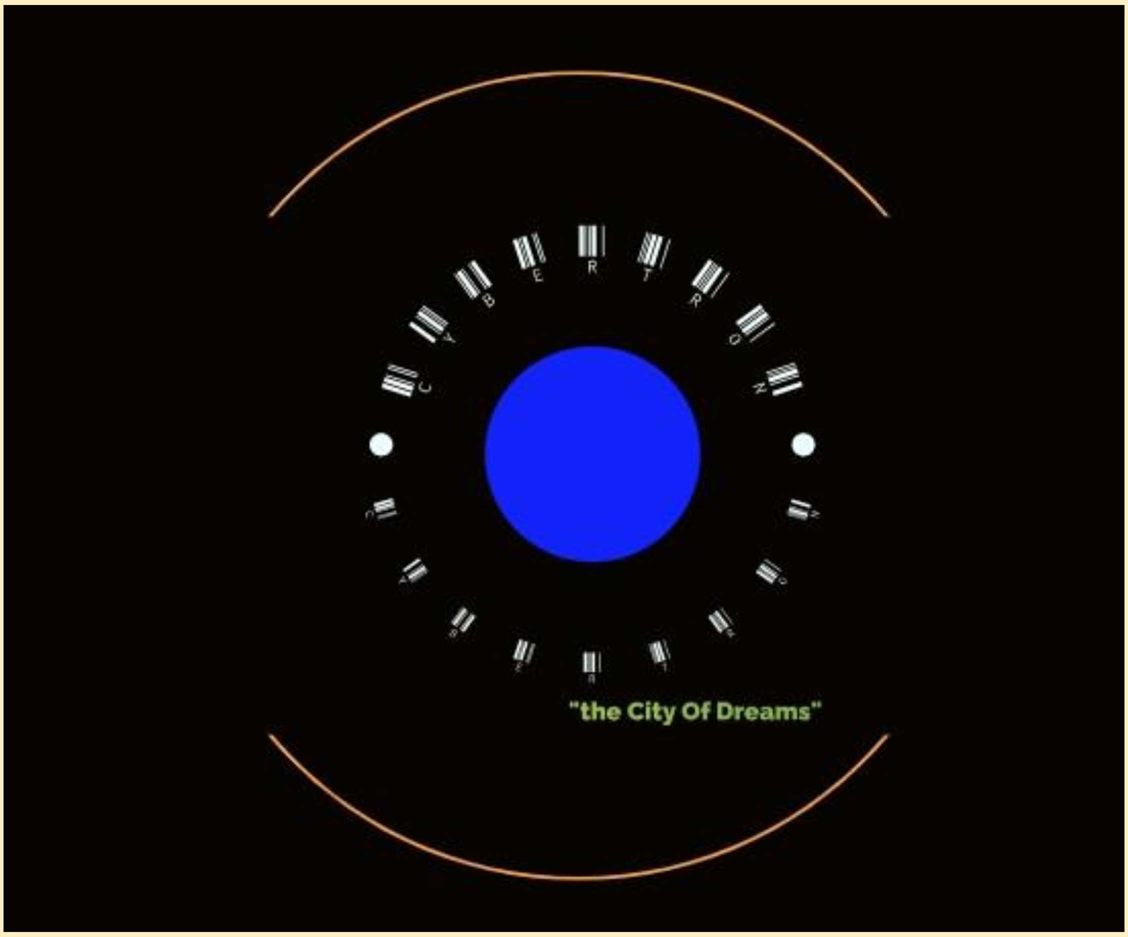
Cybertron Election

“Vote all you want; it will not matter.”



/ * - - ---

N a m e : W i l l i a m S t e e l e M a h l e r

S t u d e n t n u m b e r : 7 5 8 3 7 1 0

E m a i l a d d r e s s : w s m 1 3 6 @ u o w m a i l . e d u . a u

S u b j e c t C o d e : C S ICI251

A s s i g n m e n t n u m b e r : 2

--- * /

Contents

Executive Summary

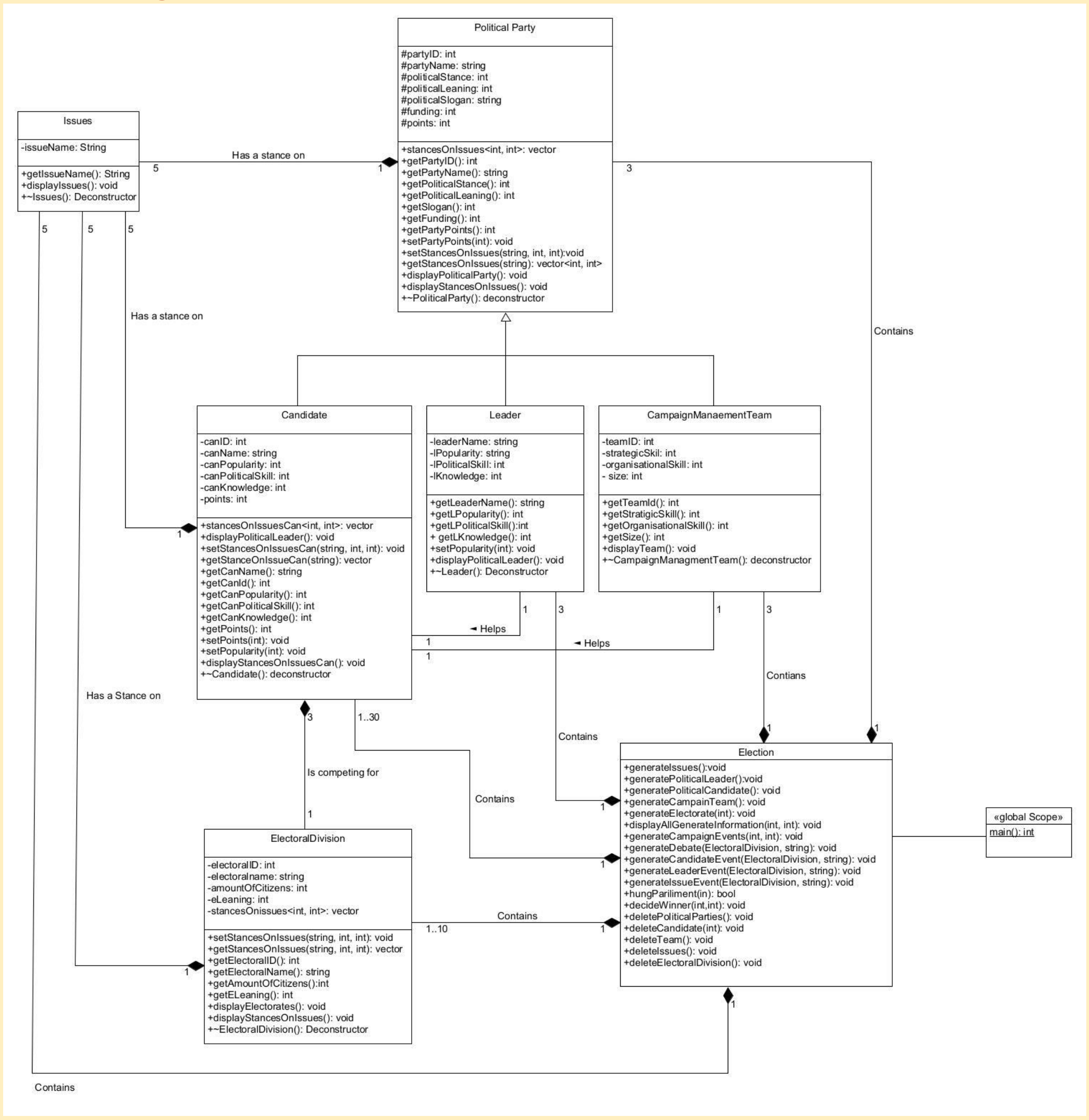
The program submitted contains three files "driver.cpp, header.cpp, and header.h". When compiled they will form a program that will simulate an election in a fictional place called Cybertron. I have designed the program so that every time it is compiled or run, it will generate completely different Winners. The following report will explain in detail how my program runs and what all the code does. Sorry it's a little long, I went a little bit overboard as I was having too much fun coding it.

Note:

This was written before fully completing the output to the terminal. At the moment it is crude and I would like to fix it up to make it look neater and more understandable but the main code is all there.

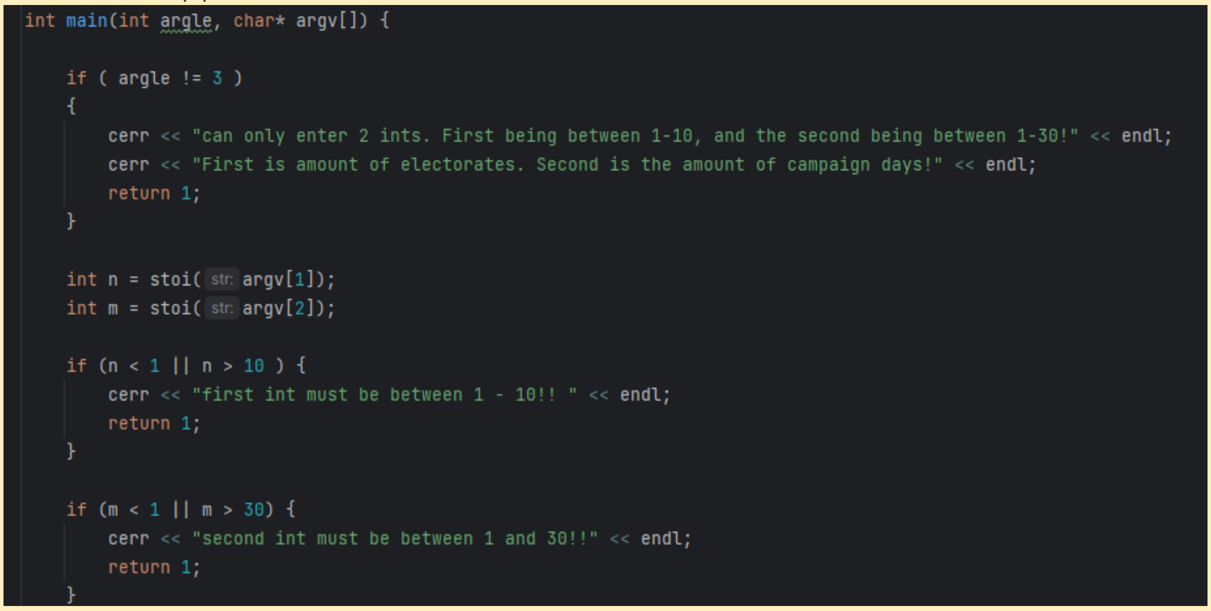
All the Points asked in the specification will be chapters called [Stuff] Point 1, 2, 3 etc.:

Umlet diagram


Political Party
#partyID: int
#partyName: string
#politicalStance: int
#politicalLeaning: int
#politicalSlogan: string
#funding: int
#points: int
+stancesOnIssues<int, int>: vector
+getPartyID(): int
+getPartyName(): string
+getPoliticalStance(): int
+getPoliticalLeaning(): int
+getSlogan(): int
+getFunding(): int
+getPartyPoints(): int
+setPartyPoints(int): void
+setStancesOnIssues(string, int, int):void
+getStancesOnIssues(string): vector<int, int>
+displayPoliticalParty(): void
+displayStancesOnIssues(): void
+~PoliticalParty(): deconstructor
Issues
-issueName: String
+getIssueName(): String
+displayIssues(): void
+~Issues(): Deconstructor
Has a stance on
5
1
3
Contains
5
5
5
Has a stance on
Candidate
-canID: int
-canName: string
-canPopularity: int
-canPoliticalSkill: int
-canKnowledge: int
-points: int
+stancesOnIssuesCan<int, int>: vector
+displayPoliticalLeader(): void
+setStancesOnIssuesCan(string, int, int): void
+getStanceOnIssueCan(string): vector
+getCanName(): string
+getCanId(): int
+getCanPopularity(): int
+getCanPoliticalSkill(): int
+getCanKnowledge(): int
+getPoints(): int
+setPoints(int): void
+setPopularity(int): void
+displayStancesOnIssuesCan(): void
+~Candidate(): deconstructor
1
Leader
-leaderName: string
-lPopularity: string
-lPoliticalSkill: int
-lKnowledge: int
+getLeaderName(): string
+getLPopularity(): int
+getLPoliticalSkill():int
+ getLKnowledge(): int
+setPopularity(int): void
+displayPoliticalLeader(): void
+~Leader(): Deconstructor
CampaignManaementTeam
-teamID: int
-strategicSkil: int
-organisationalSkill: int
- size: int
+getTeamId(): int
+getStratigicSkill(): int
+getOrganisationalSkill(): int
+getSize(): int
+displayTeam(): void
+~CampaignManagmentTeam(): deconstructor
1
3
◄ Helps
1
1
◄ Helps
1
3
Contians
Has a Stance on
3
1..30
Contains
1
1
Is competing for
1
Contains
Election
+generateIssues():void
+generatePoliticalLeader():void
+generatePoliticalCandidate(): void
+generateCampainTeam(): void
+generateElectorate(int): void
+displayAllGenerateInformation(int, int): void
+generateCampaignEvents(int, int): void
+generateDebate(ElectoralDivision, string): void
+generateCandidateEvent(ElectoralDivision, string): void
+generateLeaderEvent(ElectoralDivision, string): void
+generateIssueEvent(ElectoralDivision, string): void
+hungPariliment(in): bool
+decideWinner(int,int): void
+deletePoliticalParties(): void
+deleteCandidate(int): void
+deleteTeam(): void
+deleteIssues(): void
+deleteElectoralDivision(): void
1
1
1
«global Scope»
main(): int
ElectoralDivision
-electoralID: int
-electoralname: string
-amountOfCitizens: int
-eLeaning: int
-stancesOnissues<int, int>: vector
+setStancesOnIssues(string, int, int): void
+getStancesOnIssues(string, int, int): vector
+getElectoralID(): int
+getElectoralName(): string
+getAmountOfCitizens():int
+getELeaning(): int
+displayElectorates(): void
+displayStancesOnIssues(): void
+~ElectoralDivision(): Deconstructor
Contains
1..10
1
1
1
Contains

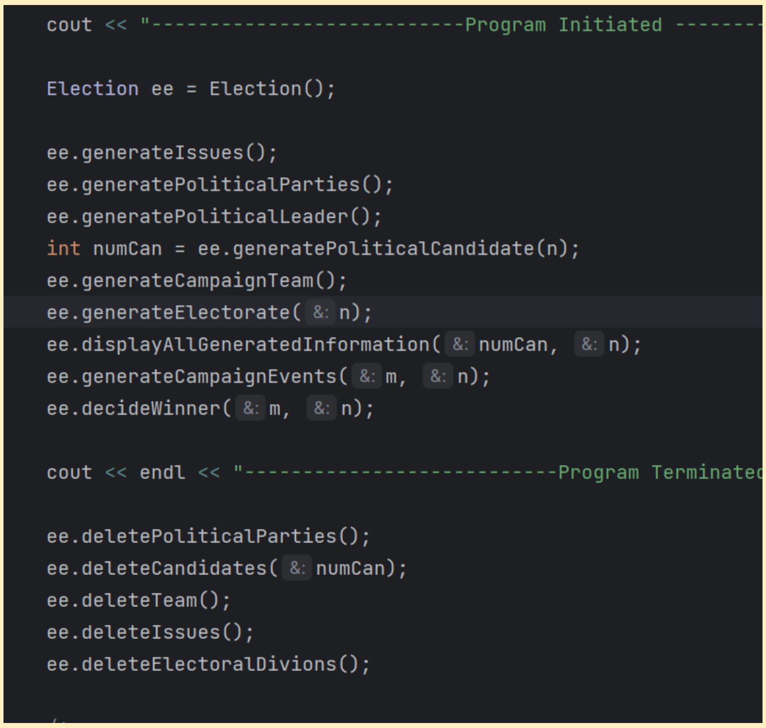
Code review

File Driver.cpp overview



```cpp
int main(int argle, char* argv[]) {

    if ( argle != 3 )
    {
        cerr << "can only enter 2 ints. First being between 1-10, and the second being between 1-30!" << endl;
        cerr << "First is amount of electorates. Second is the amount of campaign days!" << endl;
        return 1;
    }

    int n = stoi( str: argv[1]);
    int m = stoi( str: argv[2]);

    if (n < 1 || n > 10 ) {
        cerr << "first int must be between 1 - 10!! " << endl;
        return 1;
    }

    if (m < 1 || m > 30) {
        cerr << "second int must be between 1 and 30!!" << endl;
        return 1;
    }
```

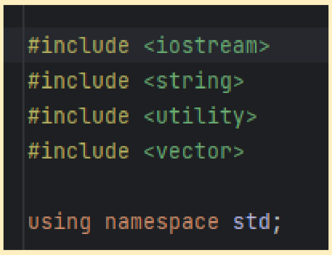
This code is to get 2 inputs from the command line, if there is more or less it throws an error. If the variables are not within a certain range then it throws an error



```cpp
    cout << "--------------------------Program Initiated -------

    Election ee = Election();

    ee.generateIssues();
    ee.generatePoliticalParties();
    ee.generatePoliticalLeader();
    int numCan = ee.generatePoliticalCandidate(n);
    ee.generateCampaignTeam();
    ee.generateElectorate( &: n);
    ee.displayAllGeneratedInformation( &: numCan,  &: n);
    ee.generateCampaignEvents( &: m,  &: n);
    ee.decideWinner( &: m,  &: n);

    cout << endl << "--------------------------Program Terminated

    ee.deletePoliticalParties();
    ee.deleteCandidates( &: numCan);
    ee.deleteTeam();
    ee.deleteIssues();
    ee.deleteElectoralDivions();
```

This is the driver that starts and runs the other two files. It runs from top to bottom, callaing all the methods in order. All the methods are in header.cpp. After -Program Terminated- I put in methods that will delete the pointers and the de-constructor can delete the memory

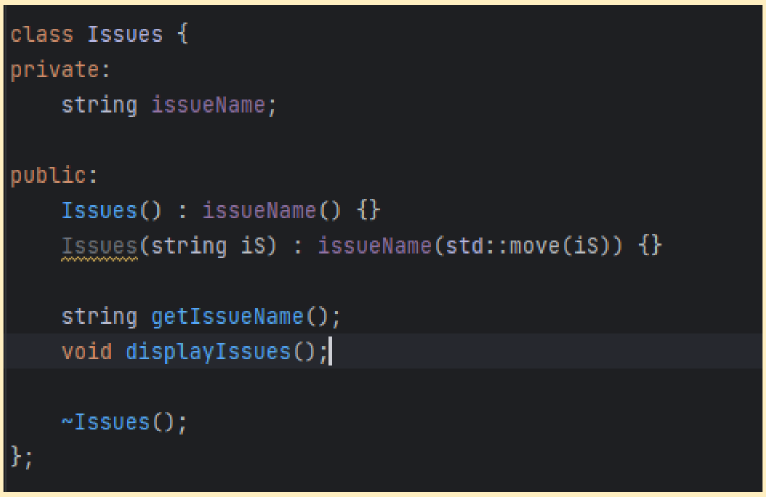
Header.h overview



```
#include <iostream>
#include <string>
#include <utility>
#include <vector>

using namespace std;
```

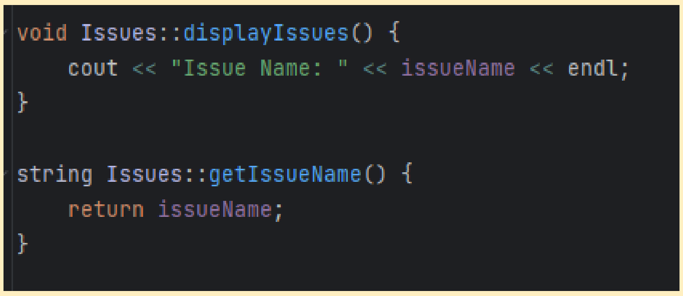
These are the imported libraries that I will be using

Class Issues



```
class Issues {
private:
    string issueName;

public:
    Issues() : issueName() {}
    Issues(string iS) : issueName(std::move(iS)) {}

    string getIssueName();
    void displayIssues();

    ~Issues();
};
```

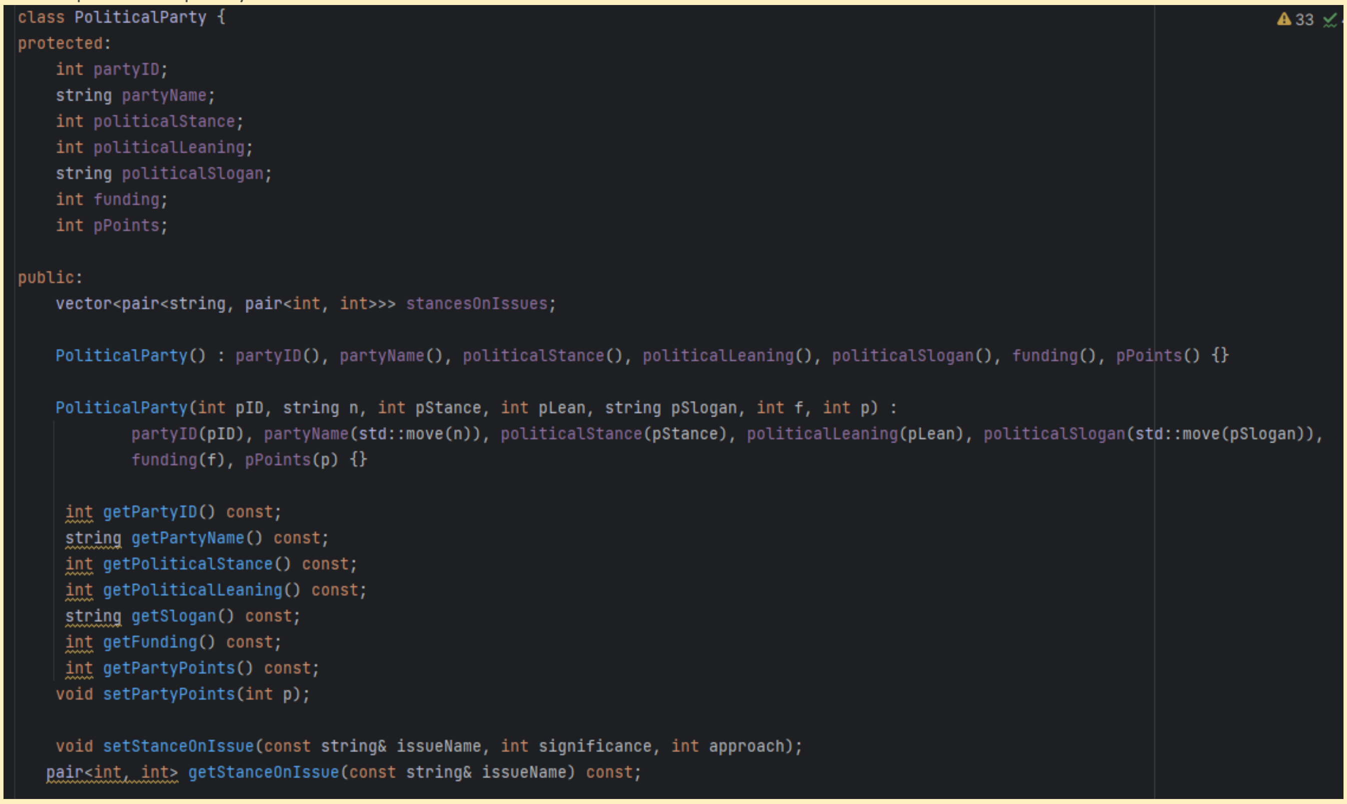
This is the Class for Issues, It generates the name for the 5 issues and creates a class out of it



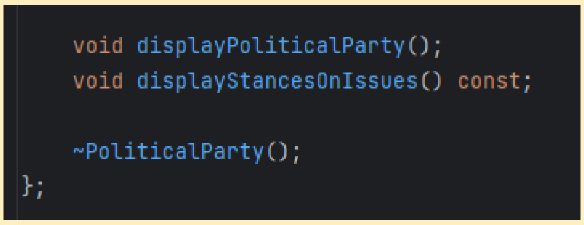
```
void Issues::displayIssues() {
    cout << "Issue Name: " << issueName << endl;
}

string Issues::getIssueName() {
    return issueName;
}
```

The method statement is in header.cpp.

Class political party



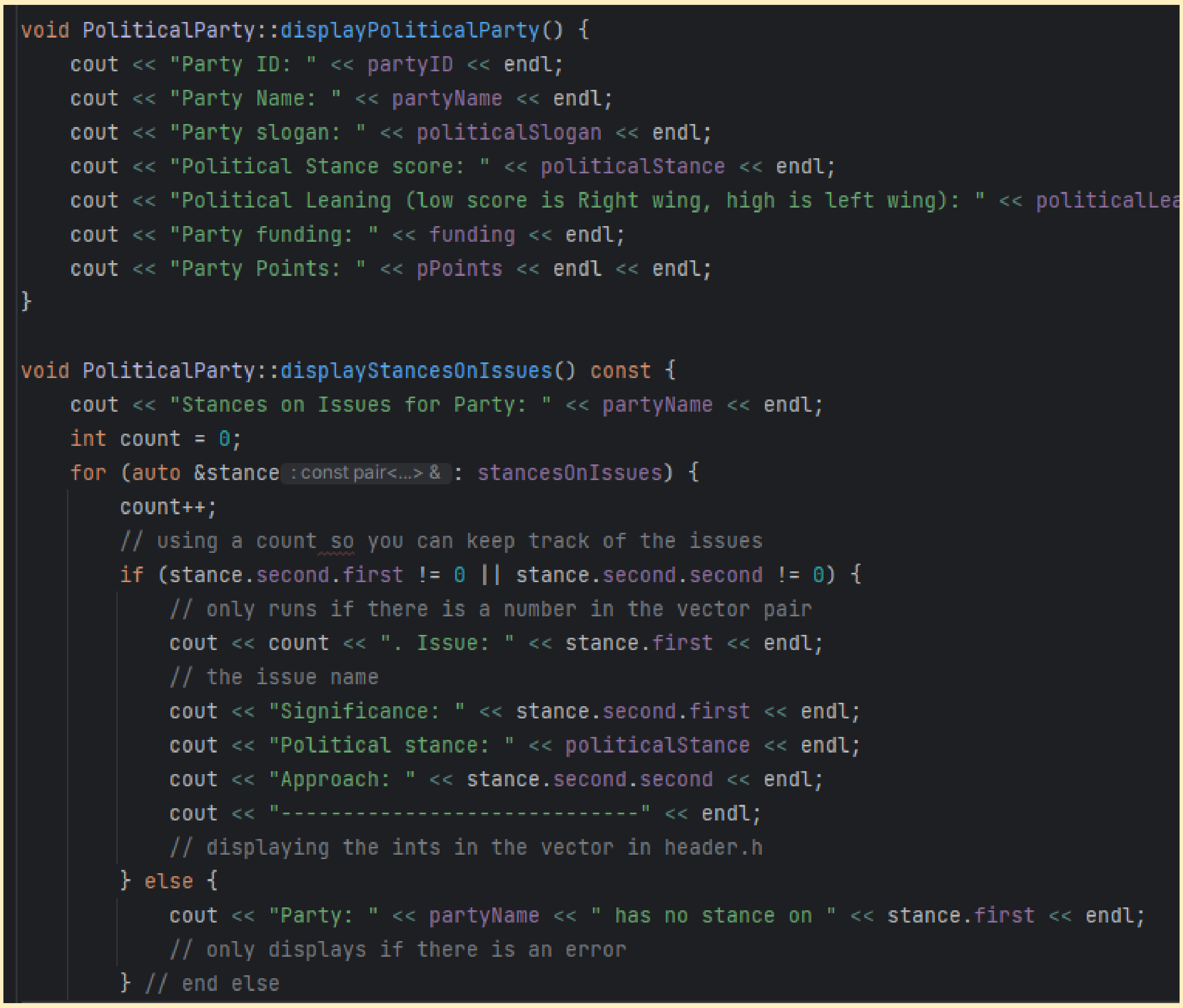
```
class PoliticalParty {
protected:
    int partyID;
    string partyName;
    int politicalStance;
    int politicalLeaning;
    string politicalSlogan;
    int funding;
    int pPoints;

public:
    vector<pair<string, pair<int, int>>> stancesOnIssues;

    PoliticalParty() : partyID(), partyName(), politicalStance(), politicalLeaning(), politicalSlogan(), funding(), pPoints() {}

    PoliticalParty(int pID, string n, int pStance, int pLean, string pSlogan, int f, int p) :
            partyID(pID), partyName(std::move(n)), politicalStance(pStance), politicalLeaning(pLean), politicalSlogan(std::move(pSlogan)),
            funding(f), pPoints(p) {}

     int getPartyID() const;
     string getPartyName() const;
     int getPoliticalStance() const;
     int getPoliticalLeaning() const;
     string getSlogan() const;
     int getFunding() const;
     int getPartyPoints() const;
    void setPartyPoints(int p);

    void setStanceOnIssue(const string& issueName, int significance, int approach);
   pair<int, int> getStanceOnIssue(const string& issueName) const;
```



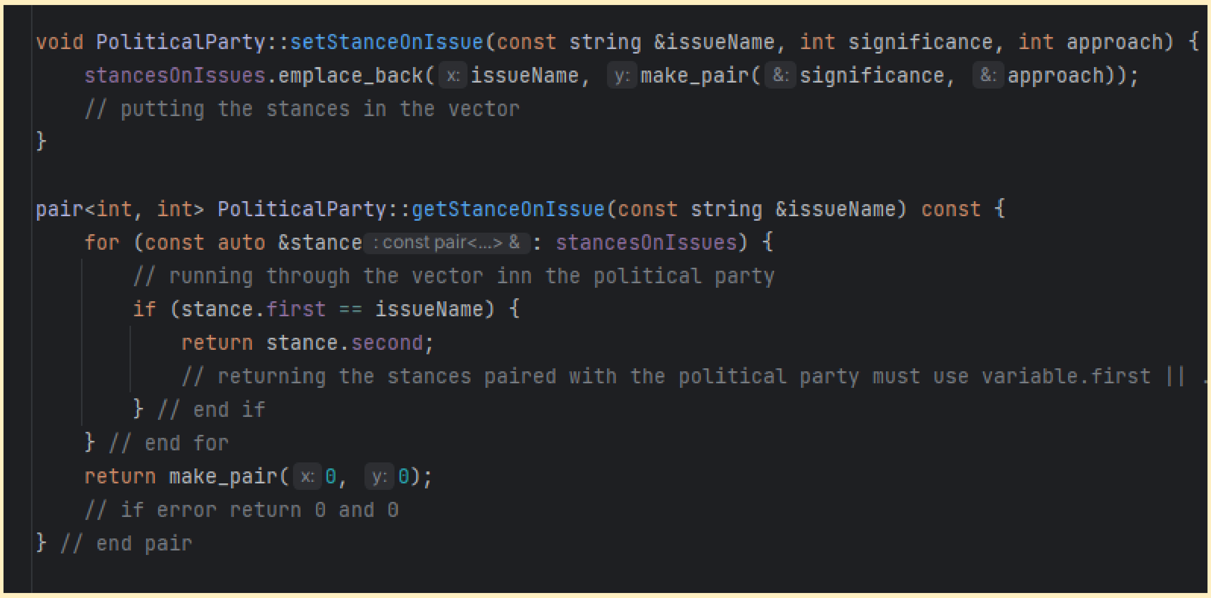
```
    void displayPoliticalParty();
    void displayStancesOnIssues() const;

    ~PoliticalParty();
};
```

This class is the main class for the political party. It has a few subclasses attached to it. It has 7 variables. I have used a lot of getters to get the information.

There is a vector with pair to get the issue name and pair it with 2 integers to store and get the points for the Average Stance Euclidean distance. I couldn’t think of a better way to pair them together. It works well at the end to get the stances though.



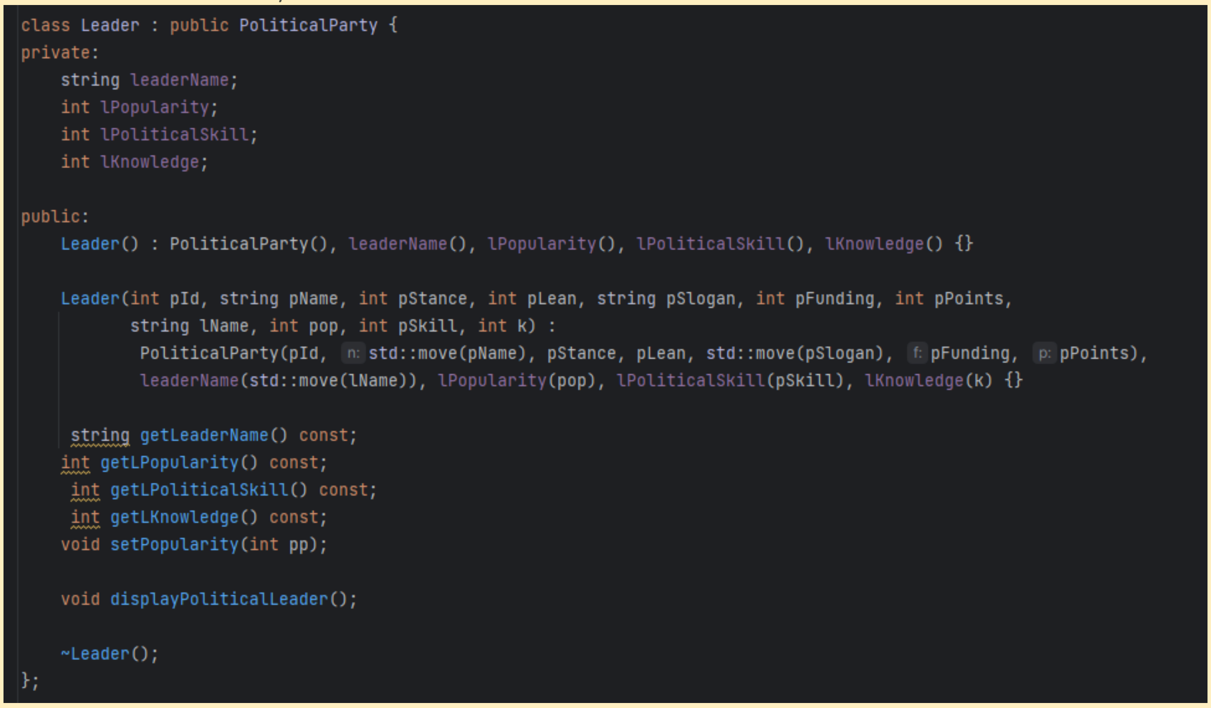
```
void PoliticalParty::displayPoliticalParty() {
    cout << "Party ID: " << partyID << endl;
    cout << "Party Name: " << partyName << endl;
    cout << "Party slogan: " << politicalSlogan << endl;
    cout << "Political Stance score: " << politicalStance << endl;
    cout << "Political Leaning (low score is Right wing, high is left wing): " << politicalLea
    cout << "Party funding: " << funding << endl;
    cout << "Party Points: " << pPoints << endl << endl;
}

void PoliticalParty::displayStancesOnIssues() const {
    cout << "Stances on Issues for Party: " << partyName << endl;
    int count = 0;
    for (auto &stance : const pair<...> & : stancesOnIssues) {
        count++;
        // using a count so you can keep track of the issues
        if (stance.second.first != 0 || stance.second.second != 0) {
            // only runs if there is a number in the vector pair
            cout << count << ". Issue: " << stance.first << endl;
            // the issue name
            cout << "Significance: " << stance.second.first << endl;
            cout << "Political stance: " << politicalStance << endl;
            cout << "Approach: " << stance.second.second << endl;
            cout << "-----------------------------" << endl;
            // displaying the ints in the vector in header.h
        } else {
            cout << "Party: " << partyName << " has no stance on " << stance.first << endl;
            // only displays if there is an error
        } // end else
```



```
void PoliticalParty::setStanceOnIssue(const string &issueName, int significance, int approach) {
    stancesOnIssues.emplace_back( x: issueName,  y: make_pair( &: significance,  &: approach));
    // putting the stances in the vector
}

pair<int, int> PoliticalParty::getStanceOnIssue(const string &issueName) const {
    for (const auto &stance : const pair<...> & : stancesOnIssues) {
        // running through the vector inn the political party
        if (stance.first == issueName) {
            return stance.second;
            // returning the stances paired with the political party must use variable.first || .
        } // end if
    } // end for
    return make_pair( x: 0,  y: 0);
    // if error return 0 and 0
} // end pair
```

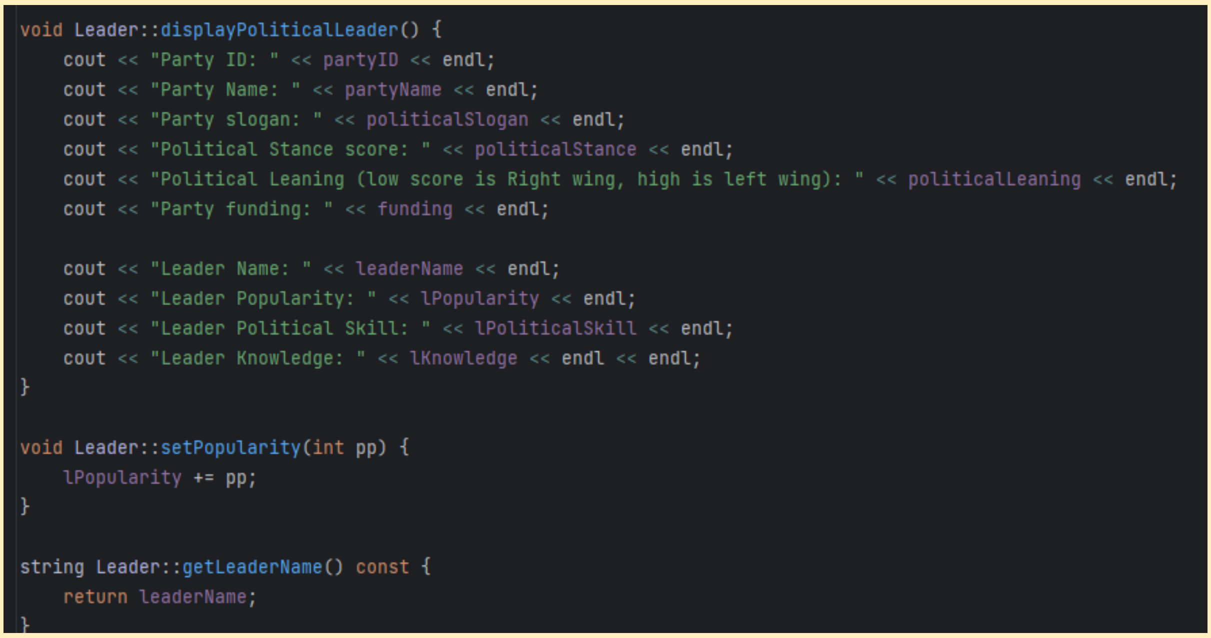
Getters and setters for the vector in the class

Leader : Political Party



```cpp
class Leader : public PoliticalParty {
private:
    string leaderName;
    int lPopularity;
    int lPoliticalSkill;
    int lKnowledge;

public:
    Leader() : PoliticalParty(), leaderName(), lPopularity(), lPoliticalSkill(), lKnowledge() {}

    Leader(int pId, string pName, int pStance, int pLean, string pSlogan, int pFunding, int pPoints,
           string lName, int pop, int pSkill, int k) :
            PoliticalParty(pId,  n: std::move(pName), pStance, pLean, std::move(pSlogan),  f: pFunding,  p: pPoints),
            leaderName(std::move(lName)), lPopularity(pop), lPoliticalSkill(pSkill), lKnowledge(k) {}

     string getLeaderName() const;
    int getLPopularity() const;
     int getLPoliticalSkill() const;
     int getLKnowledge() const;
    void setPopularity(int pp);

    void displayPoliticalLeader();

    ~Leader();
};
```

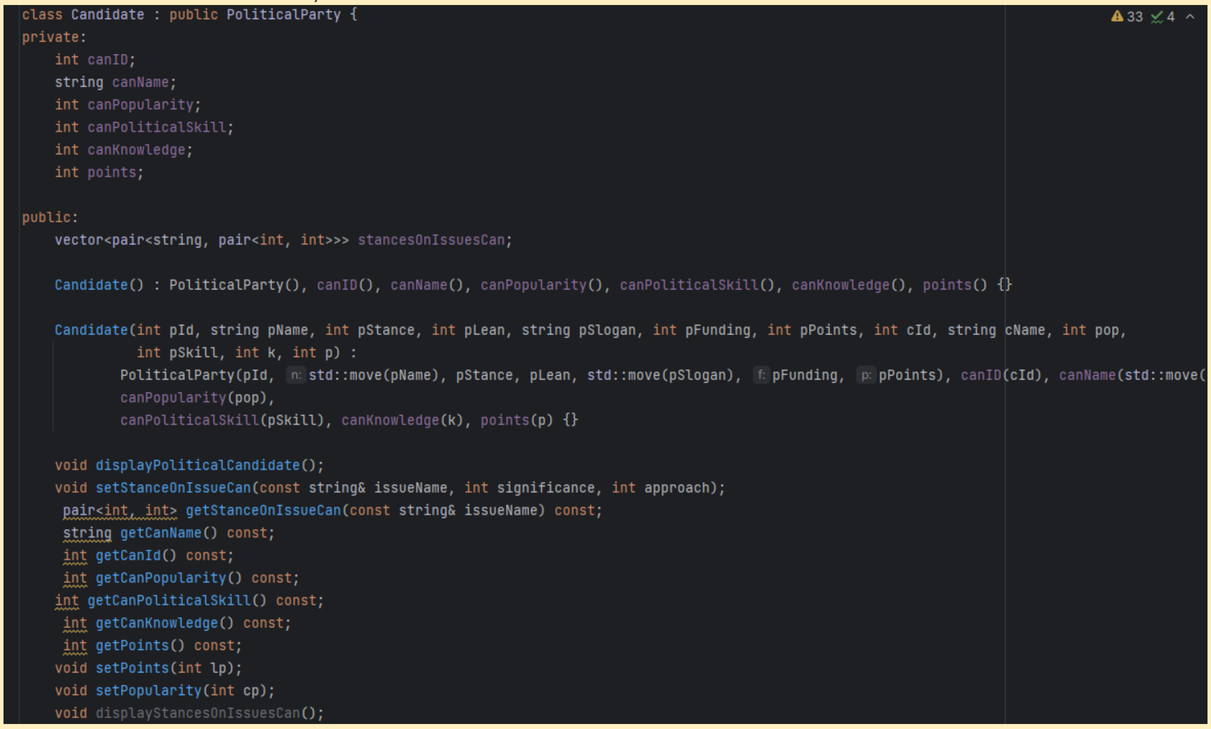
The leader is the first subclass of Political party. It uses variables from the Political party. It has a few getters and setters. And has a display method.



```cpp
void Leader::displayPoliticalLeader() {
    cout << "Party ID: " << partyID << endl;
    cout << "Party Name: " << partyName << endl;
    cout << "Party slogan: " << politicalSlogan << endl;
    cout << "Political Stance score: " << politicalStance << endl;
    cout << "Political Leaning (low score is Right wing, high is left wing): " << politicalLeaning << endl;
    cout << "Party funding: " << funding << endl;

    cout << "Leader Name: " << leaderName << endl;
    cout << "Leader Popularity: " << lPopularity << endl;
    cout << "Leader Political Skill: " << lPoliticalSkill << endl;
    cout << "Leader Knowledge: " << lKnowledge << endl << endl;
}

void Leader::setPopularity(int pp) {
    lPopularity += pp;
}

string Leader::getLeaderName() const {
    return leaderName;
}
```

I am using popularity += as I want to add or minus points further into the program

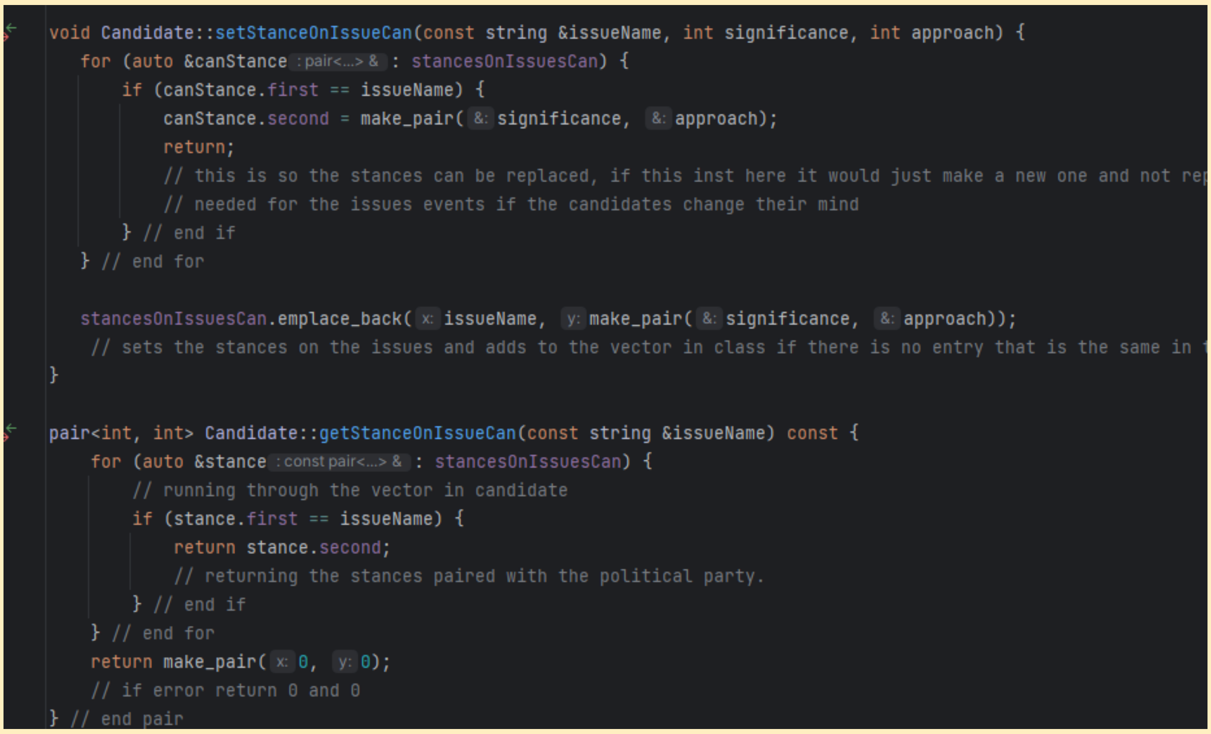
Candidate: Political Party



```cpp
class Candidate : public PoliticalParty {
private:
    int canID;
    string canName;
    int canPopularity;
    int canPoliticalSkill;
    int canKnowledge;
    int points;

public:
    vector<pair<string, pair<int, int>>> stancesOnIssuesCan;

    Candidate() : PoliticalParty(), canID(), canName(), canPopularity(), canPoliticalSkill(), canKnowledge(), points() {}

    Candidate(int pId, string pName, int pStance, int pLean, string pSlogan, int pFunding, int pPoints, int cId, string cName, int pop,
              int pSkill, int k, int p) :
            PoliticalParty(pId, n: std::move(pName), pStance, pLean, std::move(pSlogan), f: pFunding, p: pPoints), canID(cId), canName(std::move(
            canPopularity(pop),
            canPoliticalSkill(pSkill), canKnowledge(k), points(p) {}

    void displayPoliticalCandidate();
    void setStanceOnIssueCan(const string& issueName, int significance, int approach);
     pair<int, int> getStanceOnIssueCan(const string& issueName) const;
     string getCanName() const;
     int getCanId() const;
     int getCanPopularity() const;
    int getCanPoliticalSkill() const;
     int getCanKnowledge() const;
     int getPoints() const;
    void setPoints(int lp);
    void setPopularity(int cp);
    void displayStancesOnIssuesCan();
```

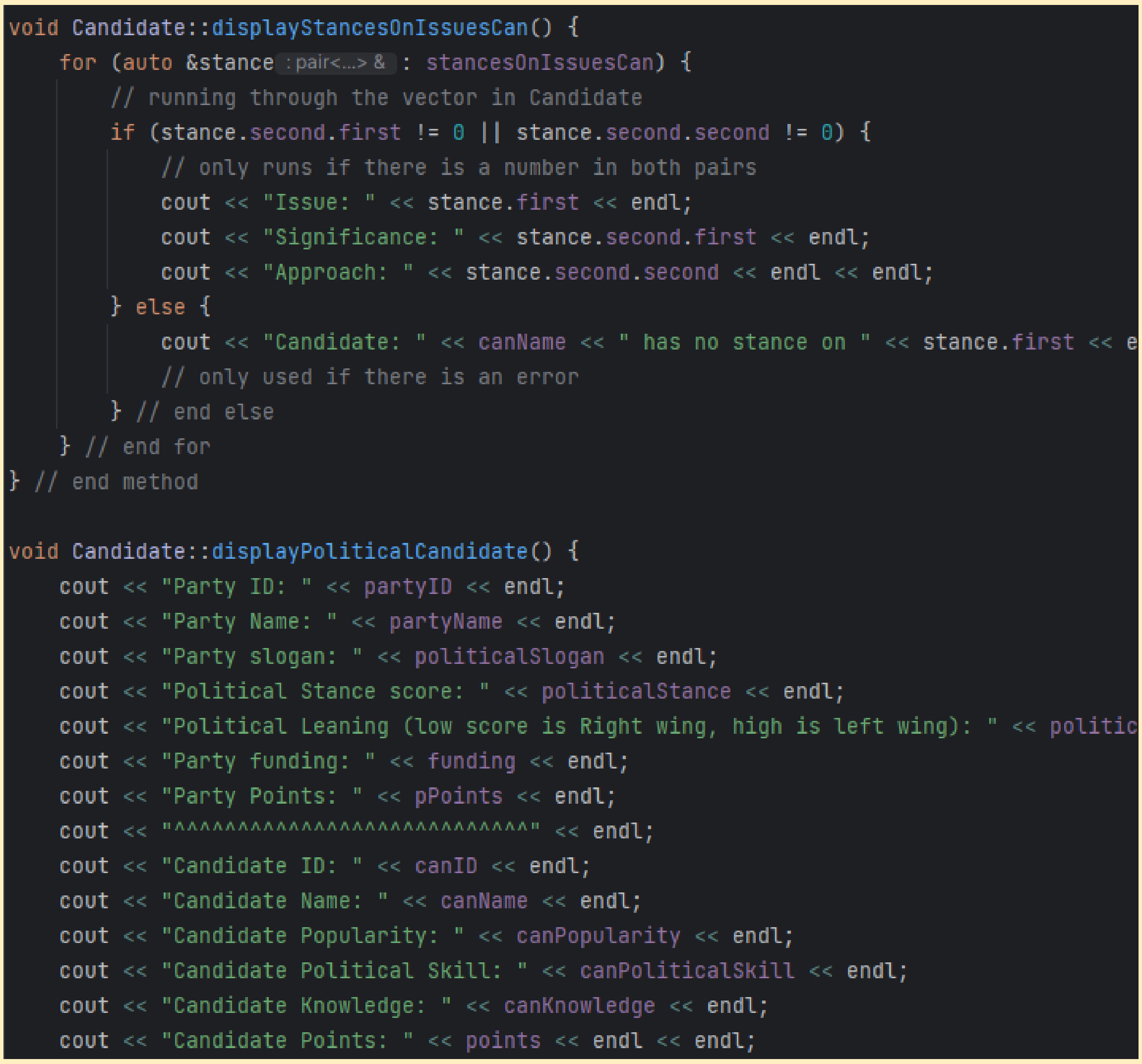
This class has many variables as it is one of the most important to the program and finding a winner. It is a subclass of the Political party. I have a vector in here to pair the issue significance and approach to get the stance for the voting method.

There is a de constructor underneath, but the screenshot would not reach it.



```cpp
void Candidate::setStanceOnIssueCan(const string &issueName, int significance, int approach) {
    for (auto &canStance :pair<...>& : stancesOnIssuesCan) {
        if (canStance.first == issueName) {
            canStance.second = make_pair( &: significance,  &: approach);
            return;
            // this is so the stances can be replaced, if this inst here it would just make a new one and not rep
            // needed for the issues events if the candidates change their mind
        } // end if
    } // end for

    stancesOnIssuesCan.emplace_back( x: issueName,  y: make_pair( &: significance,  &: approach));
     // sets the stances on the issues and adds to the vector in class if there is no entry that is the same in t
}

pair<int, int> Candidate::getStanceOnIssueCan(const string &issueName) const {
     for (auto &stance :const pair<...>& : stancesOnIssuesCan) {
         // running through the vector in candidate
         if (stance.first == issueName) {
             return stance.second;
             // returning the stances paired with the political party.
         } // end if
     } // end for
     return make_pair( x: 0,  y: 0);
     // if error return 0 and 0
} // end pair
```

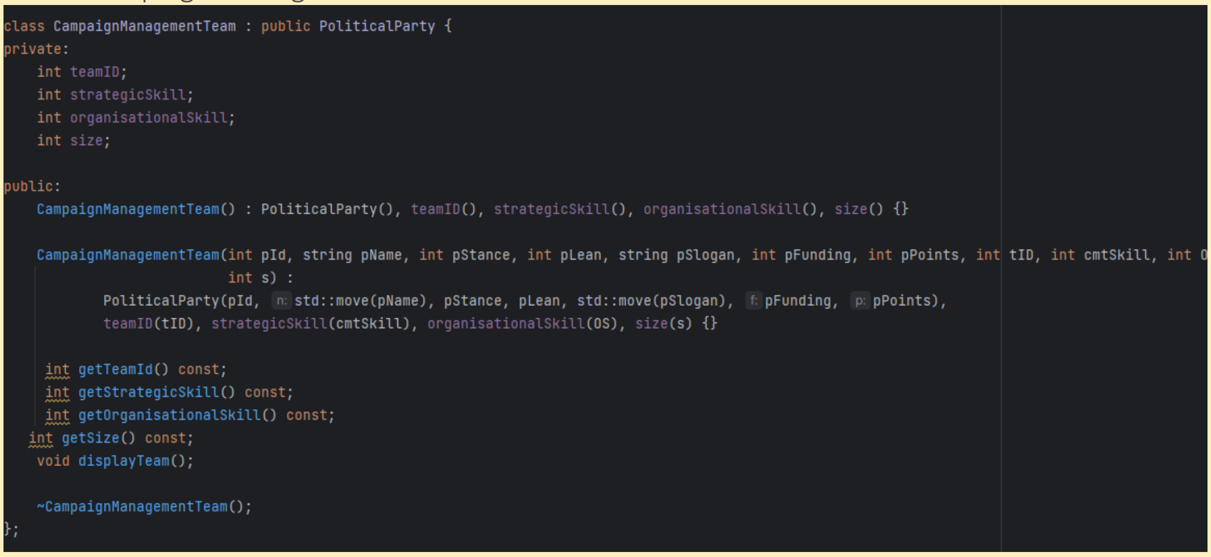
This is the only one that is different as it's the only one that needs to have their stance changed to due events. The only way to do that was to run a for loop to loop through the vector then return once its been replaced.



```cpp
void Candidate::displayStancesOnIssuesCan() {
    for (auto &stance : pair<...> & : stancesOnIssuesCan) {
        // running through the vector in Candidate
        if (stance.second.first != 0 || stance.second.second != 0) {
            // only runs if there is a number in both pairs
            cout << "Issue: " << stance.first << endl;
            cout << "Significance: " << stance.second.first << endl;
            cout << "Approach: " << stance.second.second << endl << endl;
        } else {
            cout << "Candidate: " << canName << " has no stance on " << stance.first << e
            // only used if there is an error
        } // end else
    } // end for
} // end method

void Candidate::displayPoliticalCandidate() {
    cout << "Party ID: " << partyID << endl;
    cout << "Party Name: " << partyName << endl;
    cout << "Party slogan: " << politicalSlogan << endl;
    cout << "Political Stance score: " << politicalStance << endl;
    cout << "Political Leaning (low score is Right wing, high is left wing): " << politic
    cout << "Party funding: " << funding << endl;
    cout << "Party Points: " << pPoints << endl;
    cout << "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^" << endl;
    cout << "Candidate ID: " << canID << endl;
    cout << "Candidate Name: " << canName << endl;
    cout << "Candidate Popularity: " << canPopularity << endl;
    cout << "Candidate Political Skill: " << canPoliticalSkill << endl;
    cout << "Candidate Knowledge: " << canKnowledge << endl;
    cout << "Candidate Points: " << points << endl << endl;
```

Displaying everything methods

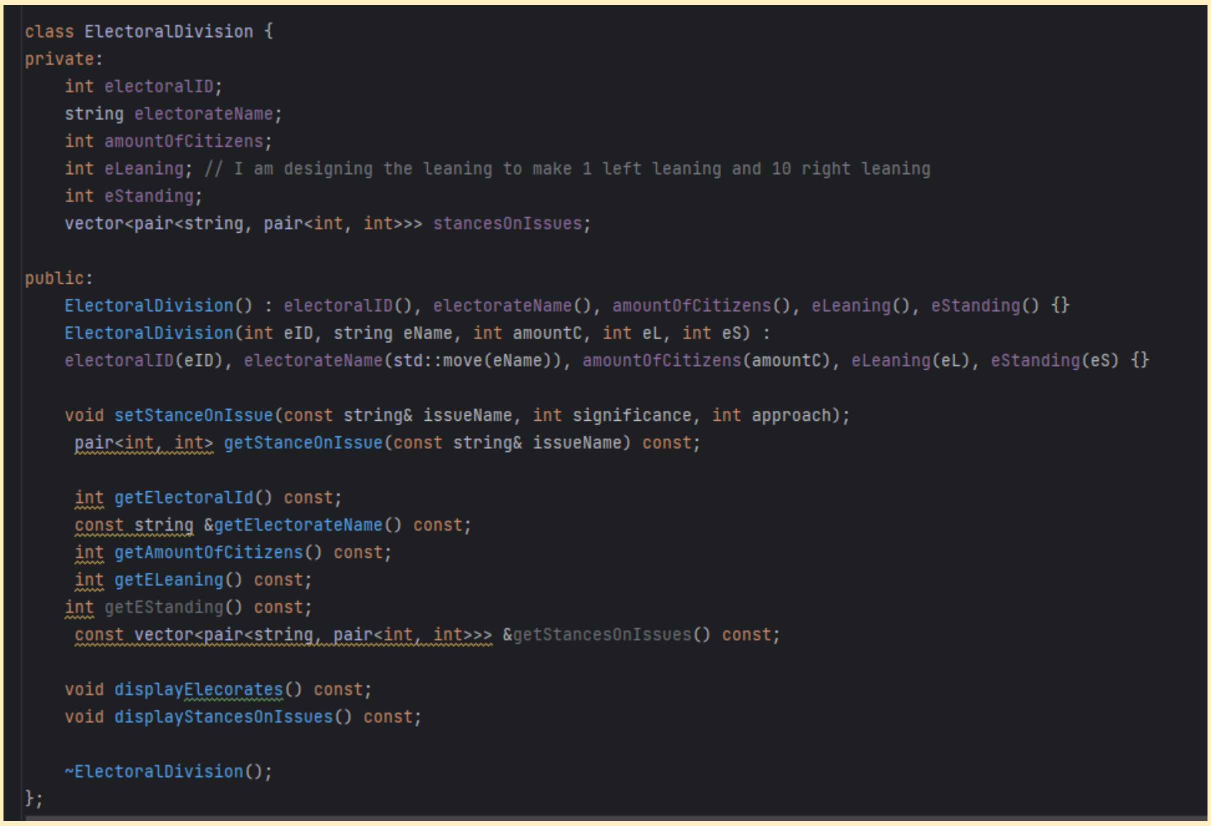
Class: Campaign management team



```cpp
class CampaignManagementTeam : public PoliticalParty {
private:
    int teamID;
    int strategicSkill;
    int organisationalSkill;
    int size;

public:
    CampaignManagementTeam() : PoliticalParty(), teamID(), strategicSkill(), organisationalSkill(), size() {}

    CampaignManagementTeam(int pId, string pName, int pStance, int pLean, string pSlogan, int pFunding, int pPoints, int tID, int cmtSkill, int O
                           int s) :
            PoliticalParty(pId, n: std::move(pName), pStance, pLean, std::move(pSlogan), f: pFunding, p: pPoints),
            teamID(tID), strategicSkill(cmtSkill), organisationalSkill(OS), size(s) {}

     int getTeamId() const;
     int getStrategicSkill() const;
     int getOrganisationalSkill() const;
   int getSize() const;
    void displayTeam();

    ~CampaignManagementTeam();
};
```

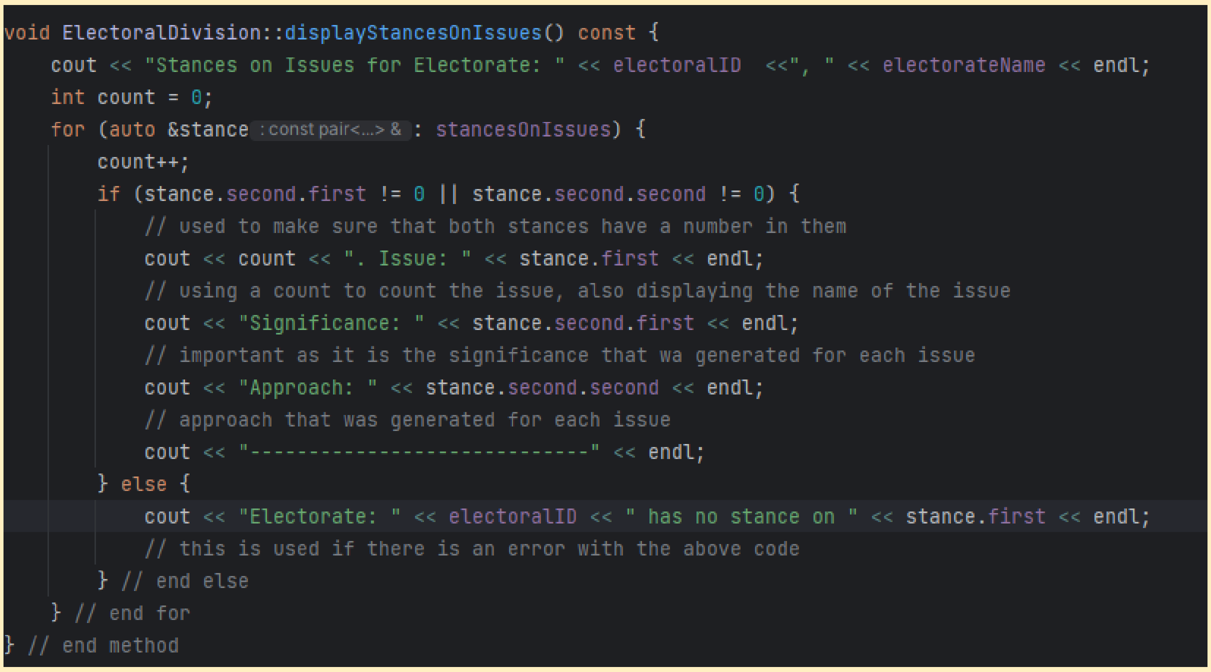
The class for CM Team, just basic getters and setters and a few variables. The are mostly used in the event stages

Class: Electoral Division

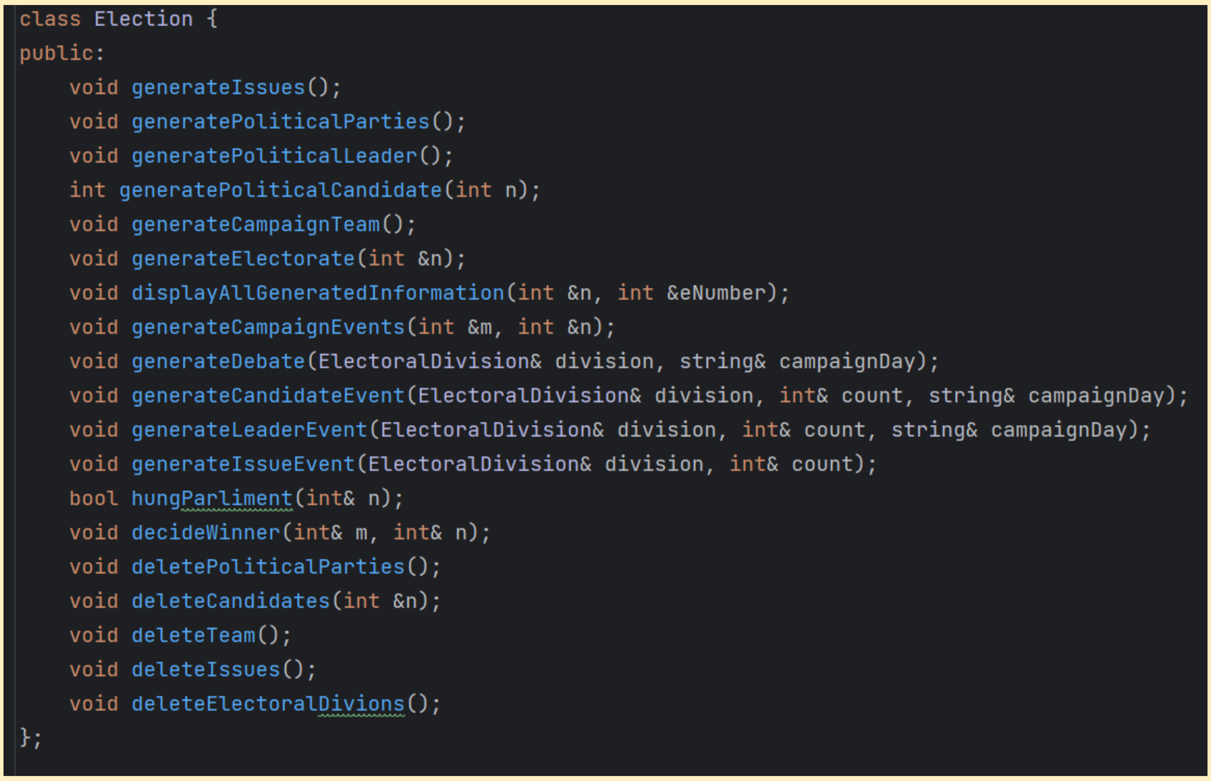


```cpp
class ElectoralDivision {
private:
    int electoralID;
    string electorateName;
    int amountOfCitizens;
    int eLeaning; // I am designing the leaning to make 1 left leaning and 10 right leaning
    int eStanding;
    vector<pair<string, pair<int, int>>> stancesOnIssues;

public:
    ElectoralDivision() : electoralID(), electorateName(), amountOfCitizens(), eLeaning(), eStanding() {}
    ElectoralDivision(int eID, string eName, int amountC, int eL, int eS) :
    electoralID(eID), electorateName(std::move(eName)), amountOfCitizens(amountC), eLeaning(eL), eStanding(eS) {}

    void setStanceOnIssue(const string& issueName, int significance, int approach);
     pair<int, int> getStanceOnIssue(const string& issueName) const;

     int getElectoralId() const;
     const string &getElectorateName() const;
     int getAmountOfCitizens() const;
     int getELeaning() const;
    int getEStanding() const;
     const vector<pair<string, pair<int, int>>> &getStancesOnIssues() const;

    void displayElecorates() const;
    void displayStancesOnIssues() const;

    ~ElectoralDivision();
};
```

Has a few variables and getters and setters. The vector is the exact same as the Candidate and political leaders. They also need to have a stance for the voting to compare with the Candidate.

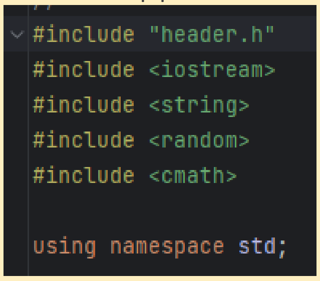


```cpp
void ElectoralDivision::displayStancesOnIssues() const {
    cout << "Stances on Issues for Electorate: " << electoralID  <<", " << electorateName << endl;
    int count = 0;
    for (auto &stance : const pair<...> & : stancesOnIssues) {
        count++;
        if (stance.second.first != 0 || stance.second.second != 0) {
            // used to make sure that both stances have a number in them
            cout << count << ". Issue: " << stance.first << endl;
            // using a count to count the issue, also displaying the name of the issue
            cout << "Significance: " << stance.second.first << endl;
            // important as it is the significance that wa generated for each issue
            cout << "Approach: " << stance.second.second << endl;
            // approach that was generated for each issue
            cout << "-----------------------------" << endl;
        } else {
            cout << "Electorate: " << electoralID << " has no stance on " << stance.first << endl;
            // this is used if there is an error with the above code
        } // end else
    } // end for
} // end method
```

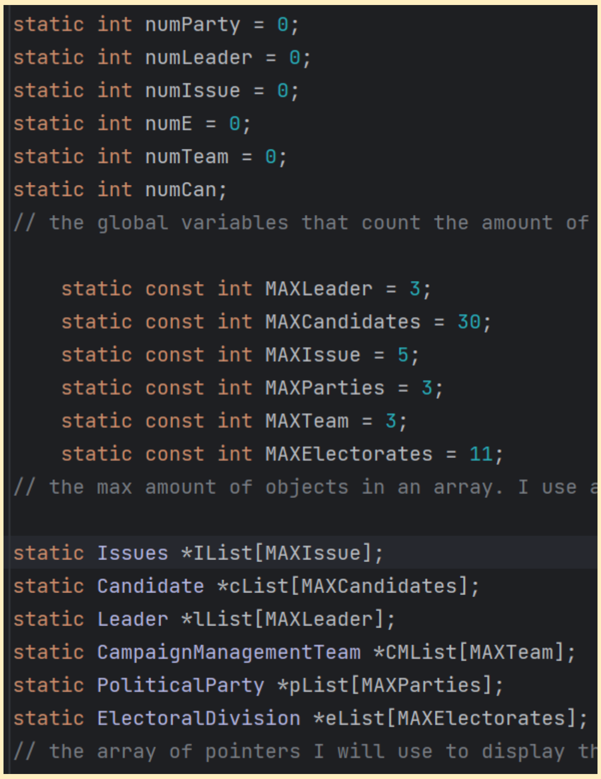

```cpp
class Election {
public:
    void generateIssues();
    void generatePoliticalParties();
    void generatePoliticalLeader();
    int generatePoliticalCandidate(int n);
    void generateCampaignTeam();
    void generateElectorate(int &n);
    void displayAllGeneratedInformation(int &n, int &eNumber);
    void generateCampaignEvents(int &m, int &n);
    void generateDebate(ElectoralDivision& division, string& campaignDay);
    void generateCandidateEvent(ElectoralDivision& division, int& count, string& campaignDay);
    void generateLeaderEvent(ElectoralDivision& division, int& count, string& campaignDay);
    void generateIssueEvent(ElectoralDivision& division, int& count);
    bool hungParliment(int& n);
    void decideWinner(int& m, int& n);
    void deletePoliticalParties();
    void deleteCandidates(int &n);
    void deleteTeam();
    void deleteIssues();
    void deleteElectoralDivions();
};
```

I made a class called election as I was told to use these methods in a global scope is not actually OO so I need to put them in a class. I called the class Election and it is in driver.cpp

Header.cpp Overview: Generating the information.


```cpp
#include "header.h"
#include <iostream>
#include <string>
#include <random>
#include <cmath>

using namespace std;
```

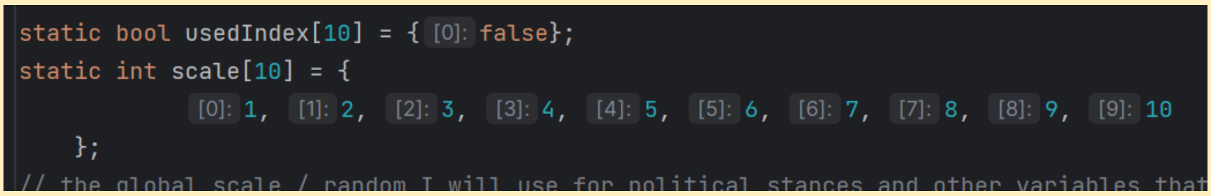
These are the libraries I used for header.cpp



```
static int numParty = 0;
static int numLeader = 0;
static int numIssue = 0;
static int numE = 0;
static int numTeam = 0;
static int numCan;
// the global variables that count the amount of

    static const int MAXLeader = 3;
    static const int MAXCandidates = 30;
    static const int MAXIssue = 5;
    static const int MAXParties = 3;
    static const int MAXTeam = 3;
    static const int MAXElectorates = 11;
// the max amount of objects in an array. I use a

static Issues *IList[MAXIssue];
static Candidate *cList[MAXCandidates];
static Leader *lList[MAXLeader];
static CampaignManagementTeam *CMList[MAXTeam];
static PoliticalParty *pList[MAXParties];
static ElectoralDivision *eList[MAXElectorates];
// the array of pointers I will use to display th
```

I used the num[insert here] as a global counter that I can use them in for loops to doge the dreaded exception bad_alloc. I decided to use arrays here instead of vectors because I wanted a challenge. It was more interesting to use fixed arrays as I had to be more careful. I also made them pointers because I didn't want my de-constructors to keep being called and ruin the output. It just meant I must be careful about memory leaks.

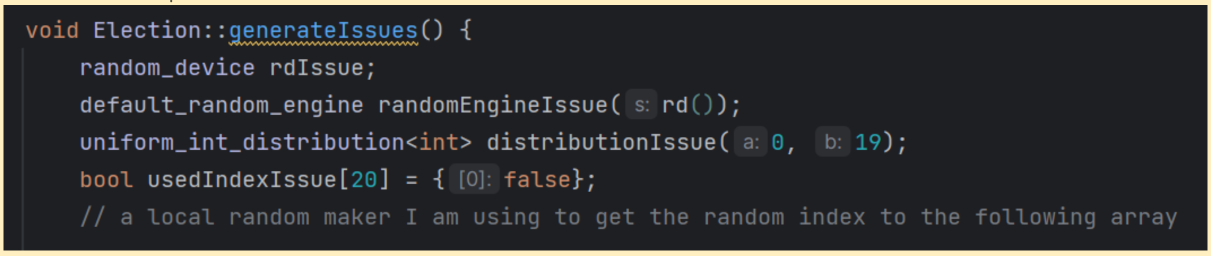
Everything is static as it must be loaded first.



```
static bool usedIndex[10] = { [0]: false};
static int scale[10] = {
            [0]: 1,  [1]: 2,  [2]: 3,  [3]: 4,  [4]: 5,  [5]: 6,  [6]: 7,  [7]: 8,  [8]: 9,  [9]: 10
    };
// the global scale / random I will use for political stances and other variables that
```

This was my global scale array. It was used to get a scale of 1-10. I used it mostly for the pop, approach, significance and other variables that I wanted to be compared. It saves me writing this many times over the course of generating the information.

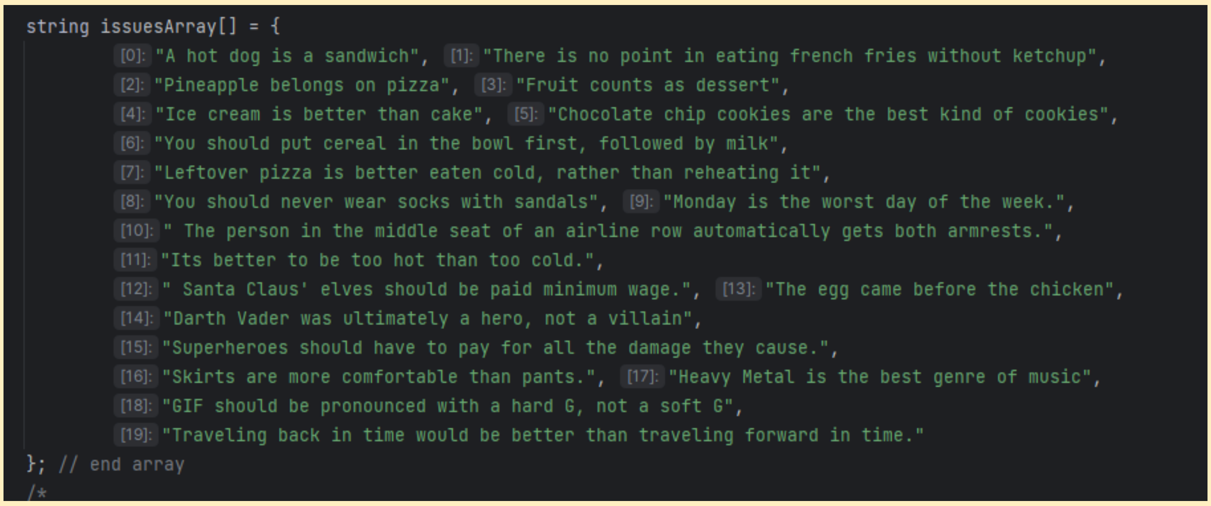
Issues Descriptions Point 2:



```
void Election::generateIssues() {
    random_device rdIssue;
    default_random_engine randomEngineIssue( s: rd());
    uniform_int_distribution<int> distributionIssue( a: 0,  b: 19);
    bool usedIndexIssue[20] = { [0]: false};
    // a local random maker I am using to get the random index to the following array
```

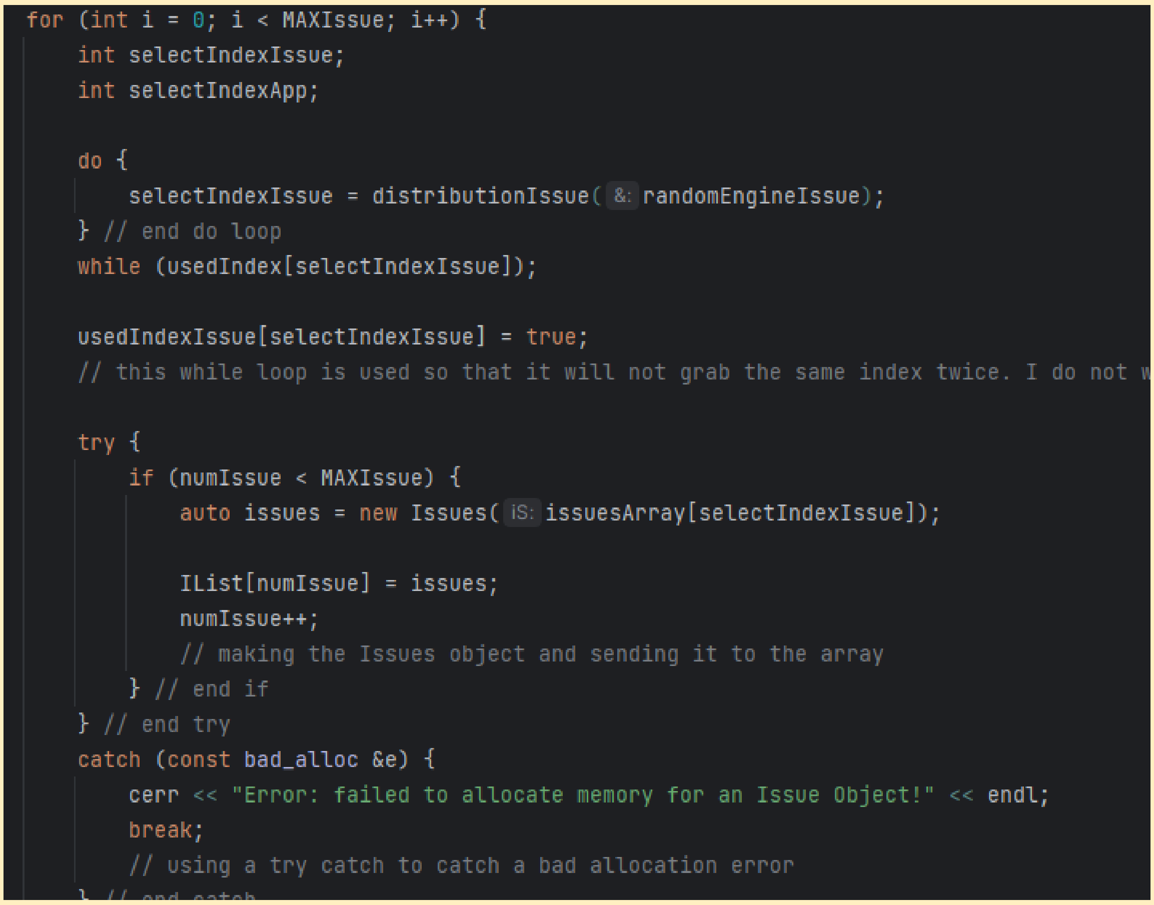
The first method for class Election

I went a different route. As I want my program completely random I made an array of issue names, and used a random index to pick one for each time the program is ran. I also didn't pick serious issues as who wants that?



```
string issuesArray[] = {
        [0]: "A hot dog is a sandwich",  [1]: "There is no point in eating french fries without ketchup",
        [2]: "Pineapple belongs on pizza",  [3]: "Fruit counts as dessert",
        [4]: "Ice cream is better than cake",  [5]: "Chocolate chip cookies are the best kind of cookies",
        [6]: "You should put cereal in the bowl first, followed by milk",
        [7]: "Leftover pizza is better eaten cold, rather than reheating it",
        [8]: "You should never wear socks with sandals",  [9]: "Monday is the worst day of the week.",
        [10]: " The person in the middle seat of an airline row automatically gets both armrests.",
        [11]: "Its better to be too hot than too cold.",
        [12]: " Santa Claus' elves should be paid minimum wage.",  [13]: "The egg came before the chicken",
        [14]: "Darth Vader was ultimately a hero, not a villain",
        [15]: "Superheroes should have to pay for all the damage they cause.",
        [16]: "Skirts are more comfortable than pants.",  [17]: "Heavy Metal is the best genre of music",
        [18]: "GIF should be pronounced with a hard G, not a soft G",
        [19]: "Traveling back in time would be better than traveling forward in time."
}; // end array
/*
```

I then used this to create a new object called issues

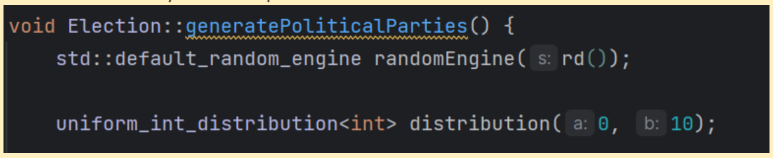


```
for (int i = 0; i < MAXIssue; i++) {
    int selectIndexIssue;
    int selectIndexApp;

    do {
        selectIndexIssue = distributionIssue( &: randomEngineIssue);
    } // end do loop
    while (usedIndex[selectIndexIssue]);

    usedIndexIssue[selectIndexIssue] = true;
    // this while loop is used so that it will not grab the same index twice. I do not w

    try {
        if (numIssue < MAXIssue) {
            auto issues = new Issues( iS: issuesArray[selectIndexIssue]);

            IList[numIssue] = issues;
            numIssue++;
            // making the Issues object and sending it to the array
        } // end if
    } // end try
    catch (const bad_alloc &e) {
        cerr << "Error: failed to allocate memory for an Issue Object!" << endl;
        break;
        // using a try catch to catch a bad allocation error
    } // end catch
```

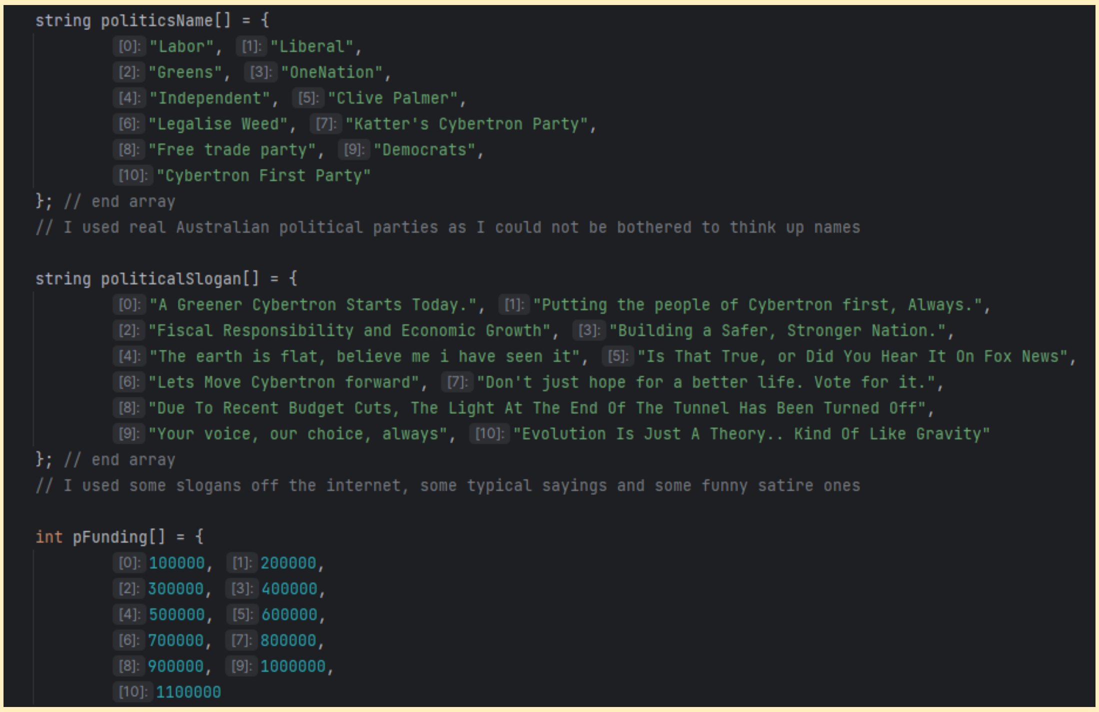
Note:

This is the basic way ill be creating all the classes. They wont differ except for a few ill explain in the coming pages.

Political Party Descriptions Point 3:



```
void Election::generatePoliticalParties() {
    std::default_random_engine randomEngine( s: rd());

    uniform_int_distribution<int> distribution( a: 0,  b: 10);
```



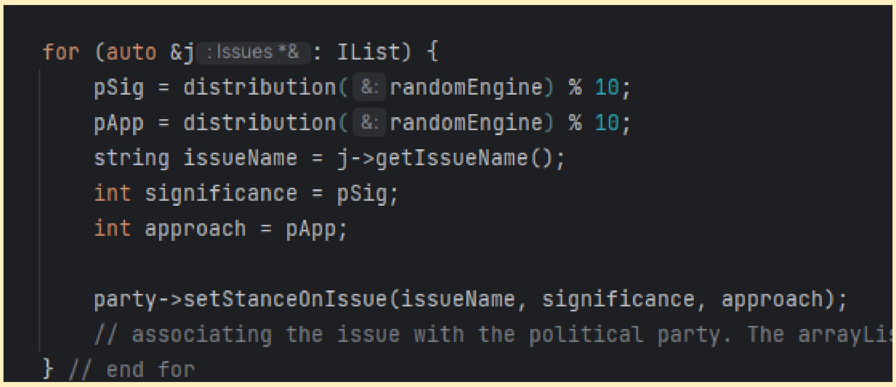
```
string politicsName[] = {
        [0]: "Labor",  [1]: "Liberal",
        [2]: "Greens",  [3]: "OneNation",
        [4]: "Independent",  [5]: "Clive Palmer",
        [6]: "Legalise Weed",  [7]: "Katter's Cybertron Party",
        [8]: "Free trade party",  [9]: "Democrats",
        [10]: "Cybertron First Party"
}; // end array
// I used real Australian political parties as I could not be bothered to think up names

string politicalSlogan[] = {
        [0]: "A Greener Cybertron Starts Today.",  [1]: "Putting the people of Cybertron first, Always.",
        [2]: "Fiscal Responsibility and Economic Growth",  [3]: "Building a Safer, Stronger Nation.",
        [4]: "The earth is flat, believe me i have seen it",  [5]: "Is That True, or Did You Hear It On Fox News",
        [6]: "Lets Move Cybertron forward",  [7]: "Don't just hope for a better life. Vote for it.",
        [8]: "Due To Recent Budget Cuts, The Light At The End Of The Tunnel Has Been Turned Off",
        [9]: "Your voice, our choice, always",  [10]: "Evolution Is Just A Theory.. Kind Of Like Gravity"
}; // end array
// I used some slogans off the internet, some typical sayings and some funny satire ones

int pFunding[] = {
        [0]: 100000,  [1]: 200000,
        [2]: 300000,  [3]: 400000,
        [4]: 500000,  [5]: 600000,
        [6]: 700000,  [7]: 800000,
        [8]: 900000,  [9]: 1000000,
        [10]: 1100000
```

All the names I have picked are from random Australian political parties. I think it would be funny seeing Clive palmers slogan things like "A greener future" etc.

I also made their slogans a mixture of real ones and satire.

These will all be picked via random uniform distribution. Every time will be different.

The way the Party is created will be the same way as Issues, using new and sending it off to the array. The only thing that will be different is the parties Approach and significance to each issue.

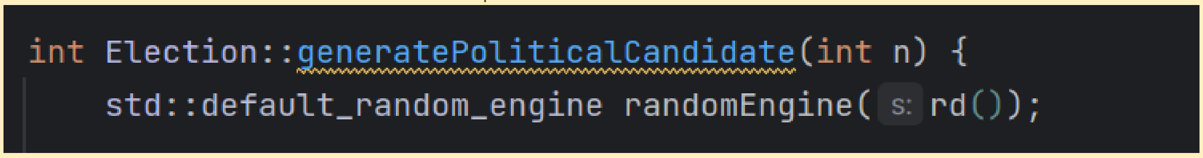


```
for (auto &j :Issues *& : IList) {
    pSig = distribution( &: randomEngine) % 10;
    pApp = distribution( &: randomEngine) % 10;
    string issueName = j->getIssueName();
    int significance = pSig;
    int approach = pApp;

    party->setStanceOnIssue(issueName, significance, approach);
    // associating the issue with the political party. The arrayLi
} // end for
```

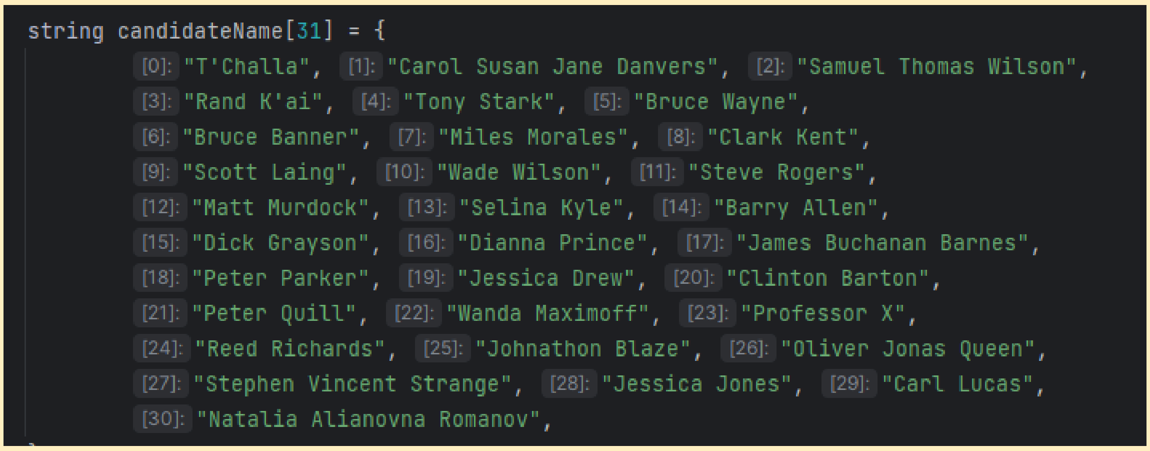
It is all based on the global scale of 1 -10 and will be completely random. It will then call the set stance method to add to the arraylist in the class.

These points will be influenced and have a chance to change in the Issues event. The candidates version of this will make the (x^2-x^1) in the stance calculation at voting day

Candidates and Electoral Generation point 4:

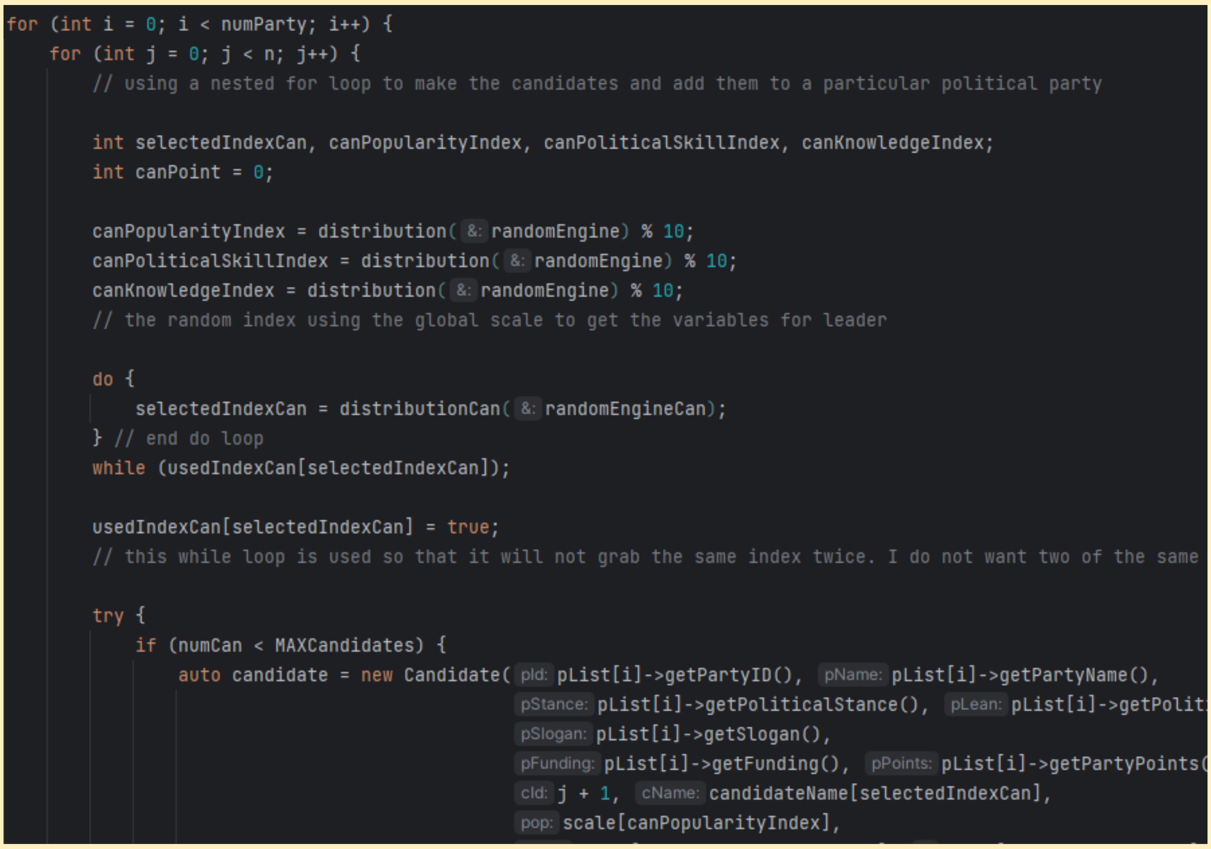


```cpp
int Election::generatePoliticalCandidate(int n) {
    std::default_random_engine randomEngine( s: rd());
```



```cpp
string candidateName[31] = {
        [0]: "T'Challa",  [1]: "Carol Susan Jane Danvers",  [2]: "Samuel Thomas Wilson",
        [3]: "Rand K'ai",  [4]: "Tony Stark",  [5]: "Bruce Wayne",
        [6]: "Bruce Banner",  [7]: "Miles Morales",  [8]: "Clark Kent",
        [9]: "Scott Laing",  [10]: "Wade Wilson",  [11]: "Steve Rogers",
        [12]: "Matt Murdock",  [13]: "Selina Kyle",  [14]: "Barry Allen",
        [15]: "Dick Grayson",  [16]: "Dianna Prince",  [17]: "James Buchanan Barnes",
        [18]: "Peter Parker",  [19]: "Jessica Drew",  [20]: "Clinton Barton",
        [21]: "Peter Quill",  [22]: "Wanda Maximoff",  [23]: "Professor X",
        [24]: "Reed Richards",  [25]: "Johnathon Blaze",  [26]: "Oliver Jonas Queen",
        [27]: "Stephen Vincent Strange",  [28]: "Jessica Jones",  [29]: "Carl Lucas",
        [30]: "Natalia Alianovna Romanov",
```

So I could not think of 30 names to create so I just decided to use names of superheroes from both the DC and marvel universes.



```cpp
for (int i = 0; i < numParty; i++) {
    for (int j = 0; j < n; j++) {
        // using a nested for loop to make the candidates and add them to a particular political party

        int selectedIndexCan, canPopularityIndex, canPoliticalSkillIndex, canKnowledgeIndex;
        int canPoint = 0;

        canPopularityIndex = distribution( &: randomEngine) % 10;
        canPoliticalSkillIndex = distribution( &: randomEngine) % 10;
        canKnowledgeIndex = distribution( &: randomEngine) % 10;
        // the random index using the global scale to get the variables for leader

        do {
            selectedIndexCan = distributionCan( &: randomEngineCan);
        } // end do loop
        while (usedIndexCan[selectedIndexCan]);

        usedIndexCan[selectedIndexCan] = true;
        // this while loop is used so that it will not grab the same index twice. I do not want two of the same

        try {
            if (numCan < MAXCandidates) {
                auto candidate = new Candidate( pId: pList[i]->getPartyID(),  pName: pList[i]->getPartyName(),
                                                pStance: pList[i]->getPoliticalStance(),  pLean: pList[i]->getPolit
                                                pSlogan: pList[i]->getSlogan(),
                                                pFunding: pList[i]->getFunding(),  pPoints: pList[i]->getPartyPoints(
                                                cId: j + 1,  cName: candidateName[selectedIndexCan],
                                                pop: scale[canPopularityIndex],
```

I used a nested for loop to loop through the parties and create the candidates, n represents the amount of candidates each party has as it represents the electorates.

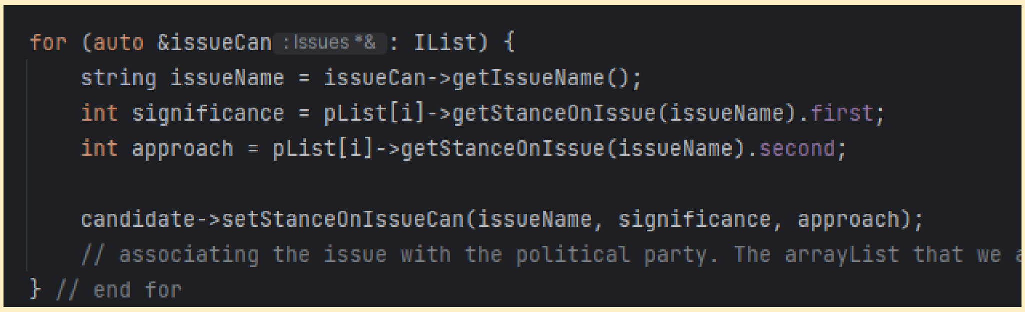
The cID is neat as it connects each candidate to the electorate, ill be using that a lot in the events bit.

All their variables are randomly chosen via uniform_distribution.

The candidates' variable popularity will influence voting score, as the person with the highest will get a few points. Their pop can be changed in the events, whether for the positive or the negative. This is one of the most important variables

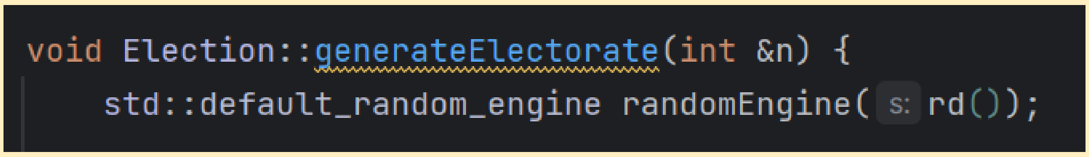
The candidates, skill and knowledge, and leaning are all factors that will take effect in the events. These factors will all add up or affect the overall score in the events that will change their popularity. It will be explained in detail in the event section.

AS with one of the points the candidates must have the political party's stance at the start. I did this using the getter for the vector in the class.

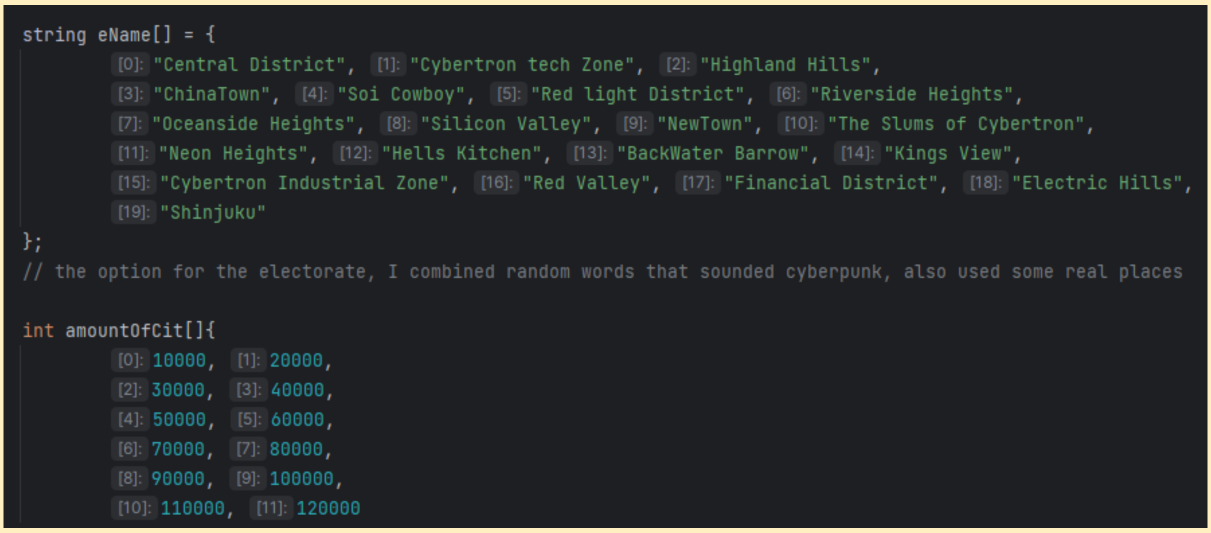


```
for (auto &issueCan : Issues *& : IList) {
    string issueName = issueCan->getIssueName();
    int significance = pList[i]->getStanceOnIssue(issueName).first;
    int approach = pList[i]->getStanceOnIssue(issueName).second;

    candidate->setStanceOnIssueCan(issueName, significance, approach);
    // associating the issue with the political party. The arrayList that we
} // end for
```

End of candidate



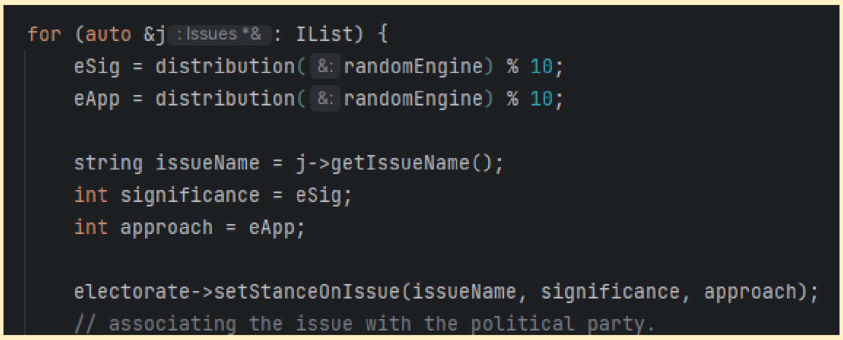
```
void Election::generateElectorate(int &n) {
    std::default_random_engine randomEngine( s: rd());
```



```
string eName[] = {
        [0]: "Central District",  [1]: "Cybertron tech Zone",  [2]: "Highland Hills",
        [3]: "ChinaTown",  [4]: "Soi Cowboy",  [5]: "Red light District",  [6]: "Riverside Heights",
        [7]: "Oceanside Heights",  [8]: "Silicon Valley",  [9]: "NewTown",  [10]: "The Slums of Cybertron",
        [11]: "Neon Heights",  [12]: "Hells Kitchen",  [13]: "BackWater Barrow",  [14]: "Kings View",
        [15]: "Cybertron Industrial Zone",  [16]: "Red Valley",  [17]: "Financial District",  [18]: "Electric Hills",
        [19]: "Shinjuku"
};
// the option for the electorate, I combined random words that sounded cyberpunk, also used some real places

int amountOfCit[]{
        [0]: 10000,  [1]: 20000,
        [2]: 30000,  [3]: 40000,
        [4]: 50000,  [5]: 60000,
        [6]: 70000,  [7]: 80000,
        [8]: 90000,  [9]: 100000,
        [10]: 110000,  [11]: 120000
```

I came up with random names and mixed in a few real ones that sounded cyberpunk. I also figured I needed to add in the amount of people in each electorate. This is one of the main factors in candidate and leader events as they have to get over a certain amount of citizens to attend their event for it to be a success or failure.

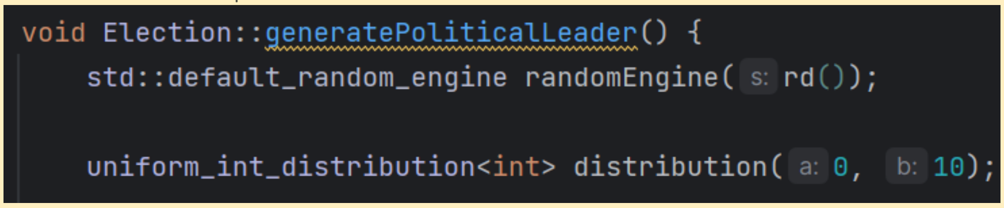
Generating the info is the exact same as the previous ones.



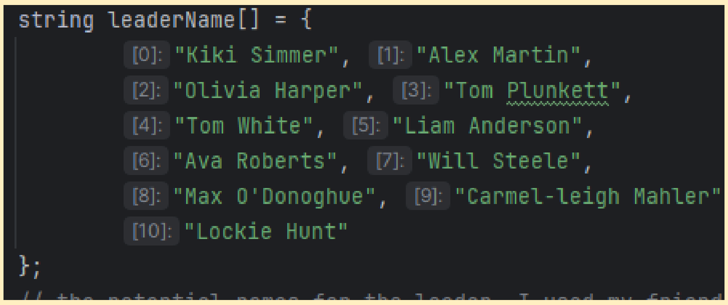
```
for (auto &j :Issues *& : IList) {
    eSig = distribution( &: randomEngine) % 10;
    eApp = distribution( &: randomEngine) % 10;

    string issueName = j->getIssueName();
    int significance = eSig;
    int approach = eApp;

    electorate->setStanceOnIssue(issueName, significance, approach);
    // associating the issue with the political party.
```

For the stances I also used a random scale of 1-10. I want them all to be consistent of a score between those numbers. It makes it easier in the end

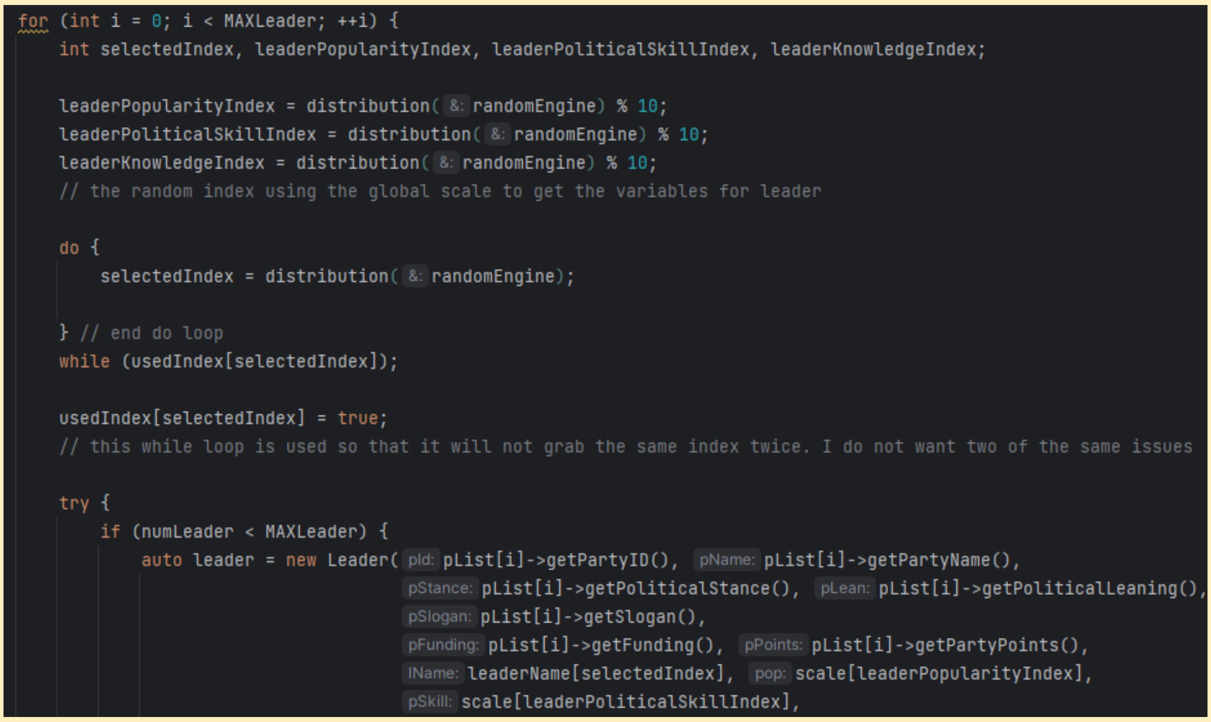
Leader and Team point 5:



```
void Election::generatePoliticalLeader() {
    std::default_random_engine randomEngine( s: rd());

    uniform_int_distribution<int> distribution( a: 0,  b: 10);
```



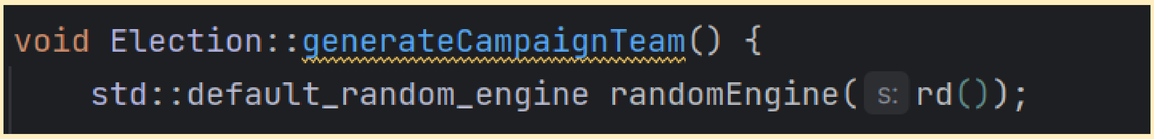
```
string leaderName[] = {
        [0]: "Kiki Simmer",  [1]: "Alex Martin",
        [2]: "Olivia Harper",  [3]: "Tom Plunkett",
        [4]: "Tom White",  [5]: "Liam Anderson",
        [6]: "Ava Roberts",  [7]: "Will Steele",
        [8]: "Max O'Donoghue",  [9]: "Carmel-leigh Mahler"
        [10]: "Lockie Hunt"
};
// the potential names for the leader. I used my friend
```



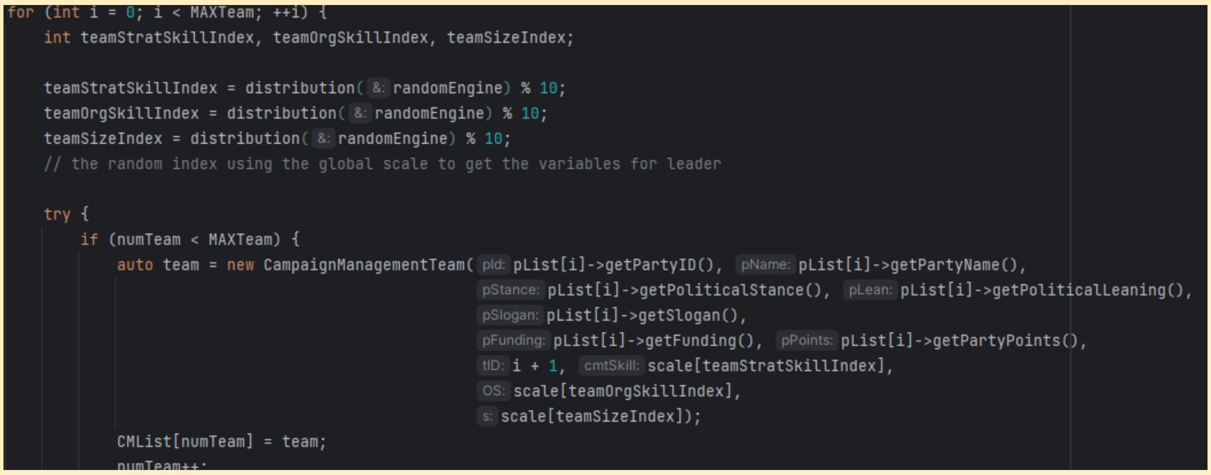
```
for (int i = 0; i < MAXLeader; ++i) {
    int selectedIndex, leaderPopularityIndex, leaderPoliticalSkillIndex, leaderKnowledgeIndex;

    leaderPopularityIndex = distribution( &: randomEngine) % 10;
    leaderPoliticalSkillIndex = distribution( &: randomEngine) % 10;
    leaderKnowledgeIndex = distribution( &: randomEngine) % 10;
    // the random index using the global scale to get the variables for leader

    do {
        selectedIndex = distribution( &: randomEngine);

    } // end do loop
    while (usedIndex[selectedIndex]);

    usedIndex[selectedIndex] = true;
    // this while loop is used so that it will not grab the same index twice. I do not want two of the same issues

    try {
        if (numLeader < MAXLeader) {
            auto leader = new Leader( pId: pList[i]->getPartyID(),  pName: pList[i]->getPartyName(),
                                      pStance: pList[i]->getPoliticalStance(),  pLean: pList[i]->getPoliticalLeaning(),
                                      pSlogan: pList[i]->getSlogan(),
                                      pFunding: pList[i]->getFunding(),  pPoints: pList[i]->getPartyPoints(),
                                      lName: leaderName[selectedIndex],  pop: scale[leaderPopularityIndex],
                                      pSkill: scale[leaderPoliticalSkillIndex],
```

I have decided to name the leaders after my friends. Their variables will use the global scale and randomness to decide their variables.

End of leader



```
void Election::generateCampaignTeam() {
    std::default_random_engine randomEngine( s: rd());
```

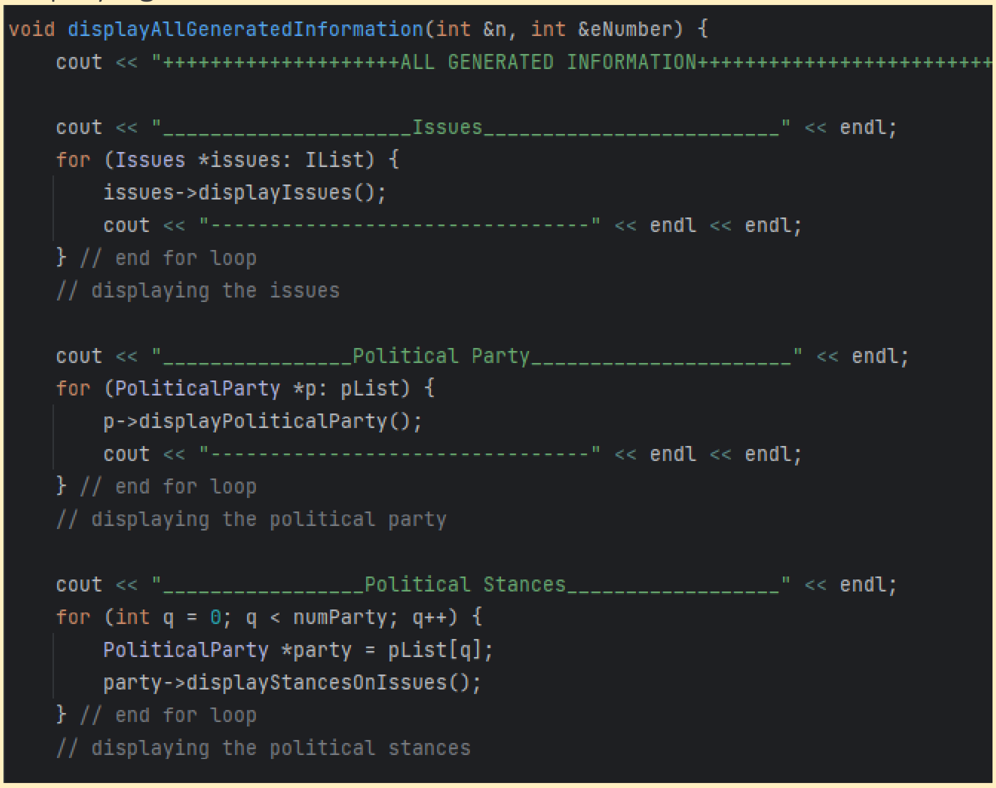


```
for (int i = 0; i < MAXTeam; ++i) {
    int teamStratSkillIndex, teamOrgSkillIndex, teamSizeIndex;

    teamStratSkillIndex = distribution( &: randomEngine) % 10;
    teamOrgSkillIndex = distribution( &: randomEngine) % 10;
    teamSizeIndex = distribution( &: randomEngine) % 10;
    // the random index using the global scale to get the variables for leader

    try {
        if (numTeam < MAXTeam) {
            auto team = new CampaignManagementTeam( pId: pList[i]->getPartyID(),  pName: pList[i]->getPartyName(),
                                                    pStance: pList[i]->getPoliticalStance(),  pLean: pList[i]->getPoliticalLeaning(),
                                                    pSlogan: pList[i]->getSlogan(),
                                                    pFunding: pList[i]->getFunding(),  pPoints: pList[i]->getPartyPoints(),
                                                    tID: i + 1,  cmtSkill: scale[teamStratSkillIndex],
                                                    OS: scale[teamOrgSkillIndex],
                                                    s: scale[teamSizeIndex]);

            CMList[numTeam] = team;
            numTeam++;
```

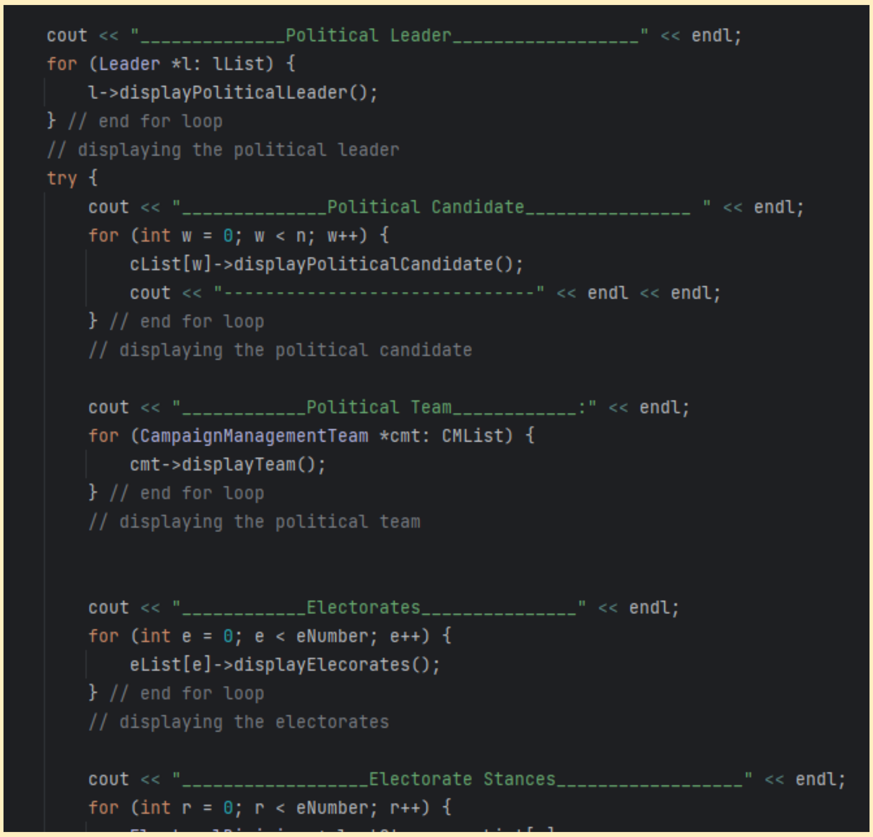
The campaign team is has all random variables, except for their id, which is just the index from the for loop to create them.

Their variables strategic skill, organizational skill and size will come into play in the events section, particularly the debate, and candidate and leader ones. The high in all of these the better

Displaying the information



```
void displayAllGeneratedInformation(int &n, int &eNumber) {
    cout << "++++++++++++++++++++ALL GENERATED INFORMATION+++++++++++++++++++++++++

    cout << "_____________________Issues_________________________" << endl;
    for (Issues *issues: IList) {
        issues->displayIssues();
        cout << "---------------------------------" << endl << endl;
    } // end for loop
    // displaying the issues

    cout << "________________Political Party______________________" << endl;
    for (PoliticalParty *p: pList) {
        p->displayPoliticalParty();
        cout << "---------------------------------" << endl << endl;
    } // end for loop
    // displaying the political party

    cout << "_________________Political Stances_________________" << endl;
    for (int q = 0; q < numParty; q++) {
        PoliticalParty *party = pList[q];
        party->displayStancesOnIssues();
    } // end for loop
    // displaying the political stances
```

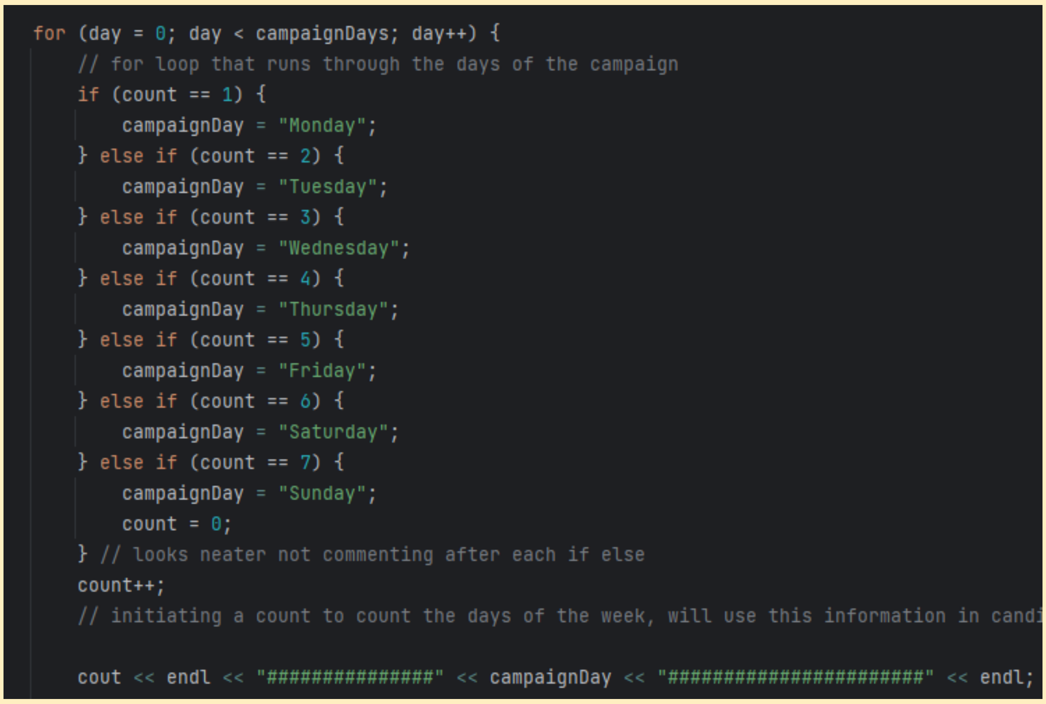


```
    cout << "_____________Political Leader________________" << endl;
    for (Leader *l: lList) {
        l->displayPoliticalLeader();
    } // end for loop
    // displaying the political leader
    try {
        cout << "_____________Political Candidate_______________ " << endl;
        for (int w = 0; w < n; w++) {
            cList[w]->displayPoliticalCandidate();
            cout << "------------------------------" << endl << endl;
        } // end for loop
        // displaying the political candidate

        cout << "___________Political Team___________:" << endl;
        for (CampaignManagementTeam *cmt: CMList) {
            cmt->displayTeam();
        } // end for loop
        // displaying the political team


        cout << "___________Electorates______________" << endl;
        for (int e = 0; e < eNumber; e++) {
            eList[e]->displayElecorates();
        } // end for loop
        // displaying the electorates

        cout << "__________________Electorate Stances__________________" << endl;
        for (int r = 0; r < eNumber; r++) {
```

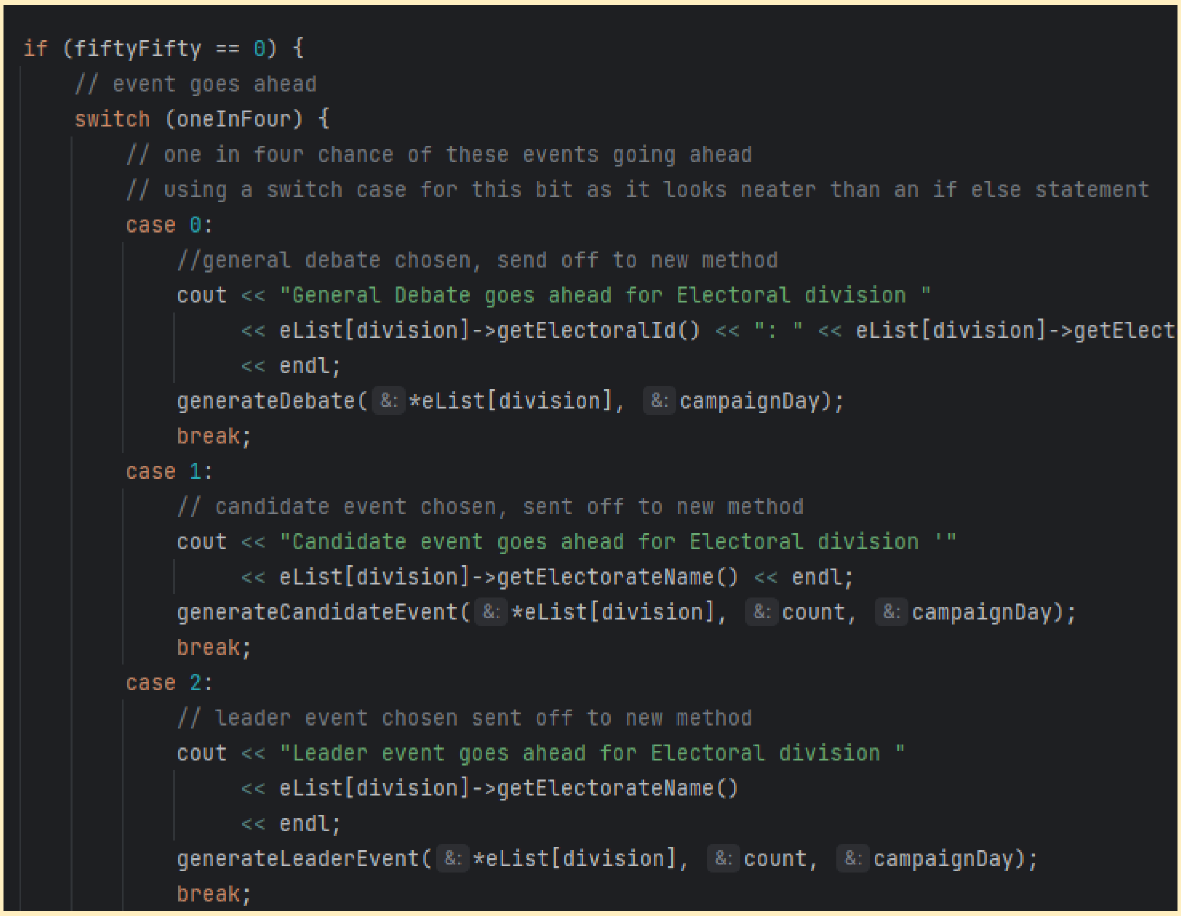
The report was just a bunch of for loops that cycle through each array and display to the terminal.

Campaign events Point 6:

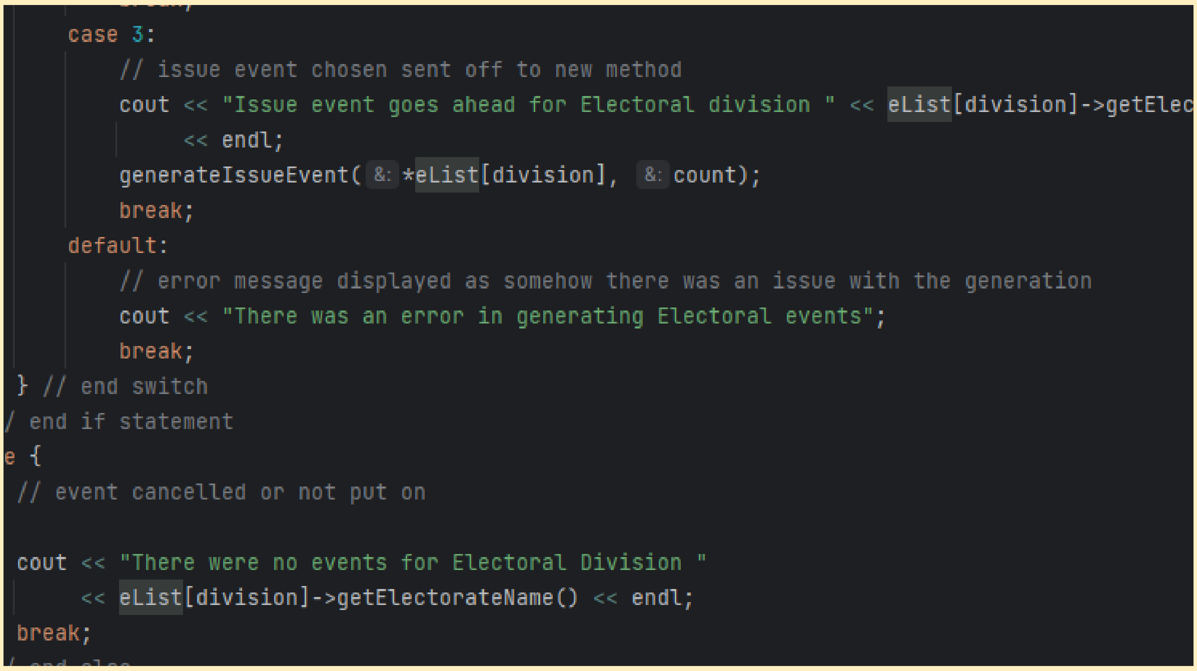
I decided to use the days of the week to count the days, it adds to the events of leader and can



```
for (day = 0; day < campaignDays; day++) {
    // for loop that runs through the days of the campaign
    if (count == 1) {
        campaignDay = "Monday";
    } else if (count == 2) {
        campaignDay = "Tuesday";
    } else if (count == 3) {
        campaignDay = "Wednesday";
    } else if (count == 4) {
        campaignDay = "Thursday";
    } else if (count == 5) {
        campaignDay = "Friday";
    } else if (count == 6) {
        campaignDay = "Saturday";
    } else if (count == 7) {
        campaignDay = "Sunday";
        count = 0;
    } // looks neater not commenting after each if else
    count++;
    // initiating a count to count the days of the week, will use this information in candi

    cout << endl << "###############" << campaignDay << "#######################" << endl;
```



```
if (fiftyFifty == 0) {
    // event goes ahead
    switch (oneInFour) {
        // one in four chance of these events going ahead
        // using a switch case for this bit as it looks neater than an if else statement
        case 0:
            //general debate chosen, send off to new method
            cout << "General Debate goes ahead for Electoral division "
                 << eList[division]->getElectoralId() << ": " << eList[division]->getElect
                 << endl;
            generateDebate( &: *eList[division],  &: campaignDay);
            break;
        case 1:
            // candidate event chosen, sent off to new method
            cout << "Candidate event goes ahead for Electoral division '"
                 << eList[division]->getElectorateName() << endl;
            generateCandidateEvent( &: *eList[division],  &: count,  &: campaignDay);
            break;
        case 2:
            // leader event chosen sent off to new method
            cout << "Leader event goes ahead for Electoral division "
                 << eList[division]->getElectorateName()
                 << endl;
            generateLeaderEvent( &: *eList[division],  &: count,  &: campaignDay);
            break;
```

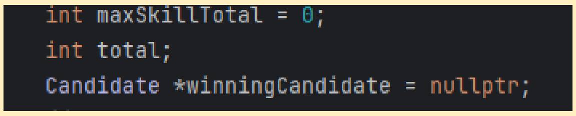


```
        case 3:
            // issue event chosen sent off to new method
            cout << "Issue event goes ahead for Electoral division " << eList[division]->getElec
                 << endl;
            generateIssueEvent( &: *eList[division],  &: count);
            break;
        default:
            // error message displayed as somehow there was an issue with the generation
            cout << "There was an error in generating Electoral events";
            break;
 } // end switch
/ end if statement
e {
 // event cancelled or not put on

 cout << "There were no events for Electoral Division "
      << eList[division]->getElectorateName() << endl;
 break;
```

This part of the code is where a random device will chose either 0, or 1. This is for whether there is an event that goes ahead or not. Then it will go to a switch statement that is using another random device to generate a 0 – 3 that will generate an event and send it off to the event method. I also pass the division object and the day. If it hits a default then there is an error and the code needs to be fixed.

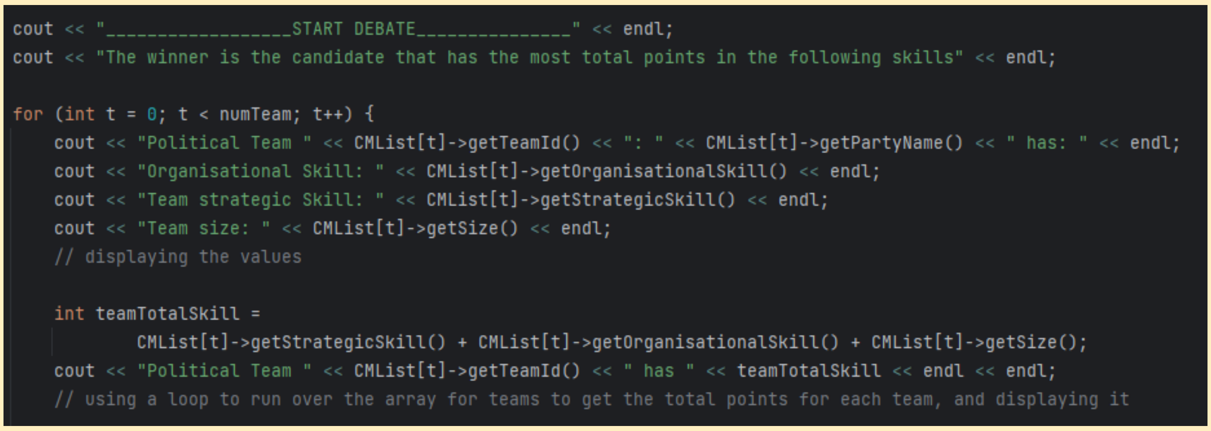
The great debate

I Designed this event to get all the variables from Candidate, and CM team and some from Leader . I decided the winner should be the one that has the largest total of all the skills.



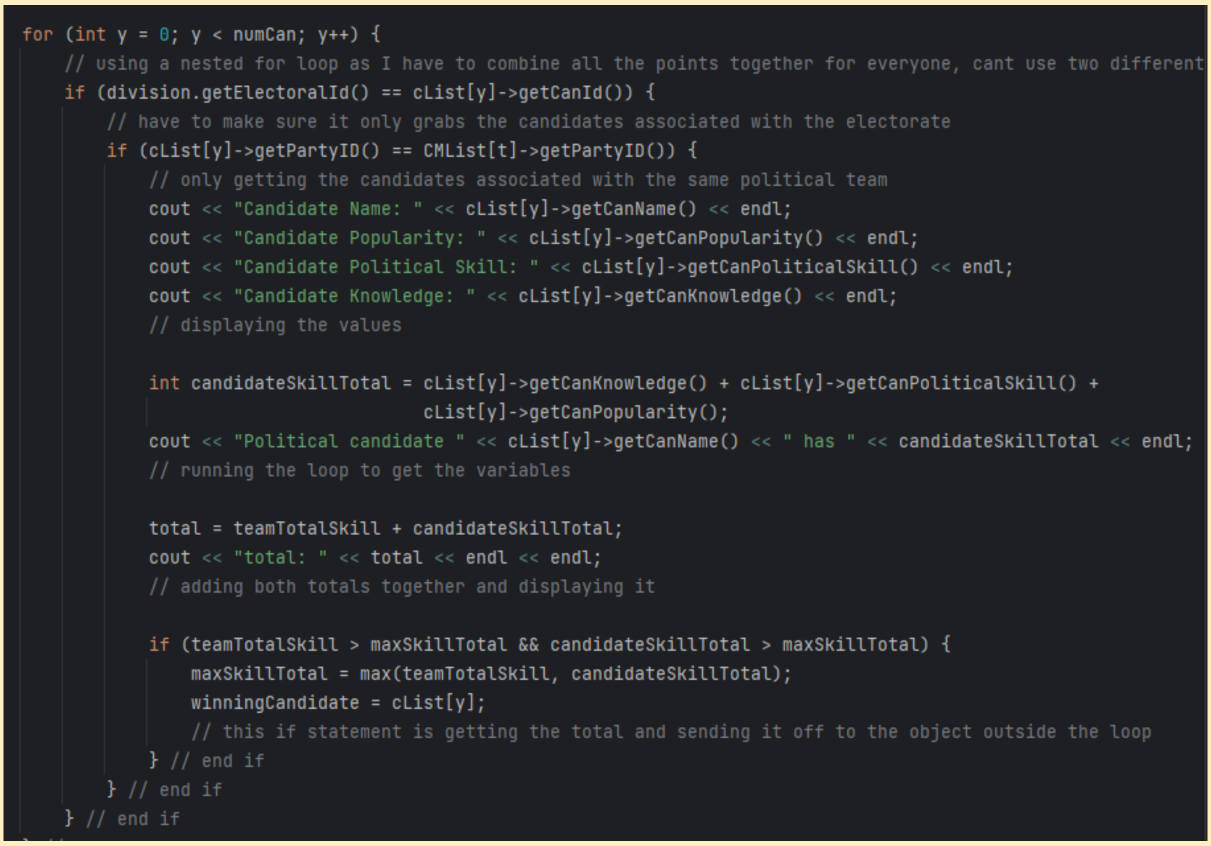
```
   int maxSkillTotal = 0;
   int total;
   Candidate *winningCandidate = nullptr;
```

It has a winning candidate outside the loop that I want to be updated last. If it was in the loop then it would keep adding candidates even if there was one after that had higher stats



```
cout << "__________________START DEBATE_______________" << endl;
cout << "The winner is the candidate that has the most total points in the following skills" << endl;

for (int t = 0; t < numTeam; t++) {
    cout << "Political Team " << CMList[t]->getTeamId() << ": " << CMList[t]->getPartyName() << " has: " << endl;
    cout << "Organisational Skill: " << CMList[t]->getOrganisationalSkill() << endl;
    cout << "Team strategic Skill: " << CMList[t]->getStrategicSkill() << endl;
    cout << "Team size: " << CMList[t]->getSize() << endl;
    // displaying the values

    int teamTotalSkill =
            CMList[t]->getStrategicSkill() + CMList[t]->getOrganisationalSkill() + CMList[t]->getSize();
    cout << "Political Team " << CMList[t]->getTeamId() << " has " << teamTotalSkill << endl << endl;
    // using a loop to run over the array for teams to get the total points for each team, and displaying it
```

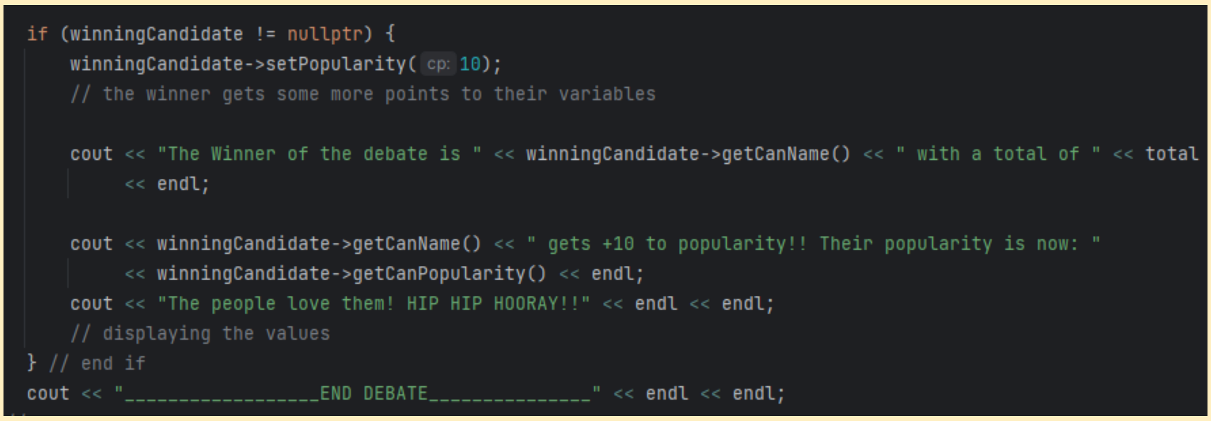
This loops is designed to get the variables from the teams and add it to a total skill, to be added later.



```cpp
for (int y = 0; y < numCan; y++) {
    // using a nested for loop as I have to combine all the points together for everyone, cant use two different
    if (division.getElectoralId() == cList[y]->getCanId()) {
        // have to make sure it only grabs the candidates associated with the electorate
        if (cList[y]->getPartyID() == CMList[t]->getPartyID()) {
            // only getting the candidates associated with the same political team
            cout << "Candidate Name: " << cList[y]->getCanName() << endl;
            cout << "Candidate Popularity: " << cList[y]->getCanPopularity() << endl;
            cout << "Candidate Political Skill: " << cList[y]->getCanPoliticalSkill() << endl;
            cout << "Candidate Knowledge: " << cList[y]->getCanKnowledge() << endl;
            // displaying the values

            int candidateSkillTotal = cList[y]->getCanKnowledge() + cList[y]->getCanPoliticalSkill() +
                                      cList[y]->getCanPopularity();
            cout << "Political candidate " << cList[y]->getCanName() << " has " << candidateSkillTotal << endl;
            // running the loop to get the variables

            total = teamTotalSkill + candidateSkillTotal;
            cout << "total: " << total << endl << endl;
            // adding both totals together and displaying it

            if (teamTotalSkill > maxSkillTotal && candidateSkillTotal > maxSkillTotal) {
                maxSkillTotal = max(teamTotalSkill, candidateSkillTotal);
                winningCandidate = cList[y];
                // this if statement is getting the total and sending it off to the object outside the loop
            } // end if
        } // end if
    } // end if
```

This loop grabs the candidate details adds it to a total Candidate variable.

Right after we plus the CM team total and the candidate total to equal the complete total. I then use an if statement that compares them to the previous max total. This starts at 0 and is declared outside of the loop so the loop will just keep adding to it then reset after. If both those totals are over the previous max total it adds the new total to the max total and creates the object outside the loop.



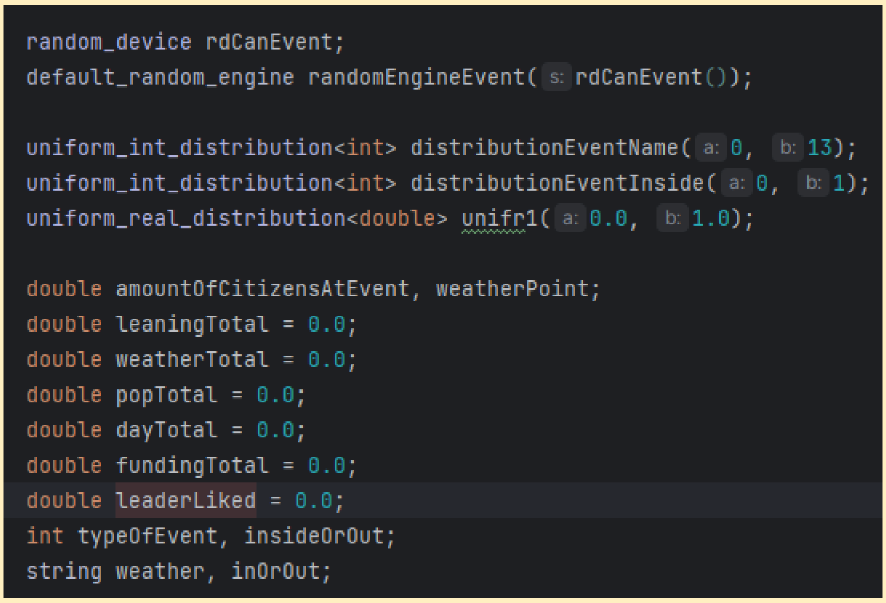
```cpp
if (winningCandidate != nullptr) {
    winningCandidate->setPopularity( cp: 10);
    // the winner gets some more points to their variables

    cout << "The Winner of the debate is " << winningCandidate->getCanName() << " with a total of " << total
         << endl;

    cout << winningCandidate->getCanName() << " gets +10 to popularity!! Their popularity is now: "
         << winningCandidate->getCanPopularity() << endl;
    cout << "The people love them! HIP HIP HOORAY!!" << endl << endl;
    // displaying the values
} // end if
cout << "_________________END DEBATE_______________" << endl << endl;
```

This takes the winning object and only runs if it isn't a null pointer. If it is a null pointer, it will mess the entire program with a bad_alloc.

It then displays all the details of the winning candidate and adds points to their popularity.

Candidate Event

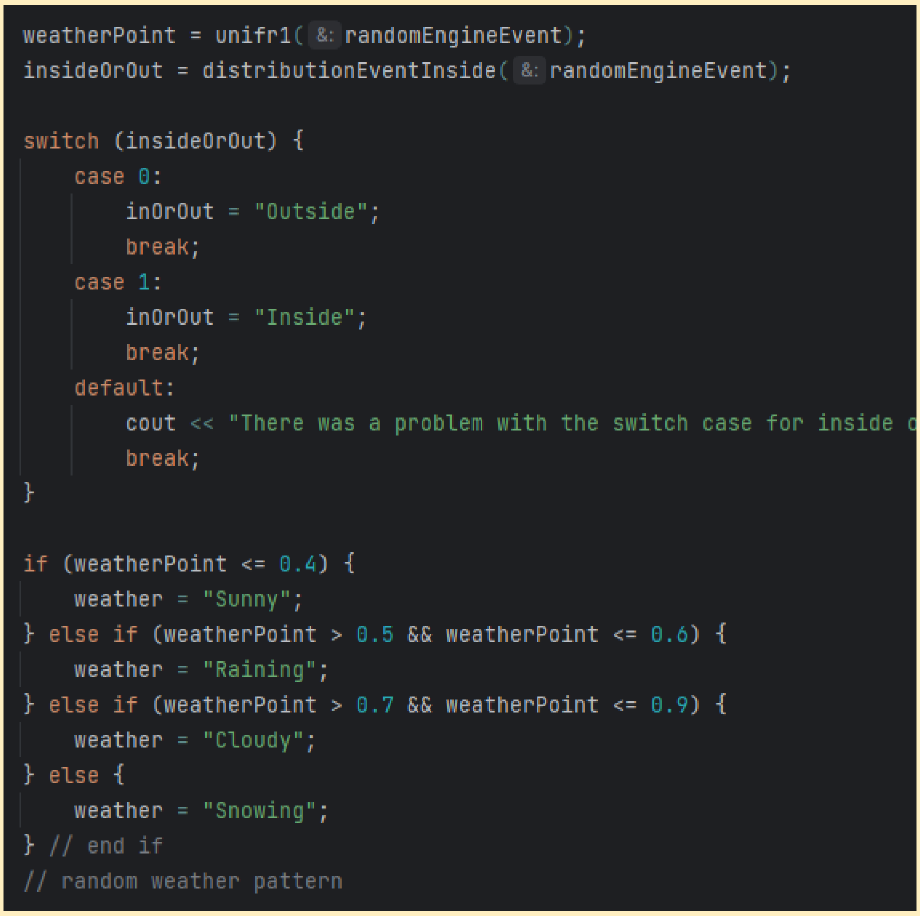
This event is designed around the amount of people a candidate must have at his event to make it successful. If it is successful the candidate will gain popularity and if it is unsuccessful they will loose popularity. They start at 0 and have to make it over the benchmark at 0.1 percent of the population for success.



```
random_device rdCanEvent;
default_random_engine randomEngineEvent( s: rdCanEvent());

uniform_int_distribution<int> distributionEventName( a: 0,  b: 13);
uniform_int_distribution<int> distributionEventInside( a: 0,  b: 1);
uniform_real_distribution<double> unifr1( a: 0.0,  b: 1.0);

double amountOfCitizensAtEvent, weatherPoint;
double leaningTotal = 0.0;
double weatherTotal = 0.0;
double popTotal = 0.0;
double dayTotal = 0.0;
double fundingTotal = 0.0;
double leaderLiked = 0.0;
int typeOfEvent, insideOrOut;
string weather, inOrOut;
```

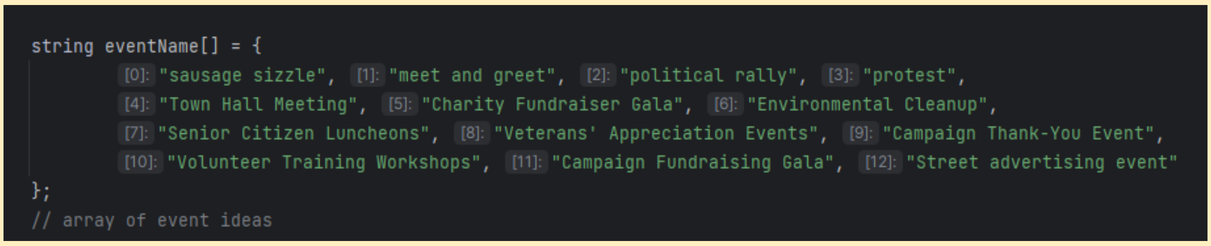
I have created a few random event machines for this, they pick the index for the event name, whether the even is inside or out, and the weather.

There is also a few variables that need to be reset to 0



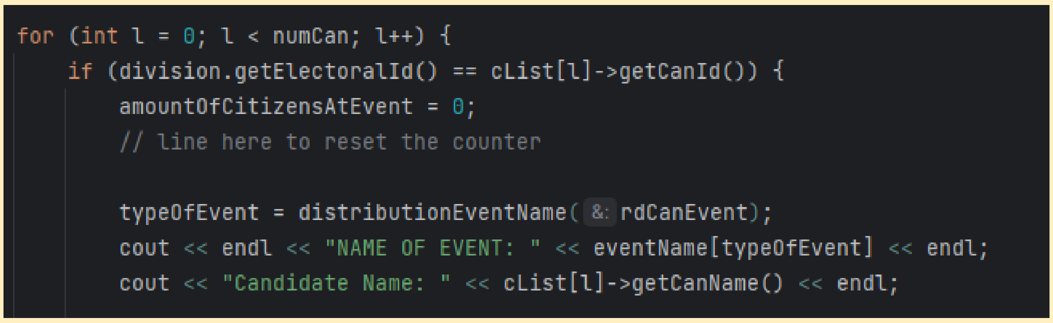
```
weatherPoint = unifr1( &: randomEngineEvent);
insideOrOut = distributionEventInside( &: randomEngineEvent);

switch (insideOrOut) {
    case 0:
        inOrOut = "Outside";
        break;
    case 1:
        inOrOut = "Inside";
        break;
    default:
        cout << "There was a problem with the switch case for inside o
        break;
}

if (weatherPoint <= 0.4) {
    weather = "Sunny";
} else if (weatherPoint > 0.5 && weatherPoint <= 0.6) {
    weather = "Raining";
} else if (weatherPoint > 0.7 && weatherPoint <= 0.9) {
    weather = "Cloudy";
} else {
    weather = "Snowing";
} // end if
// random weather pattern
```

The code for picking the weather and location of the event



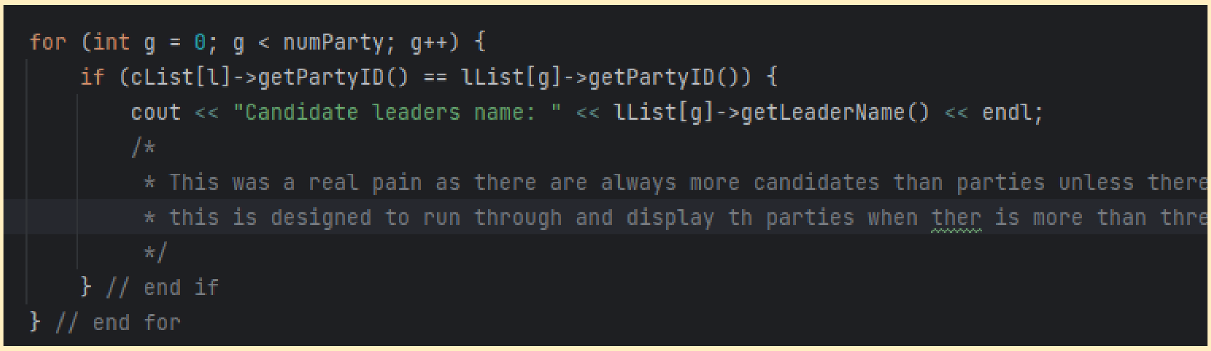
```
string eventName[] = {
        [0]: "sausage sizzle",  [1]: "meet and greet",  [2]: "political rally",  [3]: "protest",
        [4]: "Town Hall Meeting",  [5]: "Charity Fundraiser Gala",  [6]: "Environmental Cleanup",
        [7]: "Senior Citizen Luncheons",  [8]: "Veterans' Appreciation Events",  [9]: "Campaign Thank-You Event",
        [10]: "Volunteer Training Workshops",  [11]: "Campaign Fundraising Gala",  [12]: "Street advertising event"
};
// array of event ideas
```

Some event names to pick from



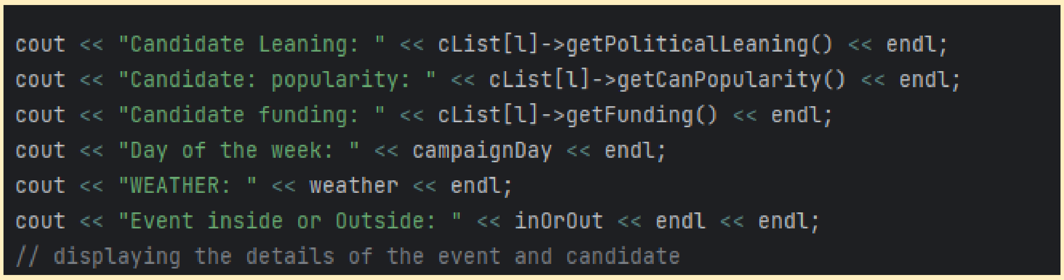
```
for (int l = 0; l < numCan; l++) {
    if (division.getElectoralId() == cList[l]->getCanId()) {
        amountOfCitizensAtEvent = 0;
        // line here to reset the counter

        typeOfEvent = distributionEventName( &: rdCanEvent);
        cout << endl << "NAME OF EVENT: " << eventName[typeOfEvent] << endl;
        cout << "Candidate Name: " << cList[l]->getCanName() << endl;
```

For loop to run through the candidates, and display the names. The if statement is designed to only grab the candidates associated with the electorate. Then it wont get all of them.



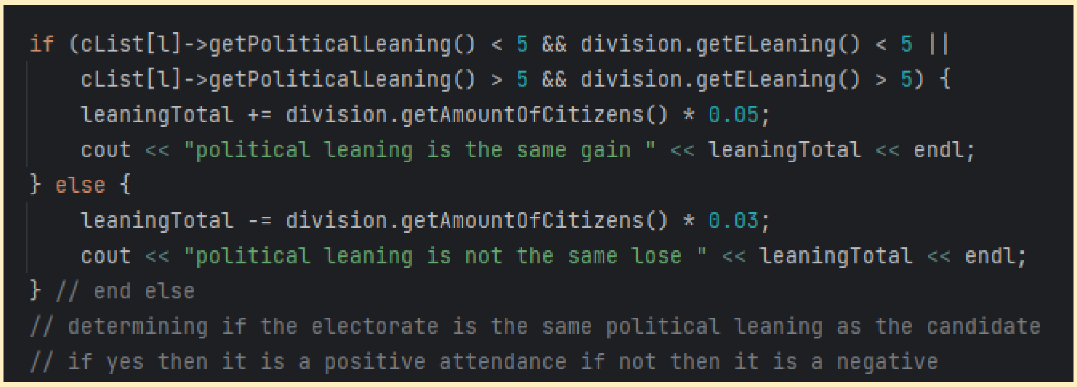
```
for (int g = 0; g < numParty; g++) {
    if (cList[l]->getPartyID() == lList[g]->getPartyID()) {
        cout << "Candidate leaders name: " << lList[g]->getLeaderName() << endl;
        /*
         * This was a real pain as there are always more candidates than parties unless there
         * this is designed to run through and display th parties when ther is more than thre
         */
    } // end if
} // end for
```

This was a real pain as there are always more candidates than parties unless there is only one electorate. This is designed to run through and display the parties when there is more than three. Avoiding bad_alloc



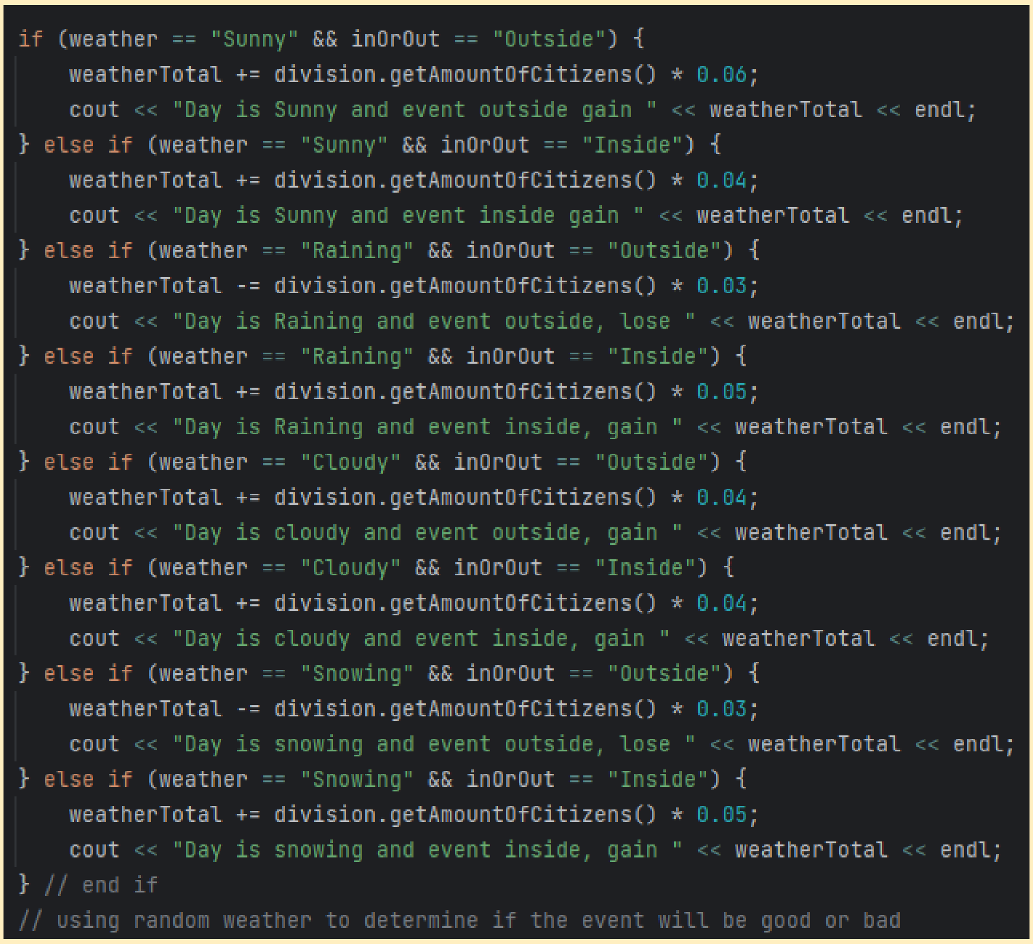
```
cout << "Candidate Leaning: " << cList[l]->getPoliticalLeaning() << endl;
cout << "Candidate: popularity: " << cList[l]->getCanPopularity() << endl;
cout << "Candidate funding: " << cList[l]->getFunding() << endl;
cout << "Day of the week: " << campaignDay << endl;
cout << "WEATHER: " << weather << endl;
cout << "Event inside or Outside: " << inOrOut << endl << endl;
// displaying the details of the event and candidate
```

I then display all the information related to the candidate



```cpp
if (cList[l]->getPoliticalLeaning() < 5 && division.getELeaning() < 5 ||
    cList[l]->getPoliticalLeaning() > 5 && division.getELeaning() > 5) {
    leaningTotal += division.getAmountOfCitizens() * 0.05;
    cout << "political leaning is the same gain " << leaningTotal << endl;
} else {
    leaningTotal -= division.getAmountOfCitizens() * 0.03;
    cout << "political leaning is not the same lose " << leaningTotal << endl;
} // end else
// determining if the electorate is the same political leaning as the candidate
// if yes then it is a positive attendance if not then it is a negative
```

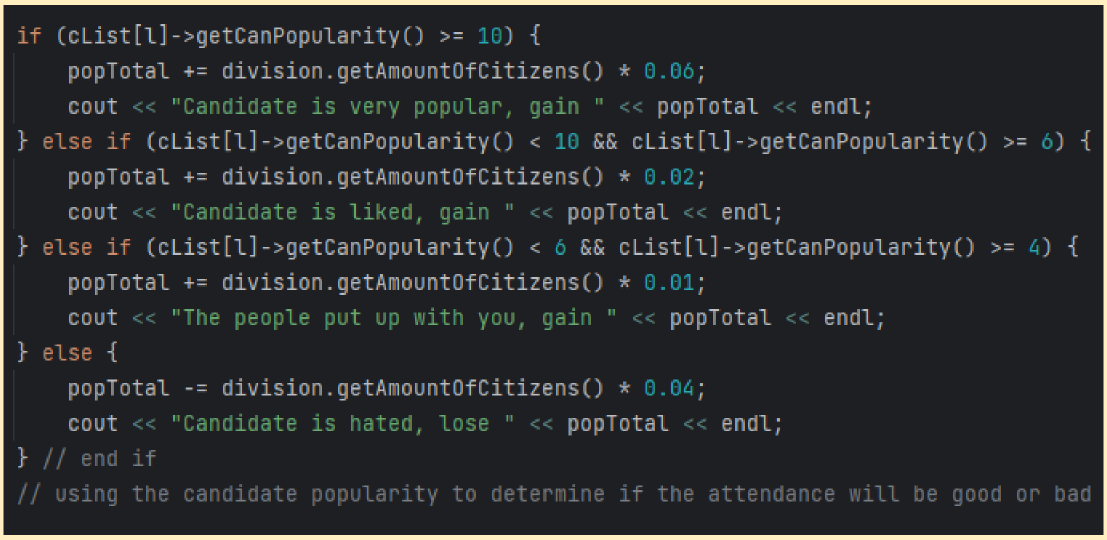
This is my first check. It is designed to check to see if the political party is the same leaning as the electorate. It was thought up because say if the electorate doesn’t agree with the party like Wollongong is left leaning wont vote for a right leaning party.



```cpp
if (weather == "Sunny" && inOrOut == "Outside") {
    weatherTotal += division.getAmountOfCitizens() * 0.06;
    cout << "Day is Sunny and event outside gain " << weatherTotal << endl;
} else if (weather == "Sunny" && inOrOut == "Inside") {
    weatherTotal += division.getAmountOfCitizens() * 0.04;
    cout << "Day is Sunny and event inside gain " << weatherTotal << endl;
} else if (weather == "Raining" && inOrOut == "Outside") {
    weatherTotal -= division.getAmountOfCitizens() * 0.03;
    cout << "Day is Raining and event outside, lose " << weatherTotal << endl;
} else if (weather == "Raining" && inOrOut == "Inside") {
    weatherTotal += division.getAmountOfCitizens() * 0.05;
    cout << "Day is Raining and event inside, gain " << weatherTotal << endl;
} else if (weather == "Cloudy" && inOrOut == "Outside") {
    weatherTotal += division.getAmountOfCitizens() * 0.04;
    cout << "Day is cloudy and event outside, gain " << weatherTotal << endl;
} else if (weather == "Cloudy" && inOrOut == "Inside") {
    weatherTotal += division.getAmountOfCitizens() * 0.04;
    cout << "Day is cloudy and event inside, gain " << weatherTotal << endl;
} else if (weather == "Snowing" && inOrOut == "Outside") {
    weatherTotal -= division.getAmountOfCitizens() * 0.03;
    cout << "Day is snowing and event outside, lose " << weatherTotal << endl;
} else if (weather == "Snowing" && inOrOut == "Inside") {
    weatherTotal += division.getAmountOfCitizens() * 0.05;
    cout << "Day is snowing and event inside, gain " << weatherTotal << endl;
} // end if
// using random weather to determine if the event will be good or bad
```

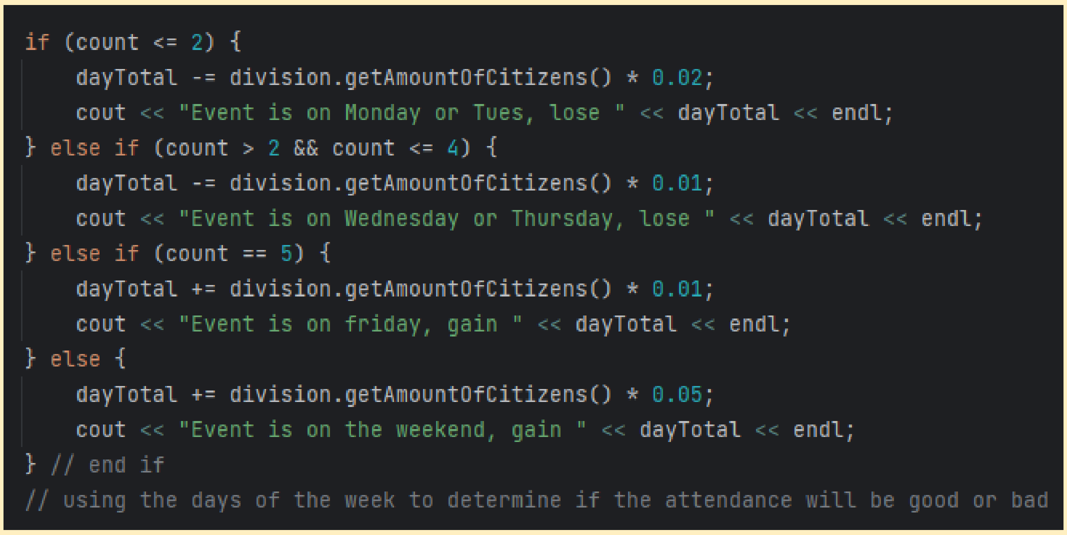
This is a fun one, I figure if the weather is shitty there will be less people turning up. So if the weather is good they get more if its bad they get less.

It Also takes into consideration if the event is inside or out. It its inside on a bad day they don’t loose people, while if its outside they loose people.



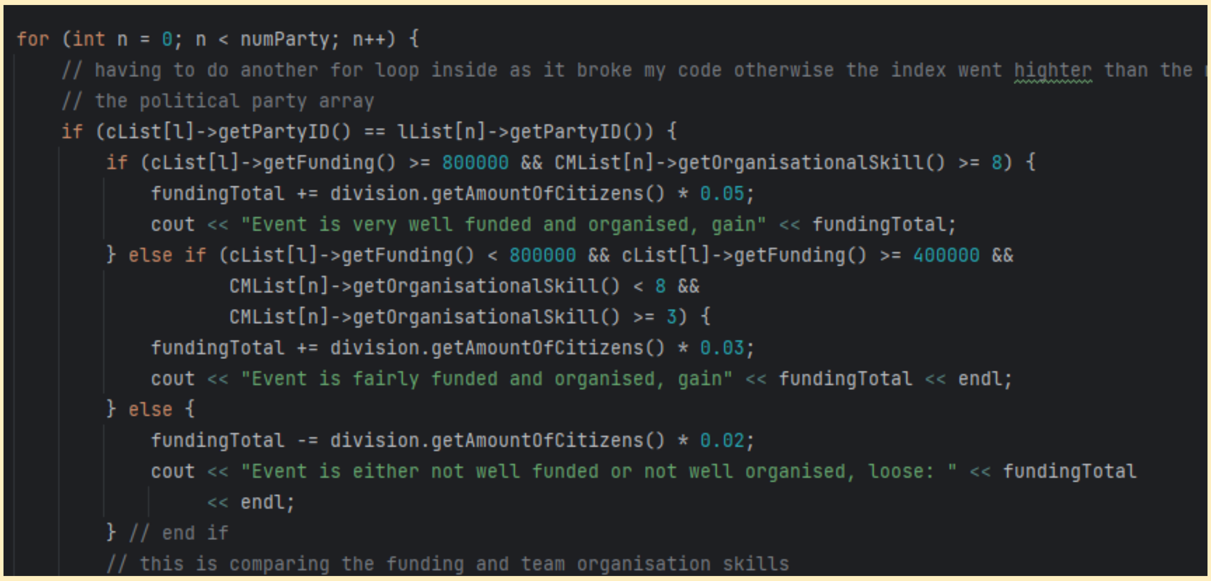
```cpp
if (cList[l]->getCanPopularity() >= 10) {
    popTotal += division.getAmountOfCitizens() * 0.06;
    cout << "Candidate is very popular, gain " << popTotal << endl;
} else if (cList[l]->getCanPopularity() < 10 && cList[l]->getCanPopularity() >= 6) {
    popTotal += division.getAmountOfCitizens() * 0.02;
    cout << "Candidate is liked, gain " << popTotal << endl;
} else if (cList[l]->getCanPopularity() < 6 && cList[l]->getCanPopularity() >= 4) {
    popTotal += division.getAmountOfCitizens() * 0.01;
    cout << "The people put up with you, gain " << popTotal << endl;
} else {
    popTotal -= division.getAmountOfCitizens() * 0.04;
    cout << "Candidate is hated, lose " << popTotal << endl;
} // end if
// using the candidate popularity to determine if the attendance will be good or bad
```

This is considering the candidate popularity. The higher the popularity the more people they will have at the event.



```cpp
if (count <= 2) {
    dayTotal -= division.getAmountOfCitizens() * 0.02;
    cout << "Event is on Monday or Tues, lose " << dayTotal << endl;
} else if (count > 2 && count <= 4) {
    dayTotal -= division.getAmountOfCitizens() * 0.01;
    cout << "Event is on Wednesday or Thursday, lose " << dayTotal << endl;
} else if (count == 5) {
    dayTotal += division.getAmountOfCitizens() * 0.01;
    cout << "Event is on friday, gain " << dayTotal << endl;
} else {
    dayTotal += division.getAmountOfCitizens() * 0.05;
    cout << "Event is on the weekend, gain " << dayTotal << endl;
} // end if
// using the days of the week to determine if the attendance will be good or bad
```

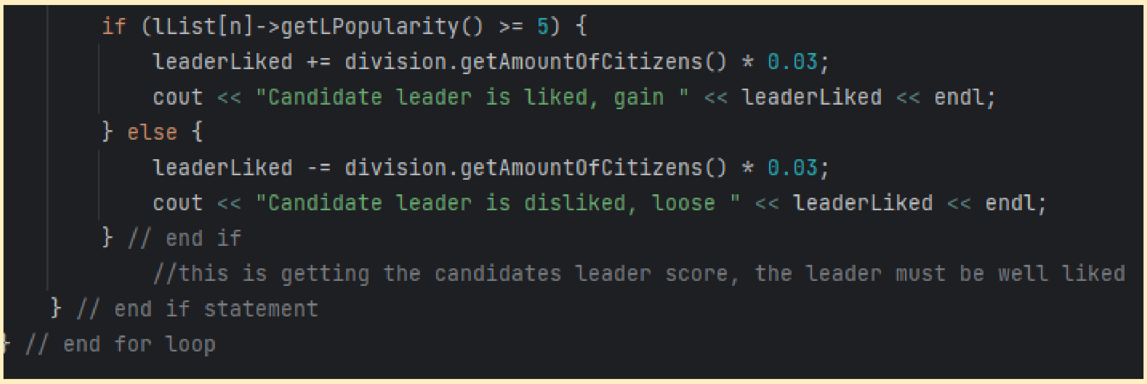
The next part is considering what day it is. Weekdays get less people, weekends get more.



```cpp
for (int n = 0; n < numParty; n++) {
    // having to do another for loop inside as it broke my code otherwise the index went highter than the
    // the political party array
    if (cList[l]->getPartyID() == lList[n]->getPartyID()) {
        if (cList[l]->getFunding() >= 800000 && CMList[n]->getOrganisationalSkill() >= 8) {
            fundingTotal += division.getAmountOfCitizens() * 0.05;
            cout << "Event is very well funded and organised, gain" << fundingTotal;
        } else if (cList[l]->getFunding() < 800000 && cList[l]->getFunding() >= 400000 &&
                   CMList[n]->getOrganisationalSkill() < 8 &&
                   CMList[n]->getOrganisationalSkill() >= 3) {
            fundingTotal += division.getAmountOfCitizens() * 0.03;
            cout << "Event is fairly funded and organised, gain" << fundingTotal << endl;
        } else {
            fundingTotal -= division.getAmountOfCitizens() * 0.02;
            cout << "Event is either not well funded or not well organised, loose: " << fundingTotal
                 << endl;
        } // end if
        // this is comparing the funding and team organisation skills
```

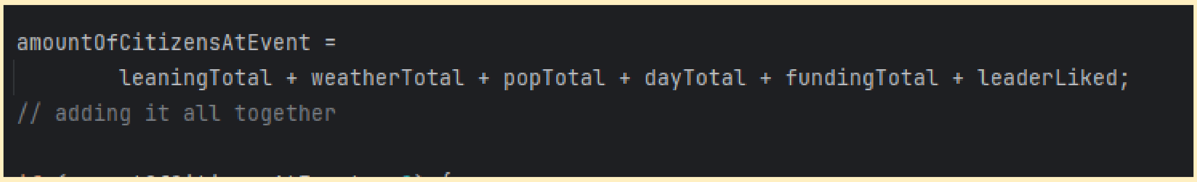
This next for loop was super frustrating but it is designed to avoid bad_alloc due to there being more candidates than parties.

This part gets the parties funding and the teams organizational skill and decides if the event is going to be well funded and organized. The more points in both of them gets more people as who doesn't like a well funded and organized event.



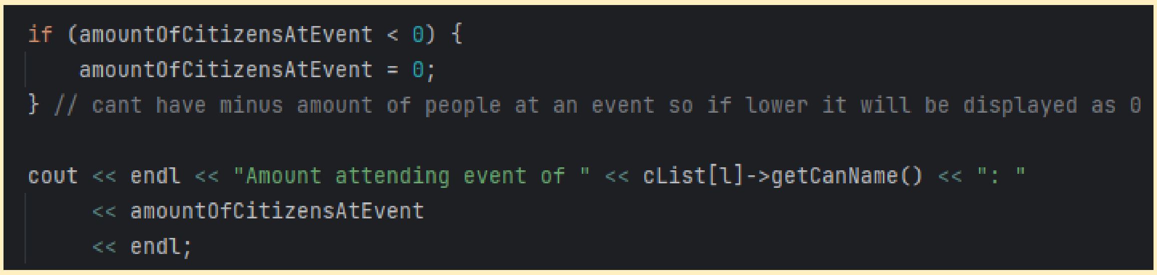
```
        if (lList[n]->getLPopularity() >= 5) {
            leaderLiked += division.getAmountOfCitizens() * 0.03;
            cout << "Candidate leader is liked, gain " << leaderLiked << endl;
        } else {
            leaderLiked -= division.getAmountOfCitizens() * 0.03;
            cout << "Candidate leader is disliked, loose " << leaderLiked << endl;
        } // end if
            //this is getting the candidates leader score, the leader must be well liked
    } // end if statement
} // end for loop
```

This gets the popularity of the leader and the higher means more people will come along



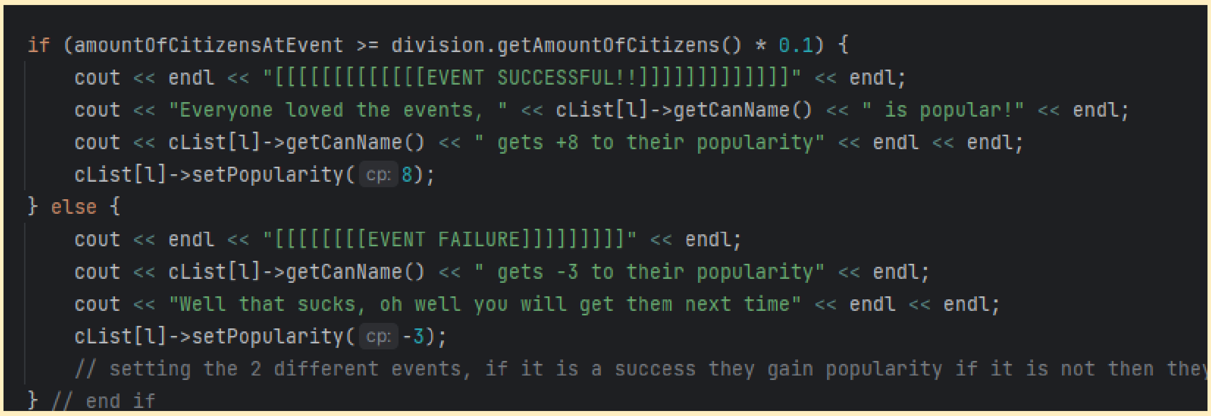
```
amountOfCitizensAtEvent =
        leaningTotal + weatherTotal + popTotal + dayTotal + fundingTotal + leaderLiked;
// adding it all together
```

Adding it all together



```
if (amountOfCitizensAtEvent < 0) {
    amountOfCitizensAtEvent = 0;
} // cant have minus amount of people at an event so if lower it will be displayed as 0

cout << endl << "Amount attending event of " << cList[l]->getCanName() << ": "
     << amountOfCitizensAtEvent
     << endl;
```

Has to make sure there is no negative, then displays the amount of people at the event



```
if (amountOfCitizensAtEvent >= division.getAmountOfCitizens() * 0.1) {
    cout << endl << "[[[[[[[[[[[[[EVENT SUCCESSFUL!!]]]]]]]]]]]]]" << endl;
    cout << "Everyone loved the events, " << cList[l]->getCanName() << " is popular!" << endl;
    cout << cList[l]->getCanName() << " gets +8 to their popularity" << endl << endl;
    cList[l]->setPopularity( cp: 8);
} else {
    cout << endl << "[[[[[[[[EVENT FAILURE]]]]]]]]" << endl;
    cout << cList[l]->getCanName() << " gets -3 to their popularity" << endl;
    cout << "Well that sucks, oh well you will get them next time" << endl << endl;
    cList[l]->setPopularity( cp: -3);
    // setting the 2 different events, if it is a success they gain popularity if it is not then they
} // end if
```

This is where it decided if the event is a success or a failure using the benchmark of 0.1

Leader Event

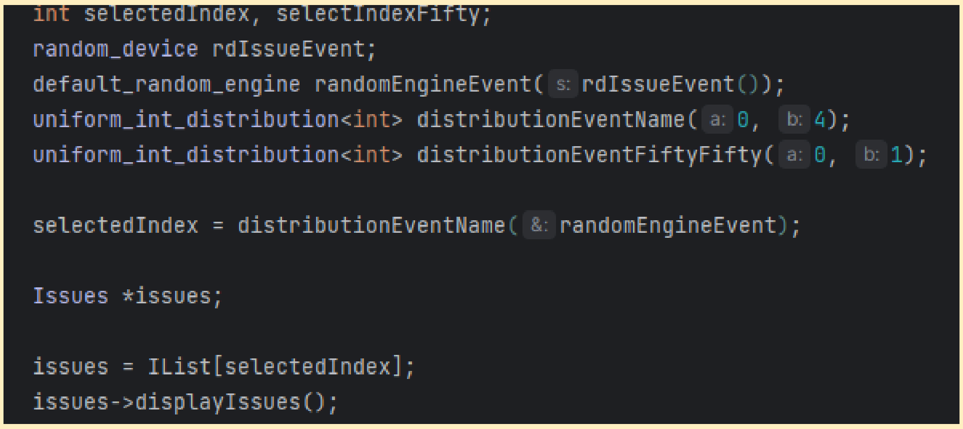
The leader event is designed pretty much the same as candidate events. It just doesn't have as many checks. There is no need to screenshot everything again. If successful it adds popularity to the leader stat, and if unsuccessful they lose popularity

Issue Events

Issue events were the most interesting to make, It took me a while to figure out what I wanted. I decided that I will calculate the Stance and give candidates a chance to change theirs if its nowhere near the electorate's stances.

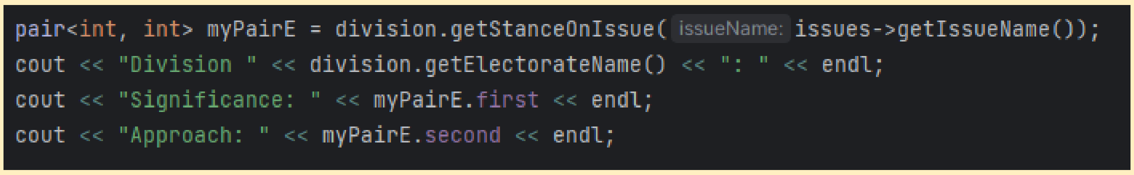
Note

I accidently forgot to comment on the file before I did the report. The appropriate comments will be in the file when handed in.



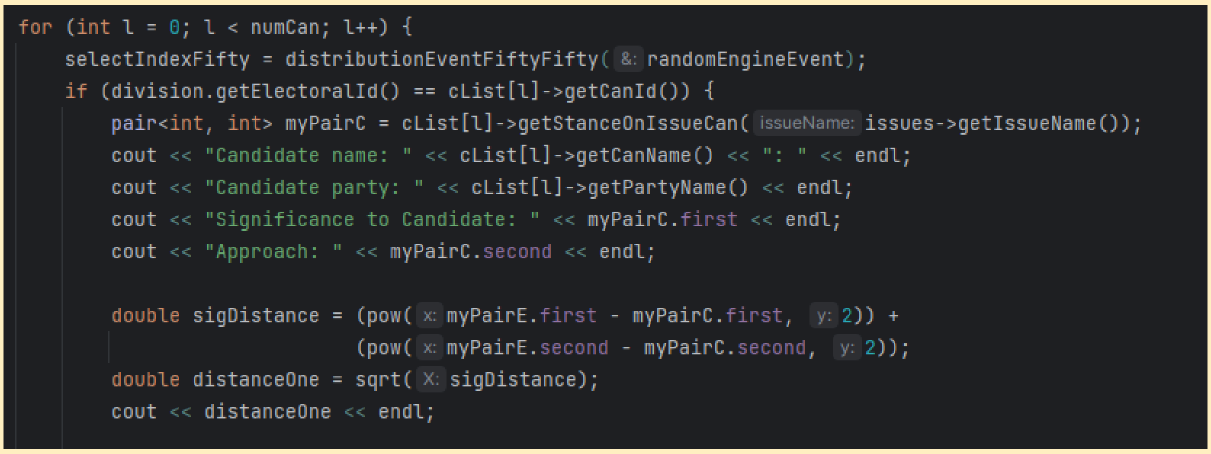
```
int selectedIndex, selectIndexFifty;
random_device rdIssueEvent;
default_random_engine randomEngineEvent( s: rdIssueEvent());
uniform_int_distribution<int> distributionEventName( a: 0,  b: 4);
uniform_int_distribution<int> distributionEventFiftyFifty( a: 0,  b: 1);

selectedIndex = distributionEventName( &: randomEngineEvent);

Issues *issues;

issues = IList[selectedIndex];
issues->displayIssues();
```

It starts with a few random generators and a pointer to get the details of the Issue that had been selected by random.



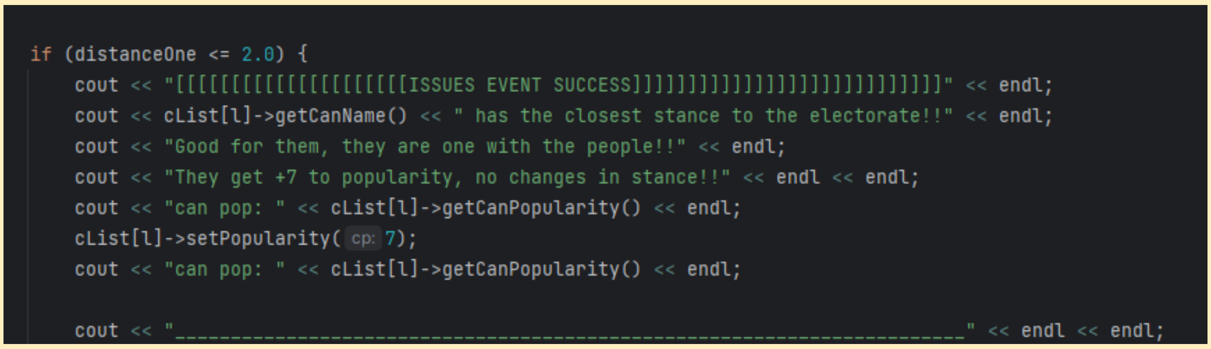
```
pair<int, int> myPairE = division.getStanceOnIssue( issueName: issues->getIssueName());
cout << "Division " << division.getElectorateName() << ": " << endl;
cout << "Significance: " << myPairE.first << endl;
cout << "Approach: " << myPairE.second << endl;
```

This is the getter to get the details for the divisions stance, it uses the issue name that was generated to get the variables.



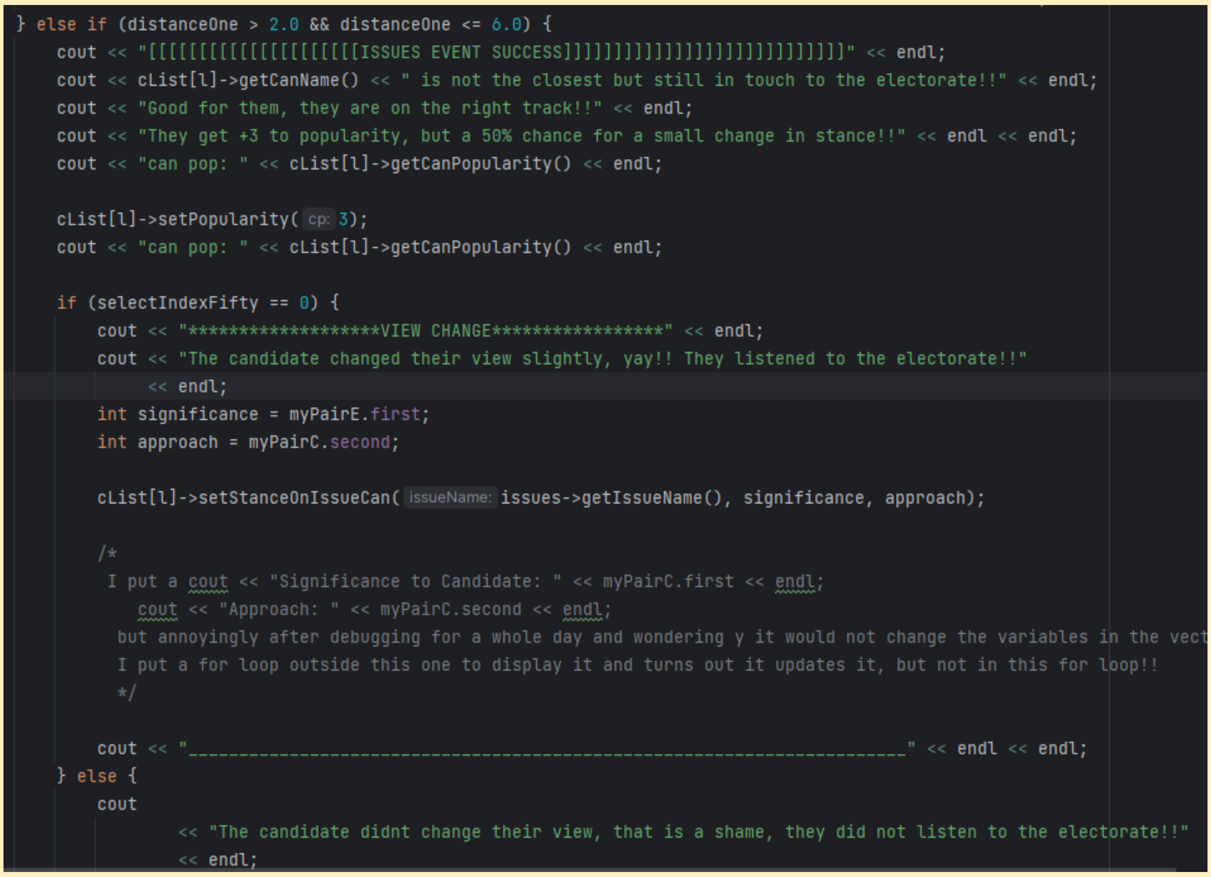
```
for (int l = 0; l < numCan; l++) {
    selectIndexFifty = distributionEventFiftyFifty( &: randomEngineEvent);
    if (division.getElectoralId() == cList[l]->getCanId()) {
        pair<int, int> myPairC = cList[l]->getStanceOnIssueCan( issueName: issues->getIssueName());
        cout << "Candidate name: " << cList[l]->getCanName() << ": " << endl;
        cout << "Candidate party: " << cList[l]->getPartyName() << endl;
        cout << "Significance to Candidate: " << myPairC.first << endl;
        cout << "Approach: " << myPairC.second << endl;

        double sigDistance = (pow( x: myPairE.first - myPairC.first,  y: 2)) +
                             (pow( x: myPairE.second - myPairC.second,  y: 2));
        double distanceOne = sqrt( X: sigDistance);
        cout << distanceOne << endl;
```

This is the fun bit. It uses a for loop and an if statement to get the right candidates then I use the getter method to get the candidates stances on the issue and display them. Then it does the stance calculations. I found the stance calculation formular on google and double checked its output on an online calculator for it



```
if (distanceOne <= 2.0) {
    cout << "[[[[[[[[[[[[[[[[[[[[[ISSUES EVENT SUCCESS]]]]]]]]]]]]]]]]]]]]]]]]]]]]" << endl;
    cout << cList[l]->getCanName() << " has the closest stance to the electorate!!" << endl;
    cout << "Good for them, they are one with the people!!" << endl;
    cout << "They get +7 to popularity, no changes in stance!!" << endl << endl;
    cout << "can pop: " << cList[l]->getCanPopularity() << endl;
    cList[l]->setPopularity( cp: 7);
    cout << "can pop: " << cList[l]->getCanPopularity() << endl;

    cout << "_______________________________________________________________________" << endl << endl;
```

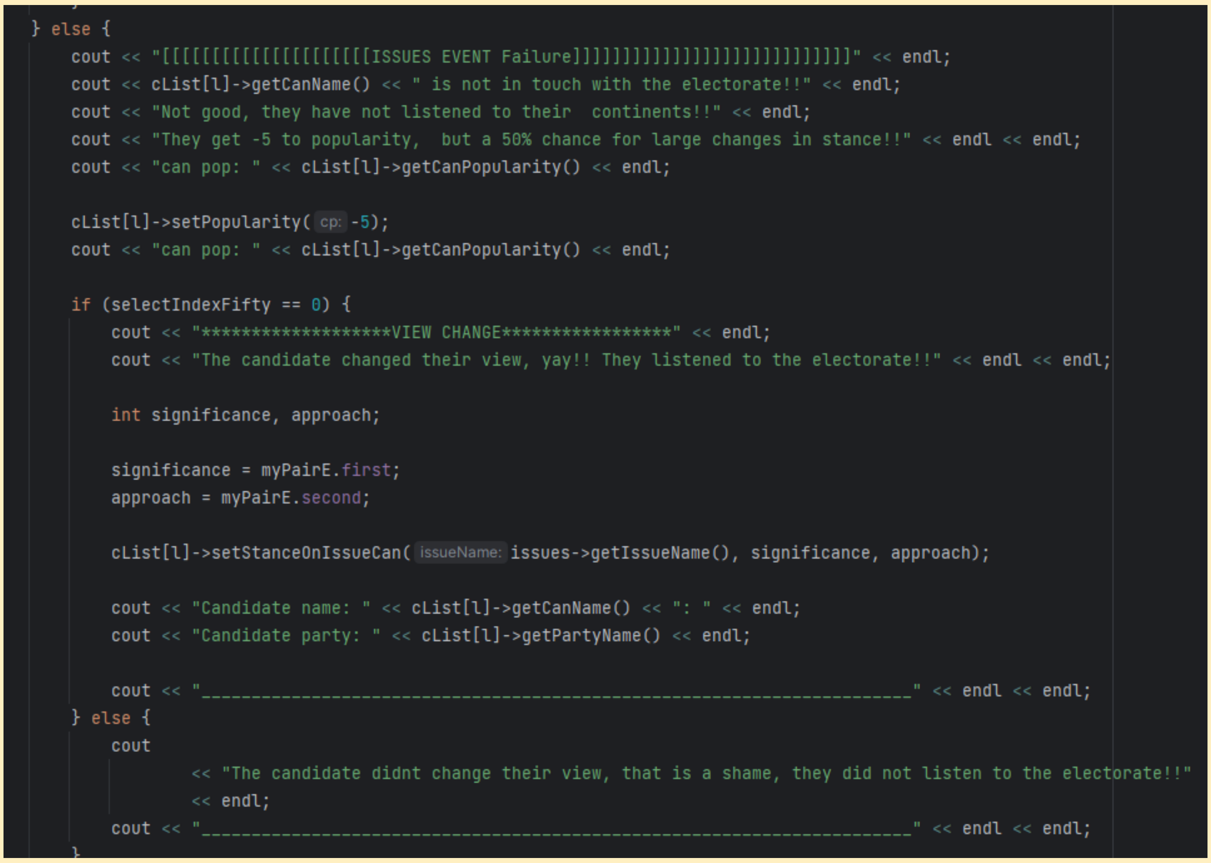
This is the first finish, if the stance is lower than 2 I call it a success and the candidate gets popularity



```
} else if (distanceOne > 2.0 && distanceOne <= 6.0) {
    cout << "[[[[[[[[[[[[[[[[[[[[[ISSUES EVENT SUCCESS]]]]]]]]]]]]]]]]]]]]]]]]]]]]" << endl;
    cout << cList[l]->getCanName() << " is not the closest but still in touch to the electorate!!" << endl;
    cout << "Good for them, they are on the right track!!" << endl;
    cout << "They get +3 to popularity, but a 50% chance for a small change in stance!!" << endl << endl;
    cout << "can pop: " << cList[l]->getCanPopularity() << endl;

    cList[l]->setPopularity( cp: 3);
    cout << "can pop: " << cList[l]->getCanPopularity() << endl;

    if (selectIndexFifty == 0) {
        cout << "******************VIEW CHANGE*****************" << endl;
        cout << "The candidate changed their view slightly, yay!! They listened to the electorate!!"
             << endl;
        int significance = myPairE.first;
        int approach = myPairC.second;

        cList[l]->setStanceOnIssueCan( issueName: issues->getIssueName(), significance, approach);

        /*
         I put a cout << "Significance to Candidate: " << myPairC.first << endl;
            cout << "Approach: " << myPairC.second << endl;
          but annoyingly after debugging for a whole day and wondering y it would not change the variables in the vect
          I put a for loop outside this one to display it and turns out it updates it, but not in this for loop!!
          */

        cout << "______________________________________________________________________" << endl << endl;
    } else {
        cout
                << "The candidate didnt change their view, that is a shame, they did not listen to the electorate!!"
                << endl;
```

This next section to the code gets the stances and if they are a little bit off it adds some popularity and gives the candidate a 50% chance to slightly change his views to that of the electorate. This means only changing one variable.

This part had me frustrated as for some reason the setter didn't change the variables till after the loop was completed.



```cpp
} else {
    cout << "[[[[[[[[[[[[[[[[[[[[[ISSUES EVENT Failure]]]]]]]]]]]]]]]]]]]]]]]]]]]]" << endl;
    cout << cList[l]->getCanName() << " is not in touch with the electorate!!" << endl;
    cout << "Not good, they have not listened to their  continents!!" << endl;
    cout << "They get -5 to popularity,  but a 50% chance for large changes in stance!!" << endl << endl;
    cout << "can pop: " << cList[l]->getCanPopularity() << endl;

    cList[l]->setPopularity( cp: -5);
    cout << "can pop: " << cList[l]->getCanPopularity() << endl;

    if (selectIndexFifty == 0) {
        cout << "******************VIEW CHANGE*****************" << endl;
        cout << "The candidate changed their view, yay!! They listened to the electorate!!" << endl << endl;

        int significance, approach;

        significance = myPairE.first;
        approach = myPairE.second;

        cList[l]->setStanceOnIssueCan( issueName: issues->getIssueName(), significance, approach);

        cout << "Candidate name: " << cList[l]->getCanName() << ": " << endl;
        cout << "Candidate party: " << cList[l]->getPartyName() << endl;

        cout << "_______________________________________________________________________" << endl << endl;
    } else {
        cout
                << "The candidate didnt change their view, that is a shame, they did not listen to the electorate!!"
                << endl;
        cout << "_______________________________________________________________________" << endl << endl;
```

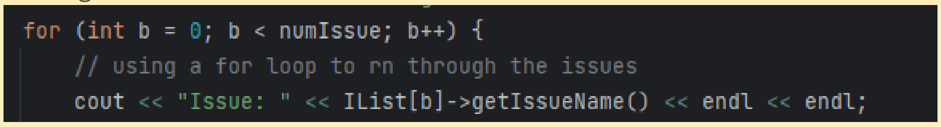
This was the final outcome. I call it a failure as the candidate has completely different views to that of the electorate, but if this is the case the candidate has a 50% chance to completely change their view to that of the electorate. That means changing the candidate's approach and significance variables.

Voting and Winner Point 7:

This part I have designed to use points, if a candidate wins due to having the best Average Stance Euclidean distance or popularity, they will earn points. The points given will be based off the spec. I will give 3 points for closest stance, 2 points for highest candidate popularity and 1 point for highest leader popularity.

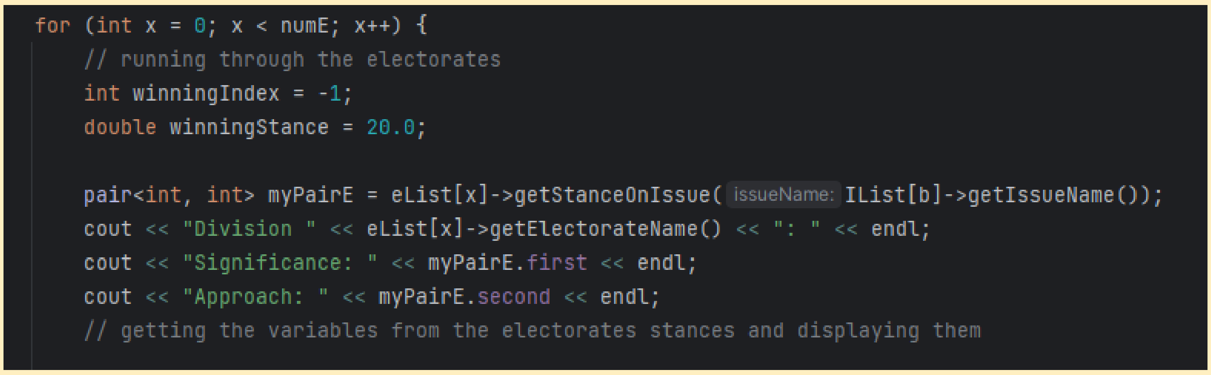
The Average Stance Euclidean distance calculation is handled pretty much the same way as the issues event did it except it calculates it all.

Average Stance Euclidean distance Calculation



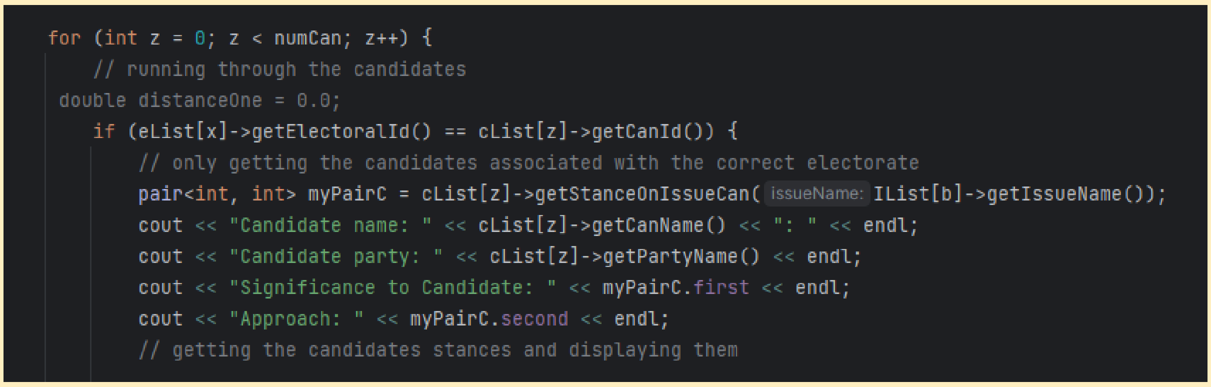
```cpp
for (int b = 0; b < numIssue; b++) {
    // using a for loop to rn through the issues
    cout << "Issue: " << IList[b]->getIssueName() << endl << endl;
```

For loop to get the issues from the array



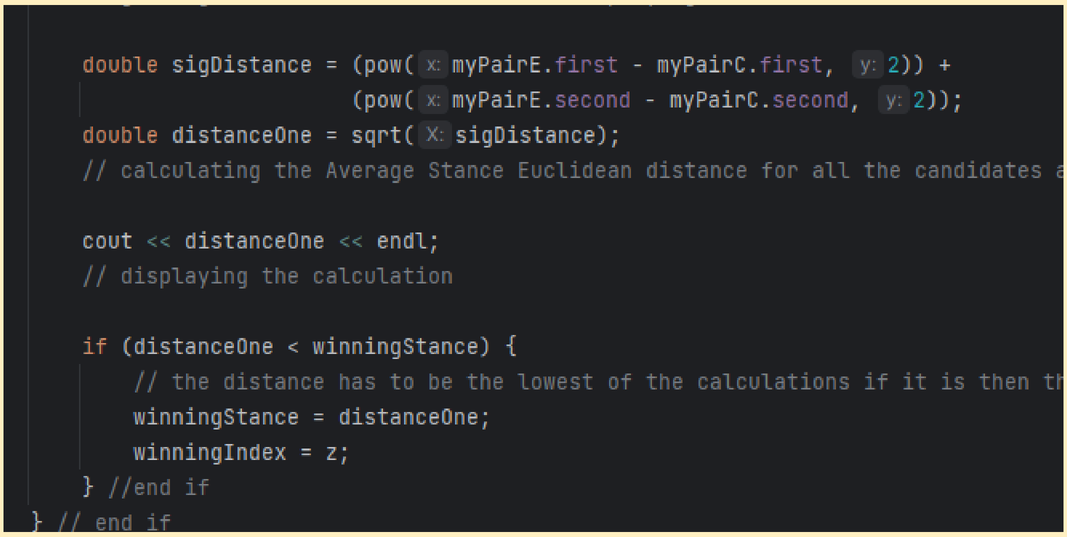
```
for (int x = 0; x < numE; x++) {
    // running through the electorates
    int winningIndex = -1;
    double winningStance = 20.0;

    pair<int, int> myPairE = eList[x]->getStanceOnIssue( issueName: IList[b]->getIssueName());
    cout << "Division " << eList[x]->getElectorateName() << ": " << endl;
    cout << "Significance: " << myPairE.first << endl;
    cout << "Approach: " << myPairE.second << endl;
    // getting the variables from the electorates stances and displaying them
```

For loop to get the stances for the electorate. I used a double as I wanted it as precise as possible



```
for (int z = 0; z < numCan; z++) {
    // running through the candidates
  double distanceOne = 0.0;
    if (eList[x]->getElectoralId() == cList[z]->getCanId()) {
        // only getting the candidates associated with the correct electorate
        pair<int, int> myPairC = cList[z]->getStanceOnIssueCan( issueName: IList[b]->getIssueName());
        cout << "Candidate name: " << cList[z]->getCanName() << ": " << endl;
        cout << "Candidate party: " << cList[z]->getPartyName() << endl;
        cout << "Significance to Candidate: " << myPairC.first << endl;
        cout << "Approach: " << myPairC.second << endl;
        // getting the candidates stances and displaying them
```

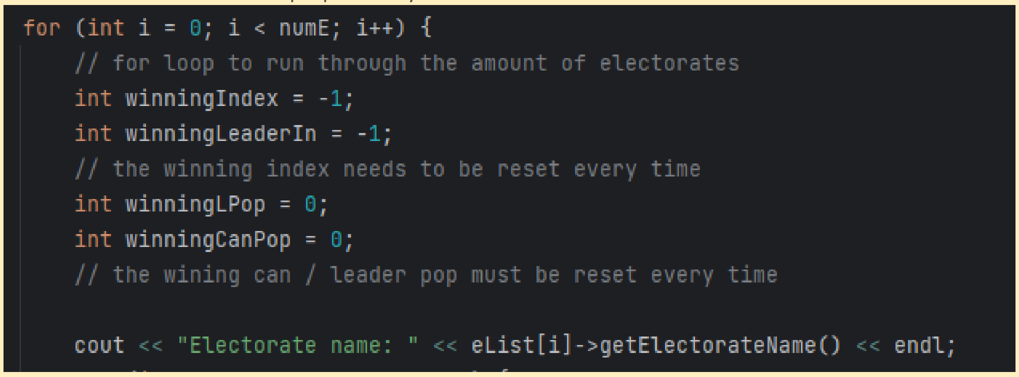
Getting the candidates stances and displaying them



```
    double sigDistance = (pow( x: myPairE.first - myPairC.first,  y: 2)) +
                         (pow( x: myPairE.second - myPairC.second,  y: 2));
    double distanceOne = sqrt( X: sigDistance);
    // calculating the Average Stance Euclidean distance for all the candidates a

    cout << distanceOne << endl;
    // displaying the calculation

    if (distanceOne < winningStance) {
        // the distance has to be the lowest of the calculations if it is then th
        winningStance = distanceOne;
        winningIndex = z;
    } //end if
} // end if
```

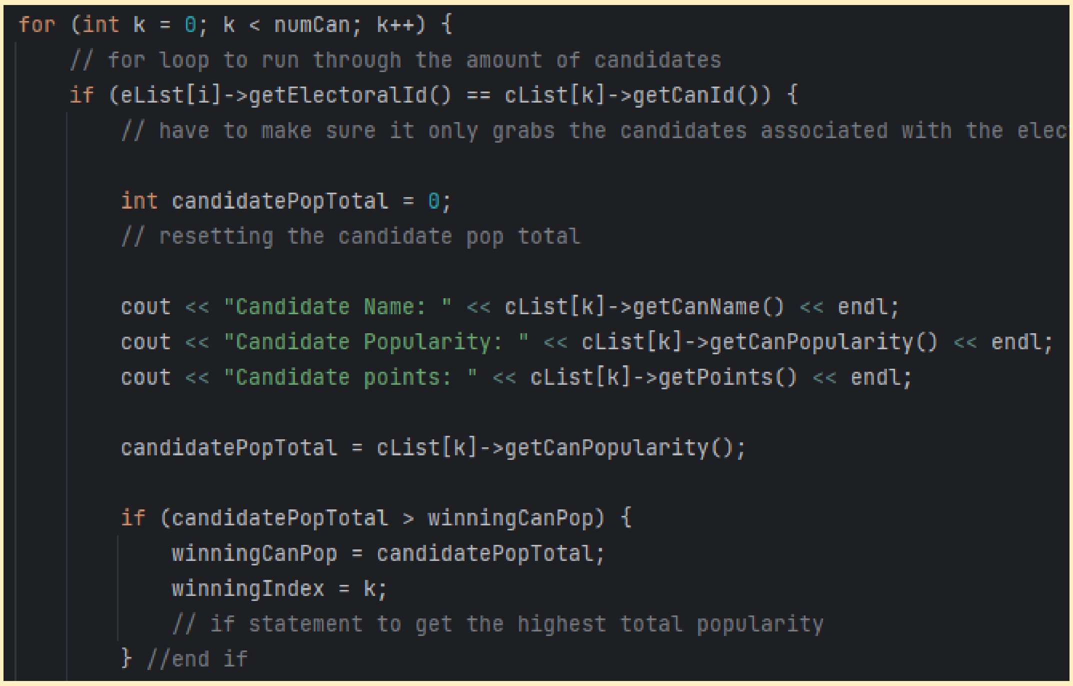
Calculating the Average Stance Euclidean distance and sending the winning index off the global variable

Leader and candidate popularity calculation



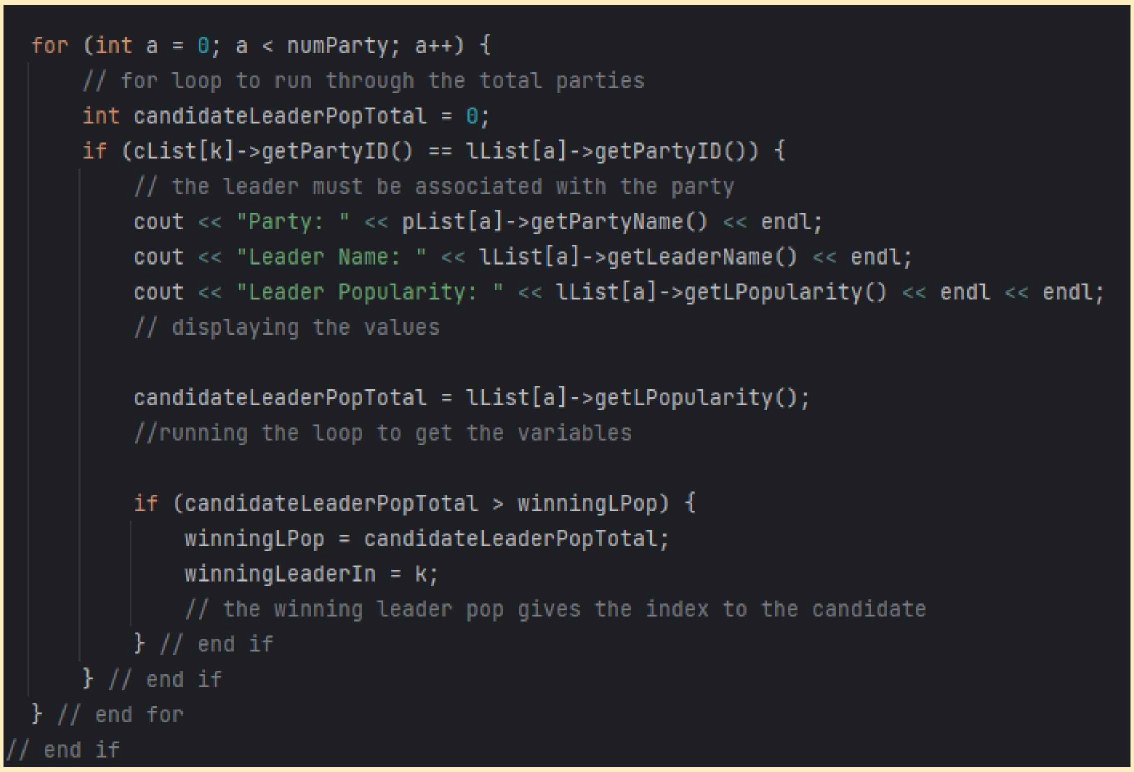
```
for (int i = 0; i < numE; i++) {
    // for loop to run through the amount of electorates
    int winningIndex = -1;
    int winningLeaderIn = -1;
    // the winning index needs to be reset every time
    int winningLPop = 0;
    int winningCanPop = 0;
    // the wining can / leader pop must be reset every time

    cout << "Electorate name: " << eList[i]->getElectorateName() << endl;
```

For loop to run through the electorates



```
for (int k = 0; k < numCan; k++) {
    // for loop to run through the amount of candidates
    if (eList[i]->getElectoralId() == cList[k]->getCanId()) {
        // have to make sure it only grabs the candidates associated with the elec

        int candidatePopTotal = 0;
        // resetting the candidate pop total

        cout << "Candidate Name: " << cList[k]->getCanName() << endl;
        cout << "Candidate Popularity: " << cList[k]->getCanPopularity() << endl;
        cout << "Candidate points: " << cList[k]->getPoints() << endl;

        candidatePopTotal = cList[k]->getCanPopularity();

        if (candidatePopTotal > winningCanPop) {
            winningCanPop = candidatePopTotal;
            winningIndex = k;
            // if statement to get the highest total popularity
        } //end if
```

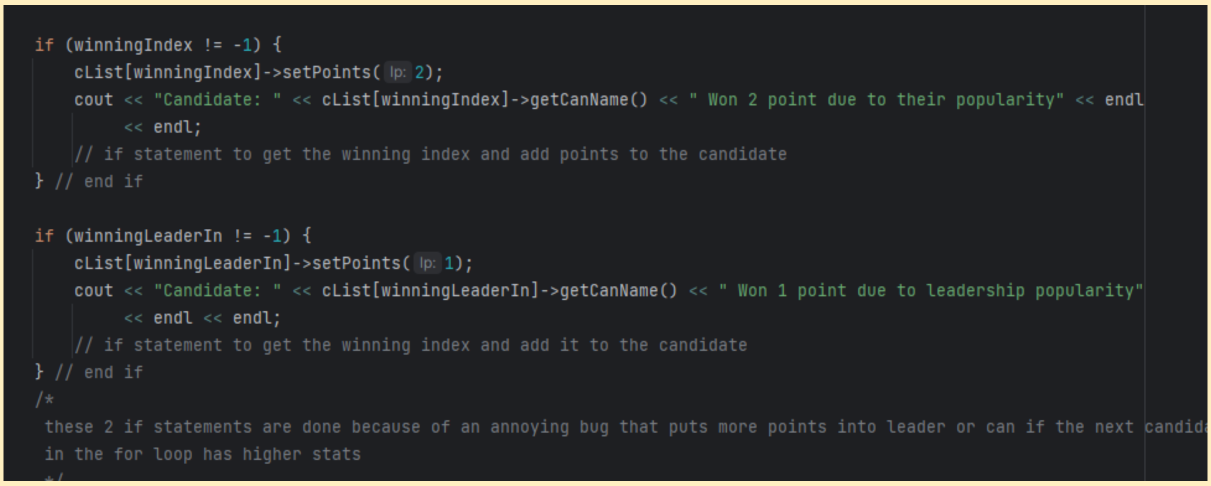
For loop to run through the candidates and get the highest popularity, this is per electorate.



```cpp
for (int a = 0; a < numParty; a++) {
    // for loop to run through the total parties
    int candidateLeaderPopTotal = 0;
    if (cList[k]->getPartyID() == lList[a]->getPartyID()) {
        // the leader must be associated with the party
        cout << "Party: " << pList[a]->getPartyName() << endl;
        cout << "Leader Name: " << lList[a]->getLeaderName() << endl;
        cout << "Leader Popularity: " << lList[a]->getLPopularity() << endl << endl;
        // displaying the values

        candidateLeaderPopTotal = lList[a]->getLPopularity();
        //running the loop to get the variables

        if (candidateLeaderPopTotal > winningLPop) {
            winningLPop = candidateLeaderPopTotal;
            winningLeaderIn = k;
            // the winning leader pop gives the index to the candidate
        } // end if
    } // end if
} // end for
// end if
```

This is a for loop to run through the leaders to get the highest popularity. I prob should of done numLeader but I figured they are always 3

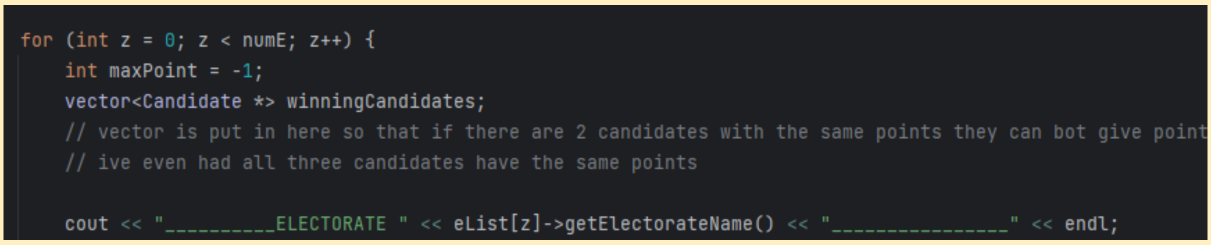
This gets the highest number and saves the index to the global variable



```cpp
if (winningIndex != -1) {
    cList[winningIndex]->setPoints( lp: 2);
    cout << "Candidate: " << cList[winningIndex]->getCanName() << " Won 2 point due to their popularity" << endl
         << endl;
    // if statement to get the winning index and add points to the candidate
} // end if

if (winningLeaderIn != -1) {
    cList[winningLeaderIn]->setPoints( lp: 1);
    cout << "Candidate: " << cList[winningLeaderIn]->getCanName() << " Won 1 point due to leadership popularity"
         << endl << endl;
    // if statement to get the winning index and add it to the candidate
} // end if
/*
 these 2 if statements are done because of an annoying bug that puts more points into leader or can if the next candida
 in the for loop has higher stats
 */
```

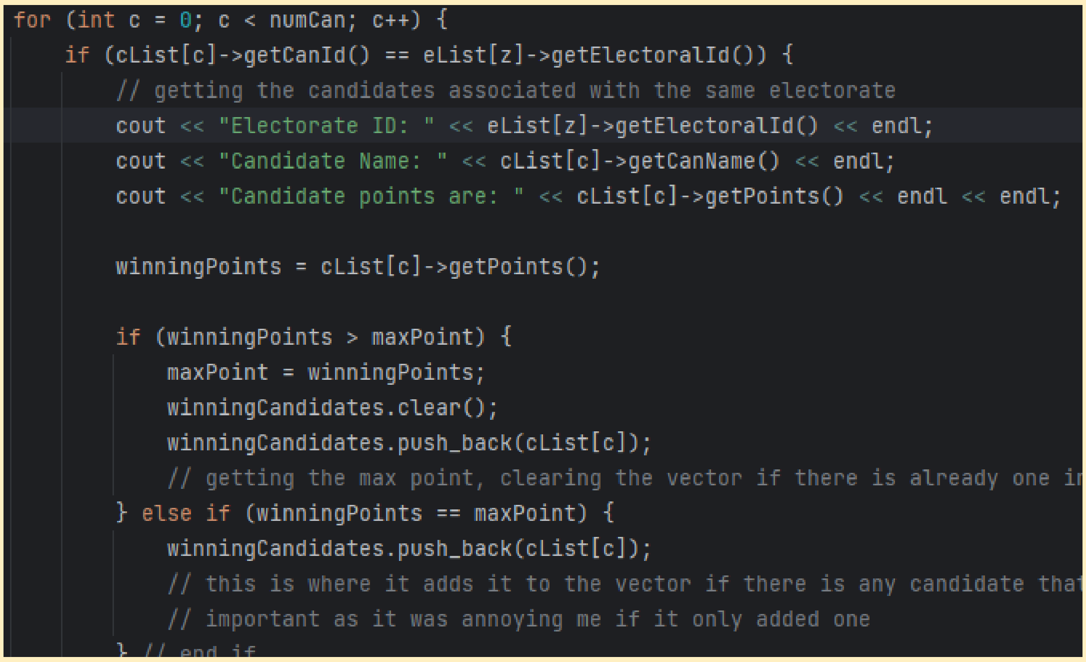
These are the if statements that are called to add points to the candidate for the highest popularity

Electorate Winner calculation



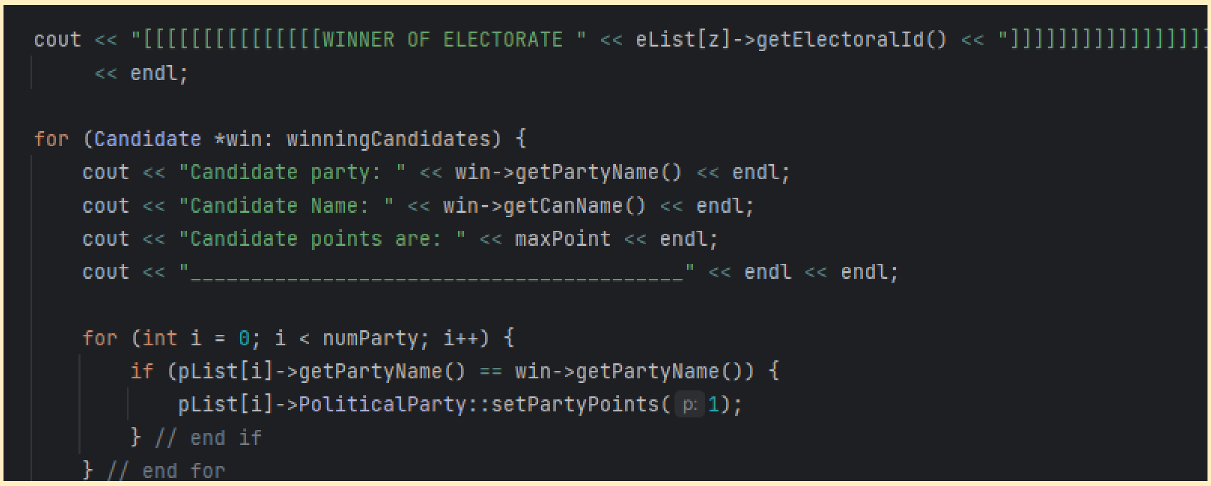
```
for (int z = 0; z < numE; z++) {
    int maxPoint = -1;
    vector<Candidate *> winningCandidates;
    // vector is put in here so that if there are 2 candidates with the same points they can bot give point
    // ive even had all three candidates have the same points

    cout << "__________ELECTORATE " << eList[z]->getElectorateName() << "_______________" << endl;
```

For loop to run through the electorates. It also has a vector in there to add multiple winners. I have even had all of the candidates have matching points of 6 apiece



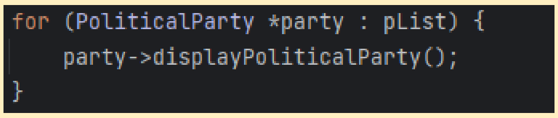
```
for (int c = 0; c < numCan; c++) {
    if (cList[c]->getCanId() == eList[z]->getElectoralId()) {
        // getting the candidates associated with the same electorate
        cout << "Electorate ID: " << eList[z]->getElectoralId() << endl;
        cout << "Candidate Name: " << cList[c]->getCanName() << endl;
        cout << "Candidate points are: " << cList[c]->getPoints() << endl << endl;

        winningPoints = cList[c]->getPoints();

        if (winningPoints > maxPoint) {
            maxPoint = winningPoints;
            winningCandidates.clear();
            winningCandidates.push_back(cList[c]);
            // getting the max point, clearing the vector if there is already one in
        } else if (winningPoints == maxPoint) {
            winningCandidates.push_back(cList[c]);
            // this is where it adds it to the vector if there is any candidate that
            // important as it was annoying me if it only added one
        } // end if
```

For loop to go through the candidates and calculate the winning points, the winners will be added to the vector.



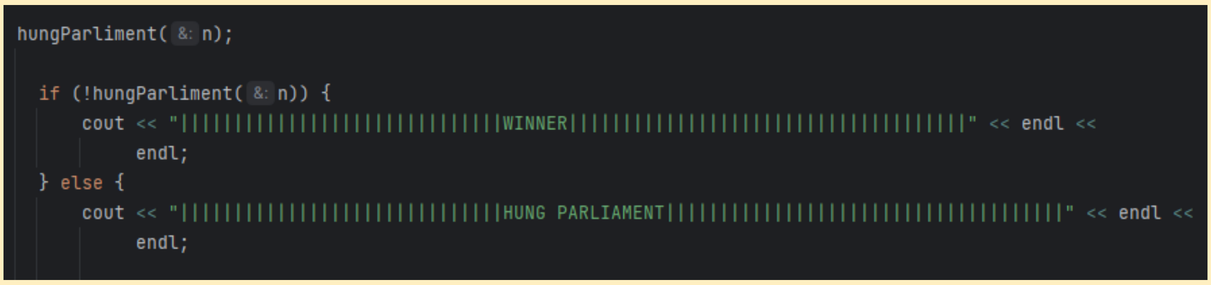
```
cout << "[[[[[[[[[[[[[[[WINNER OF ELECTORATE " << eList[z]->getElectoralId() << "]]]]]]]]]]]]]]]
     << endl;

for (Candidate *win: winningCandidates) {
    cout << "Candidate party: " << win->getPartyName() << endl;
    cout << "Candidate Name: " << win->getCanName() << endl;
    cout << "Candidate points are: " << maxPoint << endl;
    cout << "_________________________________________" << endl << endl;

    for (int i = 0; i < numParty; i++) {
        if (pList[i]->getPartyName() == win->getPartyName()) {
            pList[i]->PoliticalParty::setPartyPoints( p: 1);
        } // end if
    } // end for
```

Once the for loop ends I display the details via a for loop then set the points to the political party associated with the candidate.

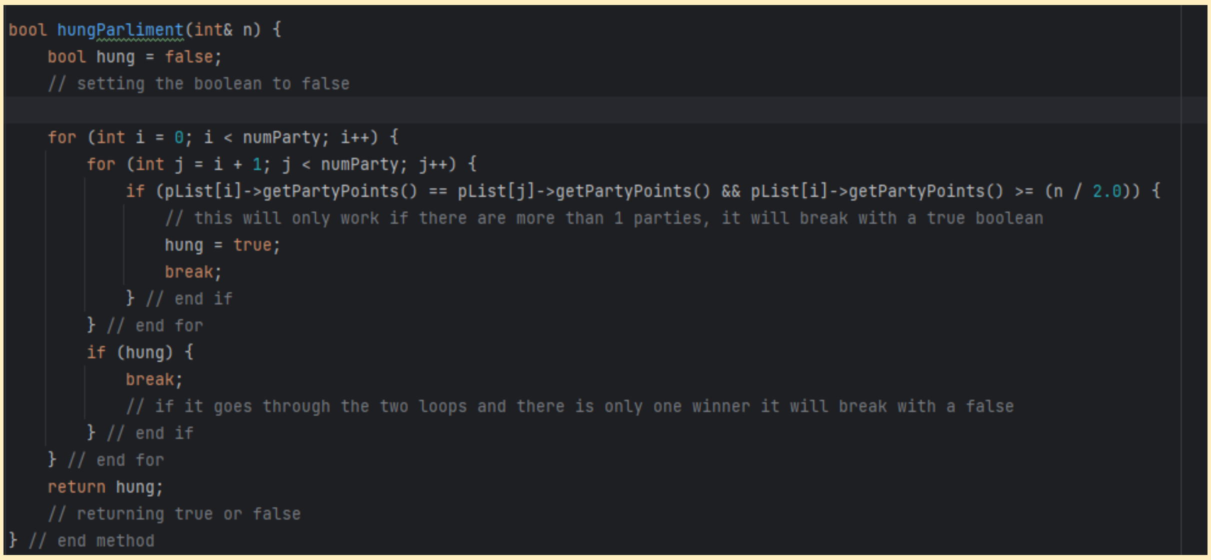


```
for (PoliticalParty *party : pList) {
    party->displayPoliticalParty();
}
```

Displaying the parties



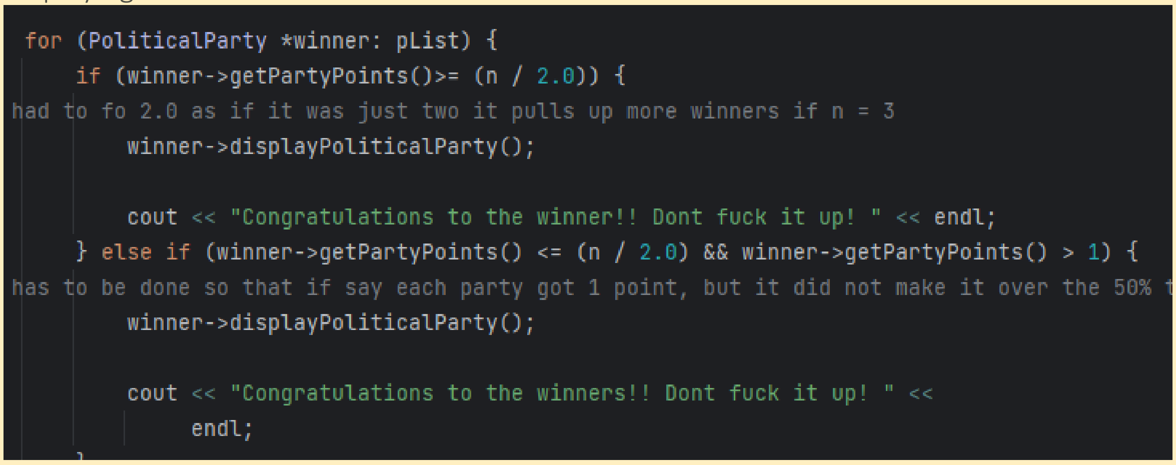
```
hungParliment( &: n);

  if (!hungParliment( &: n)) {
      cout << "||||||||||||||||||||||||||||||WINNER|||||||||||||||||||||||||||||||||||||" << endl <<
          endl;
  } else {
      cout << "||||||||||||||||||||||||||||||HUNG PARLIAMENT|||||||||||||||||||||||||||||||||||||" << endl <<
          endl;
```



```
bool hungParliment(int& n) {
    bool hung = false;
    // setting the boolean to false

    for (int i = 0; i < numParty; i++) {
        for (int j = i + 1; j < numParty; j++) {
            if (pList[i]->getPartyPoints() == pList[j]->getPartyPoints() && pList[i]->getPartyPoints() >= (n / 2.0)) {
                // this will only work if there are more than 1 parties, it will break with a true boolean
                hung = true;
                break;
            } // end if
        } // end for
        if (hung) {
            break;
            // if it goes through the two loops and there is only one winner it will break with a false
        } // end if
    } // end for
    return hung;
    // returning true or false
} // end method
```

I used a Boolean method to display the output of either winner or hung parliament. It it designed so that it runs a nested for loop to check for 2 parties or more who have the number of points needed (50% of electorates) i.e. there are 2 electorates and two parties win one point.

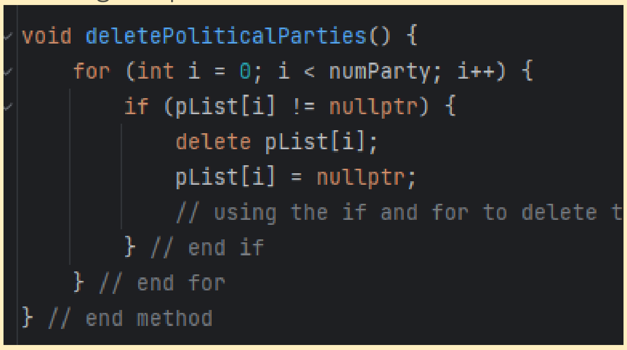
Displaying the winner



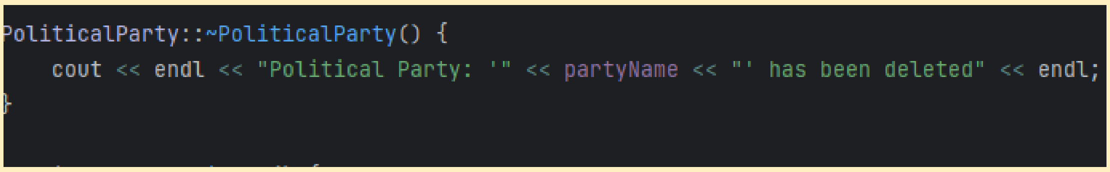
```
 for (PoliticalParty *winner: pList) {
     if (winner->getPartyPoints()>= (n / 2.0)) {
had to fo 2.0 as if it was just two it pulls up more winners if n = 3
         winner->displayPoliticalParty();

         cout << "Congratulations to the winner!! Dont fuck it up! " << endl;
     } else if (winner->getPartyPoints() <= (n / 2.0) && winner->getPartyPoints() > 1) {
has to be done so that if say each party got 1 point, but it did not make it over the 50% t
         winner->displayPoliticalParty();

         cout << "Congratulations to the winners!! Dont fuck it up! " <<
             endl;
```

Political party winner displayed

Deleting the pointers



```
void deletePoliticalParties() {
    for (int i = 0; i < numParty; i++) {
        if (pList[i] != nullptr) {
            delete pList[i];
            pList[i] = nullptr;
            // using the if and for to delete t
        } // end if
    } // end for
} // end method
```



```
PoliticalParty::~PoliticalParty() {
    cout << endl << "Political Party: '" << partyName << "' has been deleted" << endl;
}
```

Running through all the arrays to delete the pointers so the de-constructor can do its job