainMenu function



```
// Menu For 3 Different Viewing Candidate Options In Option 1
void ViewingSelectionMenu(bool* Quit, string Selection, string CaseArray[])
{
    const char* division = R"(C:\\StoreData\\Division.db)"; // Calls The Database

    bool divisionselection = false;
    bool partyselection = false;
    string choosendivision;
    string choosenparty;

    if (Selection == "1") // If User Choose Option 1
    {
        cout << CaseArray[0] << endl; // Display First Index Text From CaseArray
        viewcandidateinfo(); // Call viewcandidateinfo Function
        *Quit = true;
    }
    else if (Selection == "2") // If User Choose Option 2
    {
        while (!divisionselection)
        {
            cout << CaseArray[1] << endl; // Display Second Index Text From CaseArray
            cin >> choosendivision;

            system("cls"); // Clear Console

            if (choosendivision == "1") // If User Choose Division 1
            {
                cout << "You Have Chosen Division 1\n" << endl; // Display Text
                viewdivision(choosendivision); // Call viewdivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else if (choosendivision == "2") // If User Choose Division 2
            {
                cout << "You Have Chosen Division 2\n" << endl; // Display Text
                viewdivision(choosendivision); // Call viewdivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else if (choosendivision == "3") // If User Choose Division 3
            {
                cout << "You Have Chosen Division 3\n" << endl; // Display Text
                viewdivision(choosendivision); // Call viewdivision Function
```

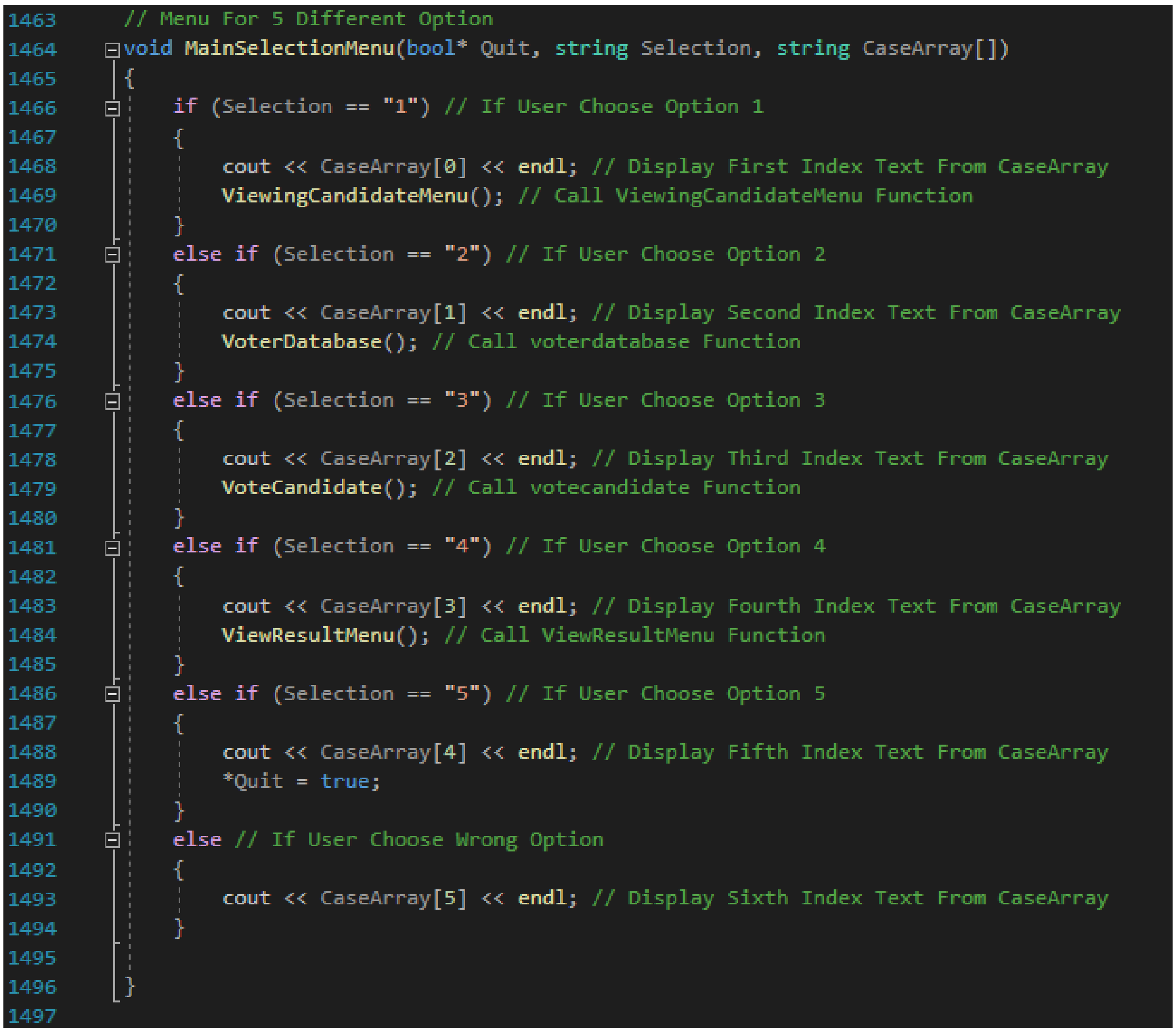


```
                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else if (choosendivision == "4") // If User Choose Division 4
            {
                cout << "You Have Chosen Division 4\n" << endl; // Display Text
                viewdivision(choosendivision); // Call viewdivision Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                divisionselection = true;
            }
            else // If User Choose Wrong Option
            {
                cout << "Invalid Option! Try Again\n" << endl; // Display Text
                divisionselection = false;
            }
        }
        *Quit = true;
    }
    else if (Selection == "3") // If User Choose Option 3
    {
        while (!partyselection)
        {
            cout << CaseArray[2] << endl; // Display Third Index Text From CaseArray
            cin >> choosenparty;

            system("cls"); // Clear Console

            if (choosenparty == "1") // If User Choose Party Einstein
            {
                cout << "Party Einstein Selected\n" << endl; // Display Text
                viewparty(choosenparty); // Call viewparty Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                partyselection = true;
```



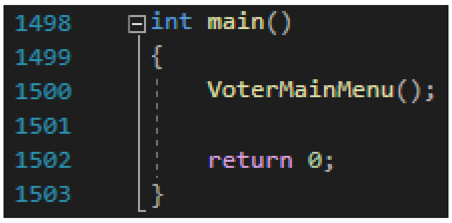
```cpp
            }
            else if (choosenparty == "2") // If User Choose Party Tesla
            {
                cout << "Party Tesla Selected\n" << endl; // Display Text
                viewparty(choosenparty); // Call viewparty Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                partyselection = true;
            }
            else if (choosenparty == "3") // If User Choose Party Mozart
            {
                cout << "Party Mozart Selected\n" << endl; // Display Text
                viewparty(choosenparty); // Call viewparty Function

                cin.clear(); // Clears Input Buffer
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignores Input After New Line

                partyselection = true;
            }
            else // If User Choose Wrong Option
            {
                cout << "Invalid Option! Try Again\n" << endl; // Display Text
                partyselection = false;
            }
        }
        *Quit = true;
    }
    else if (Selection == "4") // If User Choose Option 4
    {
        cout << CaseArray[3] << endl; // Display Fourth Index Text From CaseArray
        *Quit = true;
    }
    else // If User Choose Wrong Option
    {
        cout << CaseArray[4] << endl; // Display Fifth Index Text From CaseArray
        *Quit = false;
    }
}
```

Figure 59:ViewingSelectionMenu function



```cpp
    // Menu For 5 Different Option
void MainSelectionMenu(bool* Quit, string Selection, string CaseArray[])
{
    if (Selection == "1") // If User Choose Option 1
    {
        cout << CaseArray[0] << endl; // Display First Index Text From CaseArray
        ViewingCandidateMenu(); // Call ViewingCandidateMenu Function
    }
    else if (Selection == "2") // If User Choose Option 2
    {
        cout << CaseArray[1] << endl; // Display Second Index Text From CaseArray
        VoterDatabase(); // Call voterdatabase Function
    }
    else if (Selection == "3") // If User Choose Option 3
    {
        cout << CaseArray[2] << endl; // Display Third Index Text From CaseArray
        VoteCandidate(); // Call votecandidate Function
    }
    else if (Selection == "4") // If User Choose Option 4
    {
        cout << CaseArray[3] << endl; // Display Fourth Index Text From CaseArray
        ViewResultMenu(); // Call ViewResultMenu Function
    }
    else if (Selection == "5") // If User Choose Option 5
    {
        cout << CaseArray[4] << endl; // Display Fifth Index Text From CaseArray
        *Quit = true;
    }
    else // If User Choose Wrong Option
    {
        cout << CaseArray[5] << endl; // Display Sixth Index Text From CaseArray
    }

}

```

Figure 60: MainSelectionMenu function

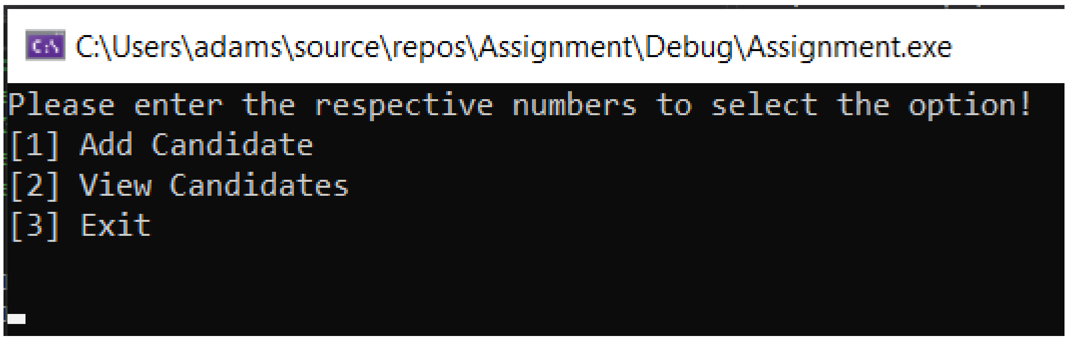


```cpp
int main()
{
    VoterMainMenu();

    return 0;
}
```

Figure 61: main function

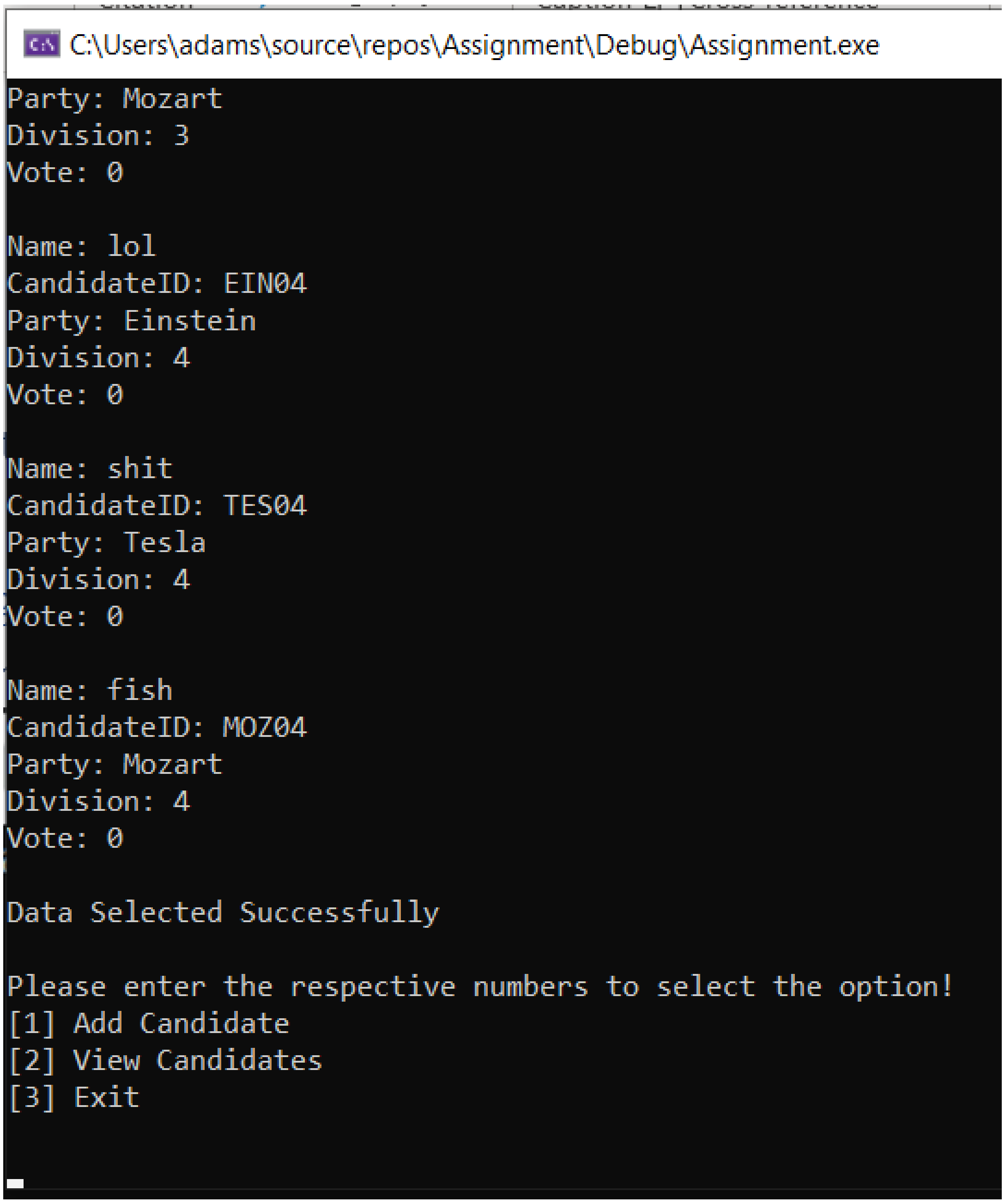
5.3 Program Walkthrough

5.3.1 Candidate Program



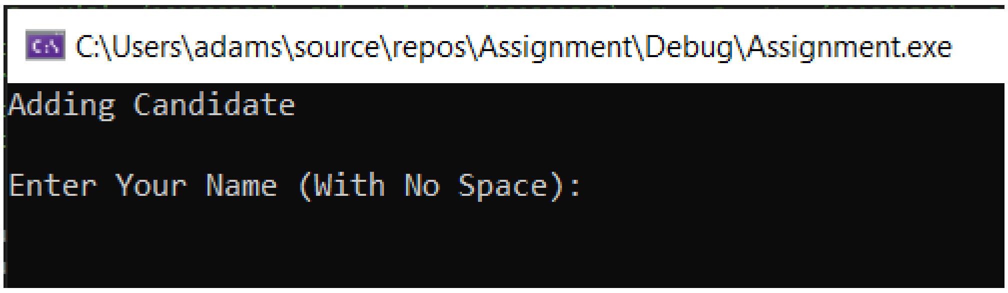
```
C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
Please enter the respective numbers to select the option!
[1] Add Candidate
[2] View Candidates
[3] Exit
```

Figure 62: Main Menu UI For Candidate Program

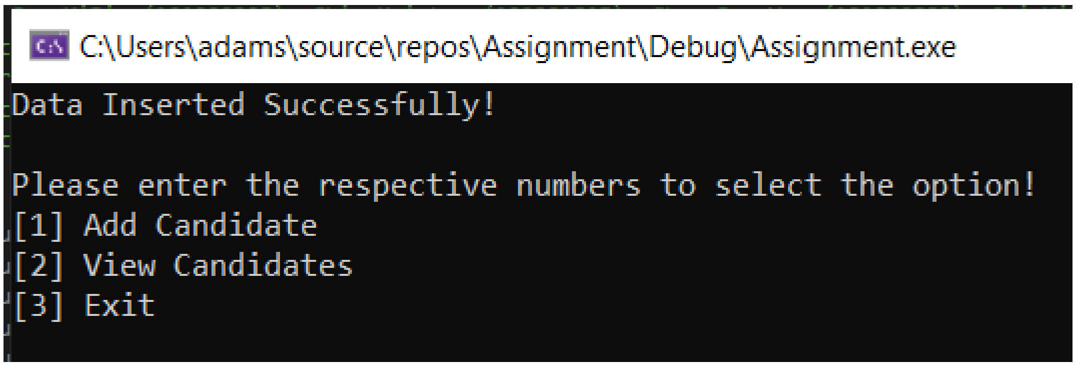


```
C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe
Party: Mozart
Division: 3
Vote: 0

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

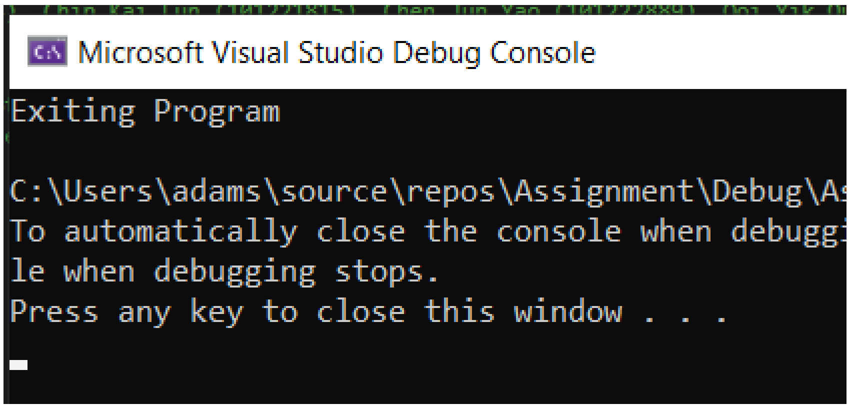

C:\Users\adams\source\repos\Assignment\Debug\Assignment.exe

```
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
```

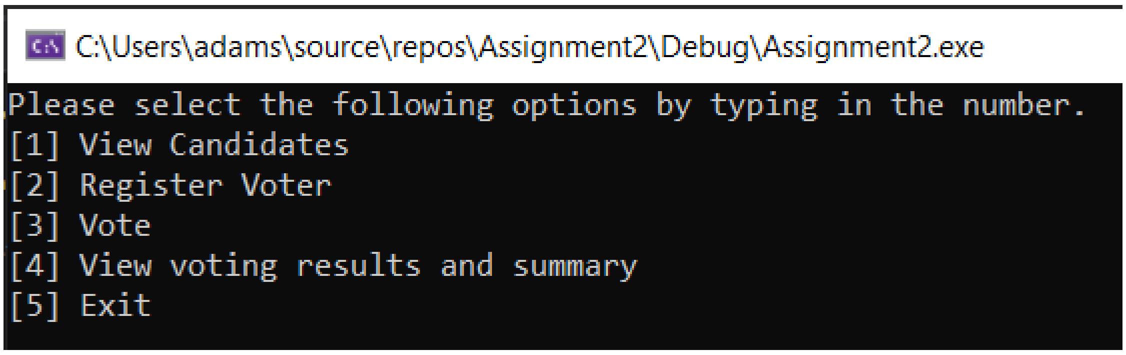
Figure 64: Add Candidate UI


Microsoft Visual Studio Debug Console

```
Exiting Program

C:\Users\adams\source\repos\Assignment\Debug\As
To automatically close the console when debuggi
le when debugging stops.
Press any key to close this window . . .
```

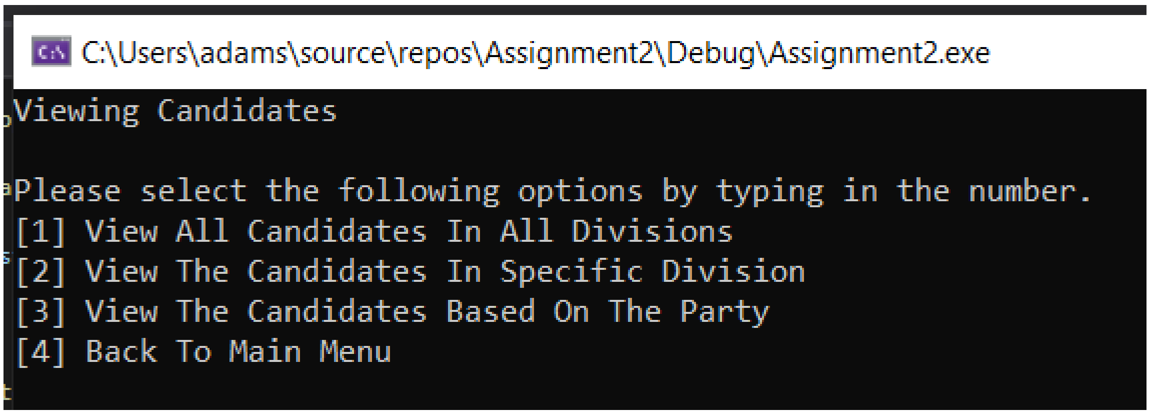
Figure 65: Exit Program UI

5.3.2 Voter Program


C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe

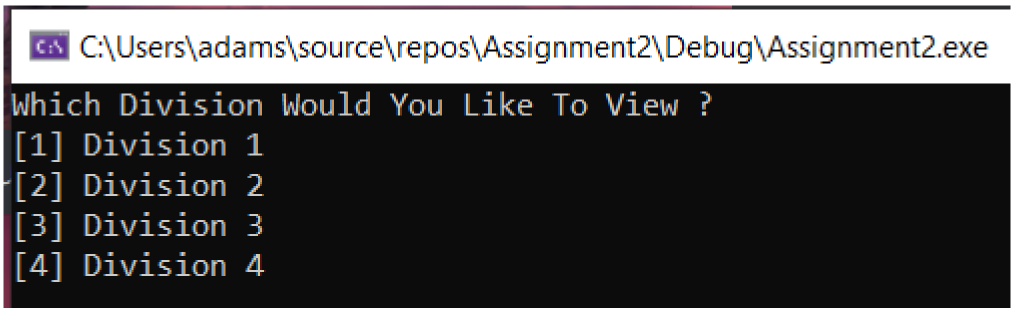
```
Please select the following options by typing in the number.
[1] View Candidates
[2] Register Voter
[3] Vote
[4] View voting results and summary
[5] Exit
```

Figure 66: Main Menu UI For Voter Program



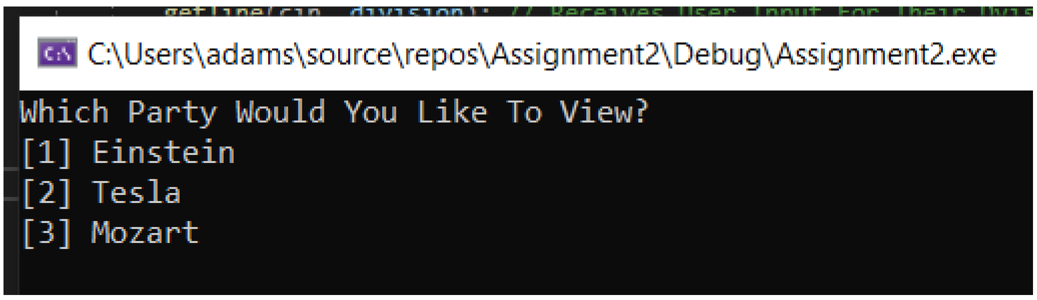
```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Viewing Candidates

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



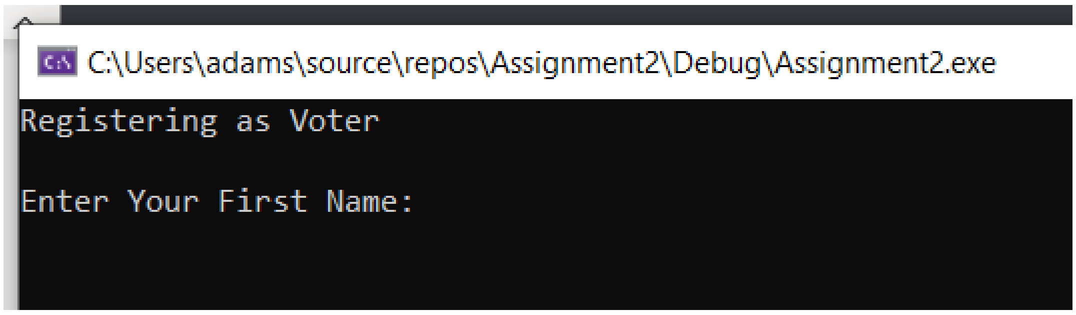
```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Which Division Would You Like To View ?
[1] Division 1
[2] Division 2
[3] Division 3
[4] Division 4
```

Figure 68: Viewing Candidates Based on Division UI

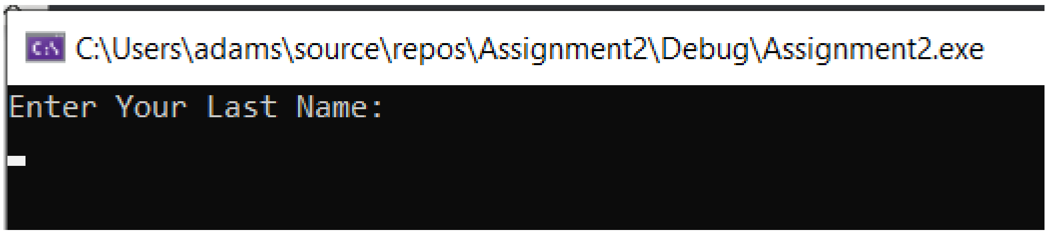


```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Which Party Would You Like To View?
[1] Einstein
[2] Tesla
[3] Mozart
```

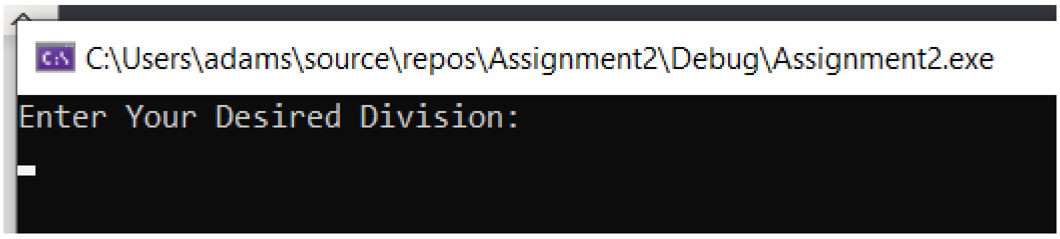
Figure 69: Viewing Candidate Based on party UI



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

Enter Your First Name:
```

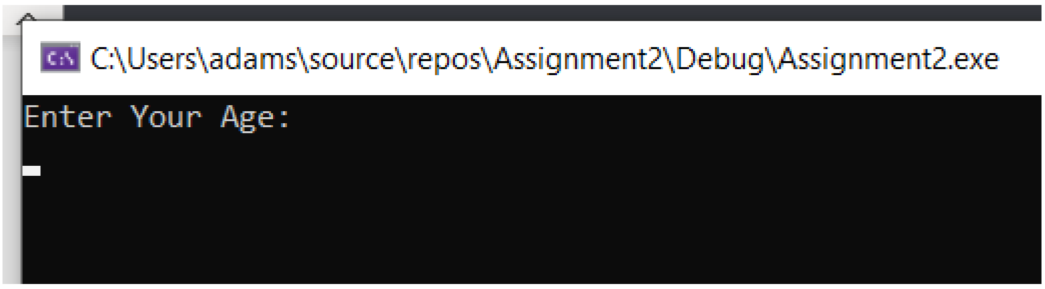


```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Enter Your Last Name:
```



C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe

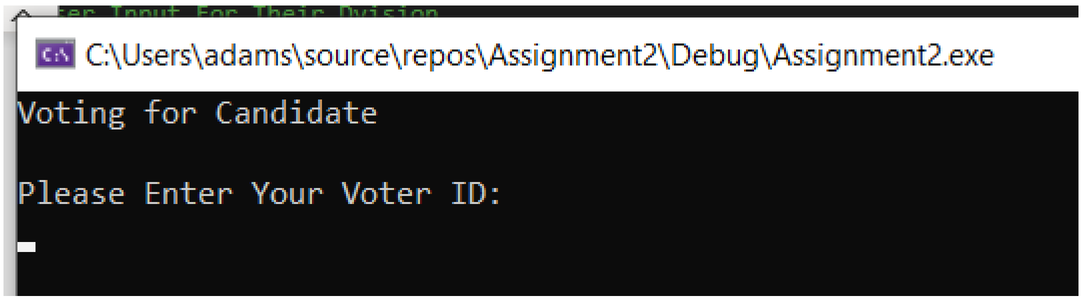
```
Enter Your Desired Division:
```



C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe

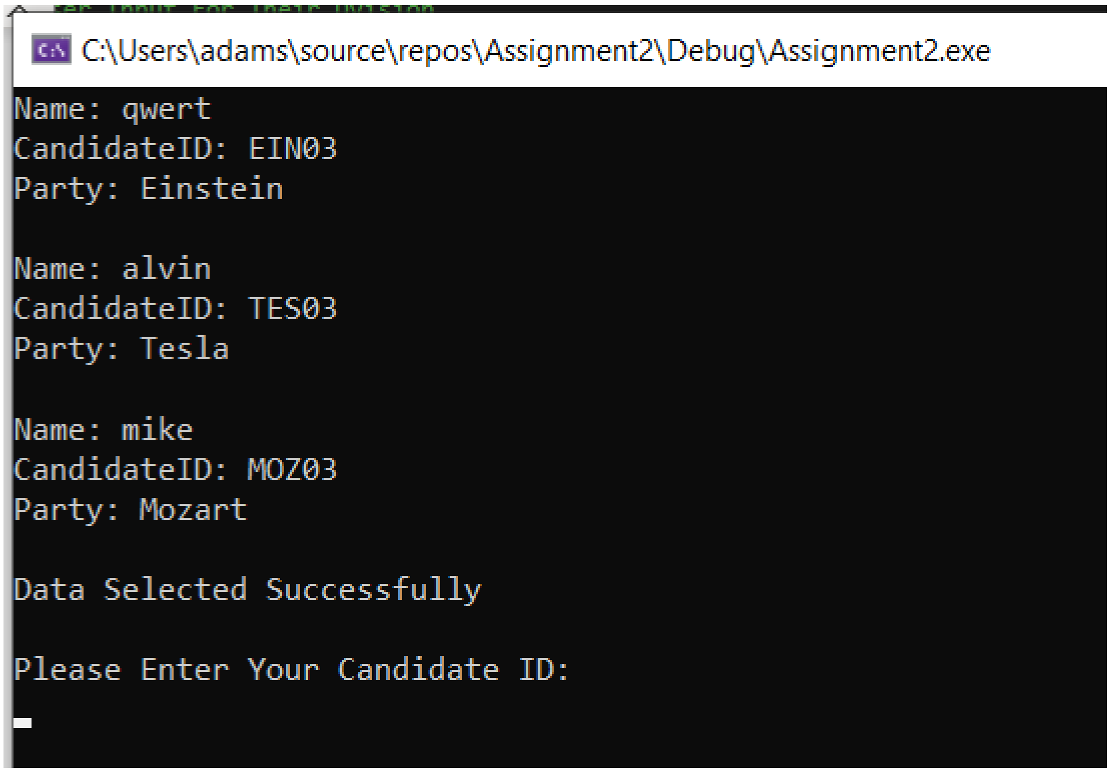
```
Enter Your Age:
```

Figure 70: Register Voter Menu UI



C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe

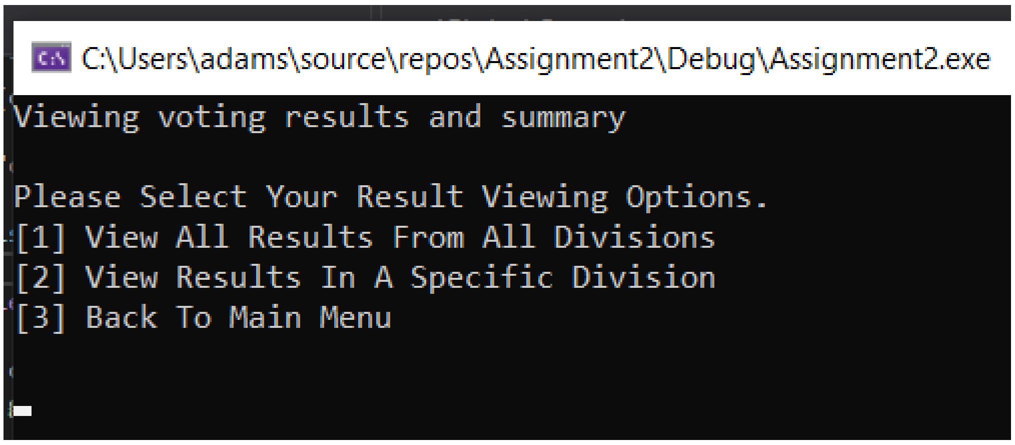
```
Voting for Candidate

Please Enter Your Voter ID:
```



C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe

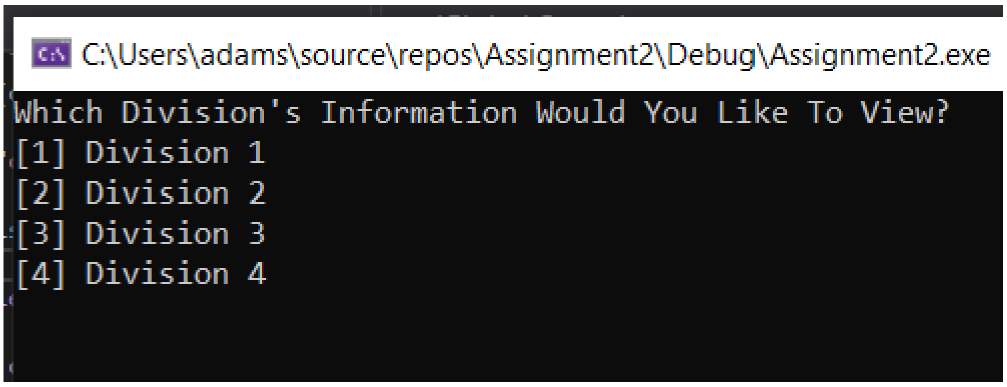
```
Name: qwert
CandidateID: EIN03
Party: Einstein

Name: alvin
CandidateID: TES03
Party: Tesla

Name: mike
CandidateID: MOZ03
Party: Mozart

Data Selected Successfully

Please Enter Your Candidate ID:
```

Figure 71: Vote Menu UI



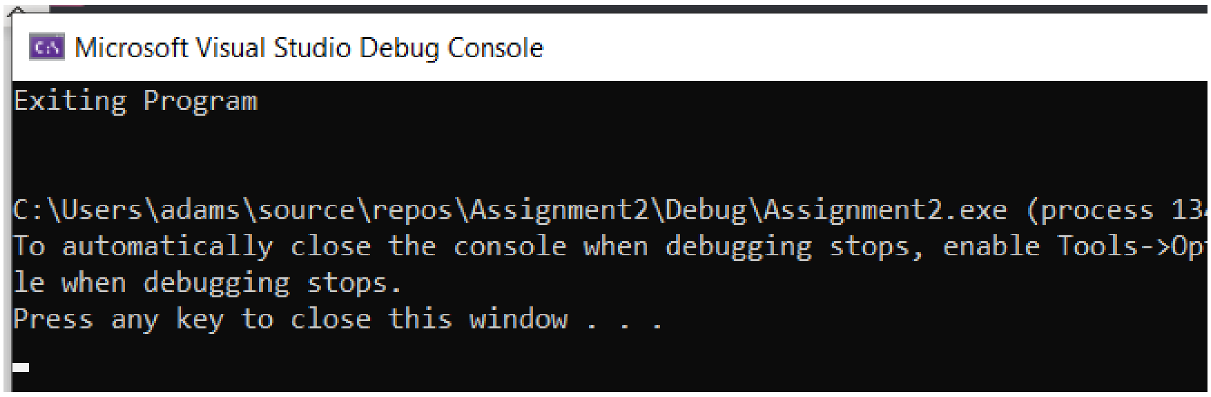
```
C:\Users\adams\source\repos\Assignment2\Debug\Assignment2.exe
Viewing voting results and summary

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 13
To automatically close the console when debugging stops, enable Tools->Op
le when debugging stops.
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
AdamSze **(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.