CMPR 121 - Final Project

TH[3]: Cris Morales, Julian Pedroza, Nathan Stevens

Fall 2022

Santiago Canyon College
Professor Ahmed Alwaheiby

Table of Contents

Summary

The software written by our team and contained in this report has been created to facilitate an electoral process. Through the use of various C++ concepts including strings, classes, objects, arrays, and many more not listed here, we have been able to accomplish just that. Essentially, the code functions as an election between different candidates running for the position of the presidency of student government in Orange County. With this in mind, our code includes the functionality necessary to simulate a competition between numerous candidates across a number of different schools. This would require our code not only to keep track of the vote totals from each of the four schools included in the election, but of the total votes for each candidate across the district as a whole. Furthermore, the ability to display user requested data is also included. Of course, this means that the user is actively interacting with the code, as they are able to request data including the names of all of the candidates, the vote totals by campus, a candidate's entire vote totals, the winner of the election, and the final results of the entire election.

Introduction

Project Objective:

While this Final Project is required to pass this class, it was also necessary for other reasons. The project aims to give us a sample of what a real-world experience would be like, were we to become professional programmers. This project allowed us to collaborate with our peers to accomplish the task. Giving us experience in team-building practices, such as cooperation, respect, communication, coordination, thought exchange, delegation, strategizing, responsibilities allocation, support, and leadership. These teamwork skills gained can and will be applied to any future career, whether as a programmer or not. It helps guide us toward becoming contributing members of our workplaces & society.

Software Goal:

Elections for the Student Government President, the student union's leading-rank officer, must perform precisely. The student performing in this role typically has diverse duties and decision-making powers and generally serves a ceremonial and managerial purpose. This software intends to mainly calculate who the following student government president will be after the assemblage of election data. The data gathered is from four colleges in Orange County; Santiago Canyon, Santa Ana, Orange Coast, and Coastline Community. Students who wish to run for the presidency will pursue student votes from these campuses. The program's functionality provides the user with the following: a list of all candidates running for the Presidential position, a specific candidate's campus-or-total votes, the election winner (aka the president-elect), and the overall final results for all candidates.

Software Description

Overview:

The software displays a menu for the user's convenience. Users can type specific answers through the interface to get their desired data. However, the software does not allow users to access the election's data collection directly. Nonetheless, the users have a handful of options limited to displaying the election data.

Assumptions:

The current version of the program assumes that users won't enter inaccurate information and that the already existing data is accurate. Consequently, the program in its current iteration does not perform any validation checks.

Software Requirements

Hardware Requirements:

Extensive testing would be required to see the minimum process requirements, but this program runs in the console on less than 1 MB of memory in its thread, and uses a fraction of a percent of CPU power. It's safe to say most anything could run this program, as long as it is on a valid OS.

The program is currently compiled for Windows 10 and 11, but it could be compiled to run in linux, Mac OS, or any OS that has support for c++.

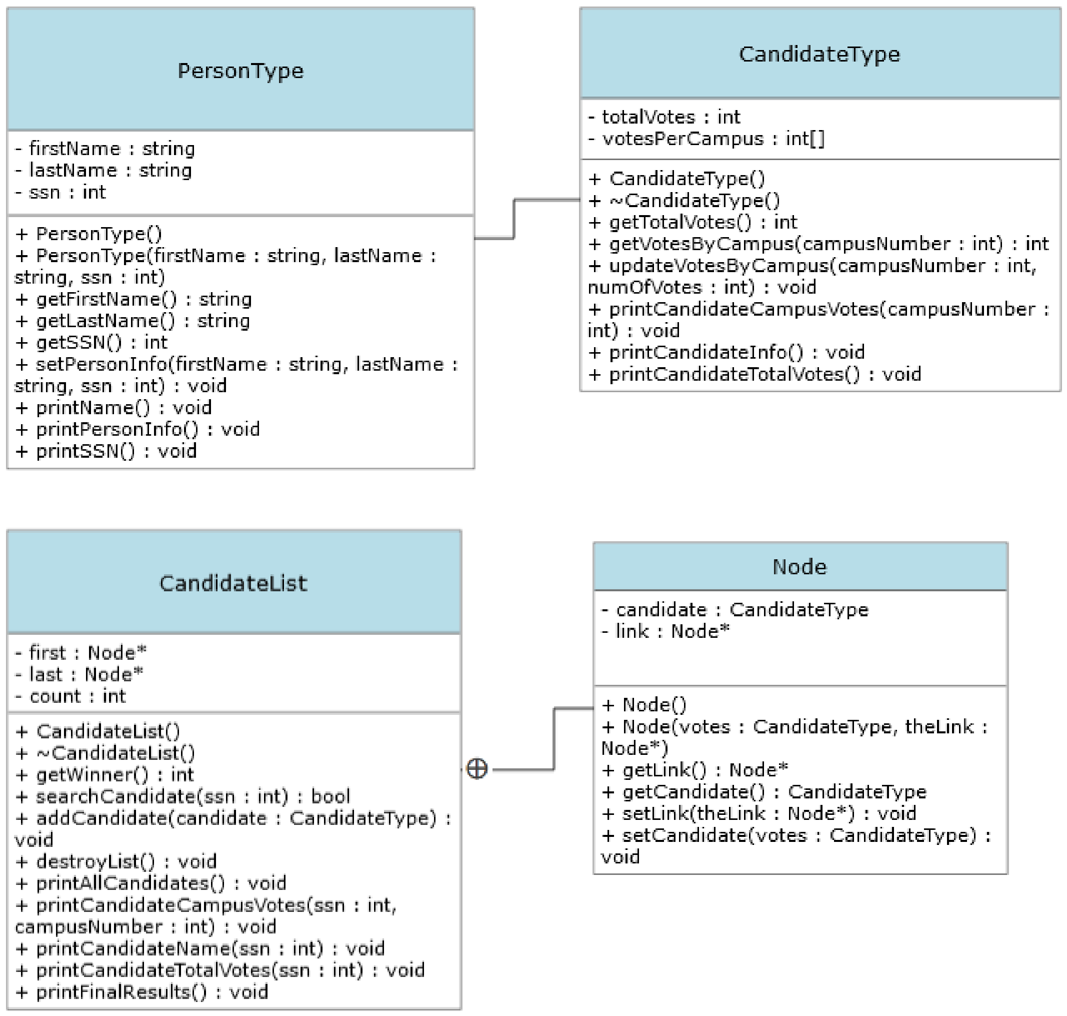
Software Requirements:

This software will print out all of the candidates currently in the election, with info about their vote counts. It will print out a specific candidate's information, including their votes per campus, and their total votes. It will print the winner of the election, and print the final votes for all candidates. This runs in the console, and requires an input file that is specifically formatted in order to read the candidate data.

This program isn't particularly secure, but that's beyond the scope of this class. All input is assumed to have been validated as good already. However, this program scales linearly with an increased number of schools and candidates. It doesn't require much memory per Candidate to store information, about 100 bytes per candidate. Searching the list will always be O(n) search time.

System Design and Implementation

System Design:


PersonType
- firstName : string
- lastName : string
- ssn : int
+ PersonType()
+ PersonType(firstName : string, lastName : string, ssn : int)
+ getFirstName() : string
+ getLastName() : string
+ getSSN() : int
+ setPersonInfo(firstName : string, lastName : string, ssn : int) : void
+ printName() : void
+ printPersonInfo() : void
+ printSSN() : void
CandidateType
- totalVotes : int
- votesPerCampus : int[]
+ CandidateType()
+ ~CandidateType()
+ getTotalVotes() : int
+ getVotesByCampus(campusNumber : int) : int
+ updateVotesByCampus(campusNumber : int, numOfVotes : int) : void
+ printCandidateCampusVotes(campusNumber : int) : void
+ printCandidateInfo() : void
+ printCandidateTotalVotes() : void
CandidateList
- first : Node*
- last : Node*
- count : int
+ CandidateList()
+ ~CandidateList()
+ getWinner() : int
+ searchCandidate(ssn : int) : bool
+ addCandidate(candidate : CandidateType) : void
+ destroyList() : void
+ printAllCandidates() : void
+ printCandidateCampusVotes(ssn : int, campusNumber : int) : void
+ printCandidateName(ssn : int) : void
+ printCandidateTotalVotes(ssn : int) : void
+ printFinalResults() : void
Node
- candidate : CandidateType
- link : Node*
+ Node()
+ Node(votes : CandidateType, theLink : Node*)
+ getLink() : Node*
+ getCandidate() : CandidateType
+ setLink(theLink : Node*) : void
+ setCandidate(votes : CandidateType) : void

Algorithm Design:

The vote count system is based on a linked list containing each candidate's information, represented as a CandidateType. We used multiple classes and inheritance to make representing each candidate in text simple and easy.

Time-complexity analysis:

PersonType : O(1)

- No arrays or loops

CandidateType : O(1)

- No loops; can only access one element of the array at a time

CandidateList : O(n)

- Several functions loop through the whole list

Node : O(1)

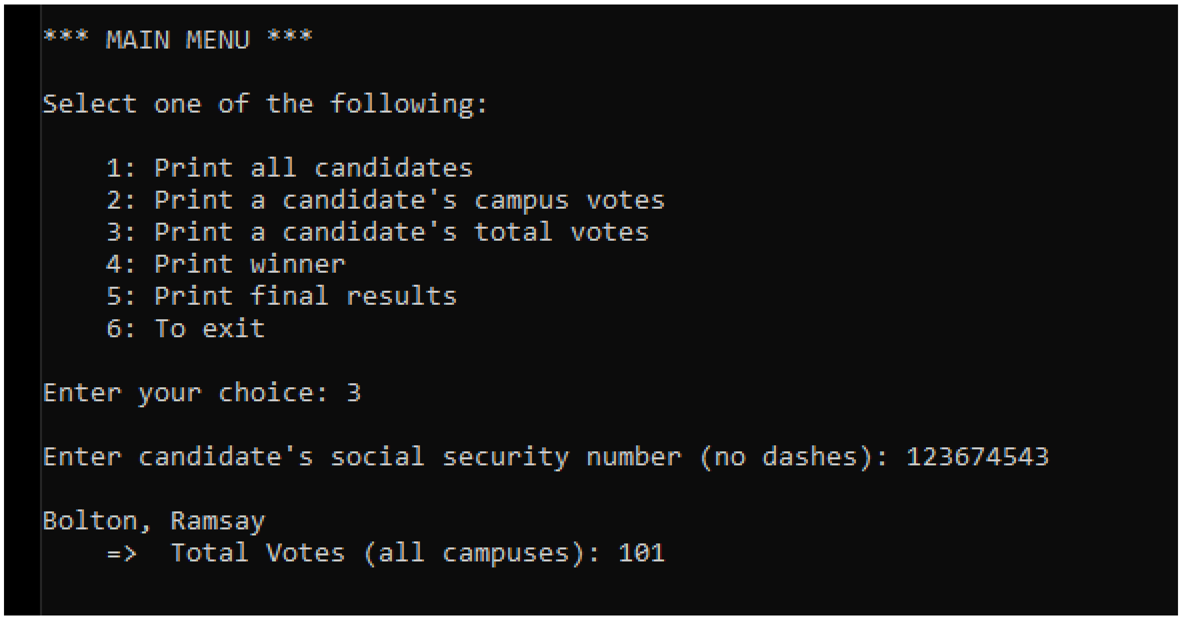
- No loops or arrays

Testing and Results

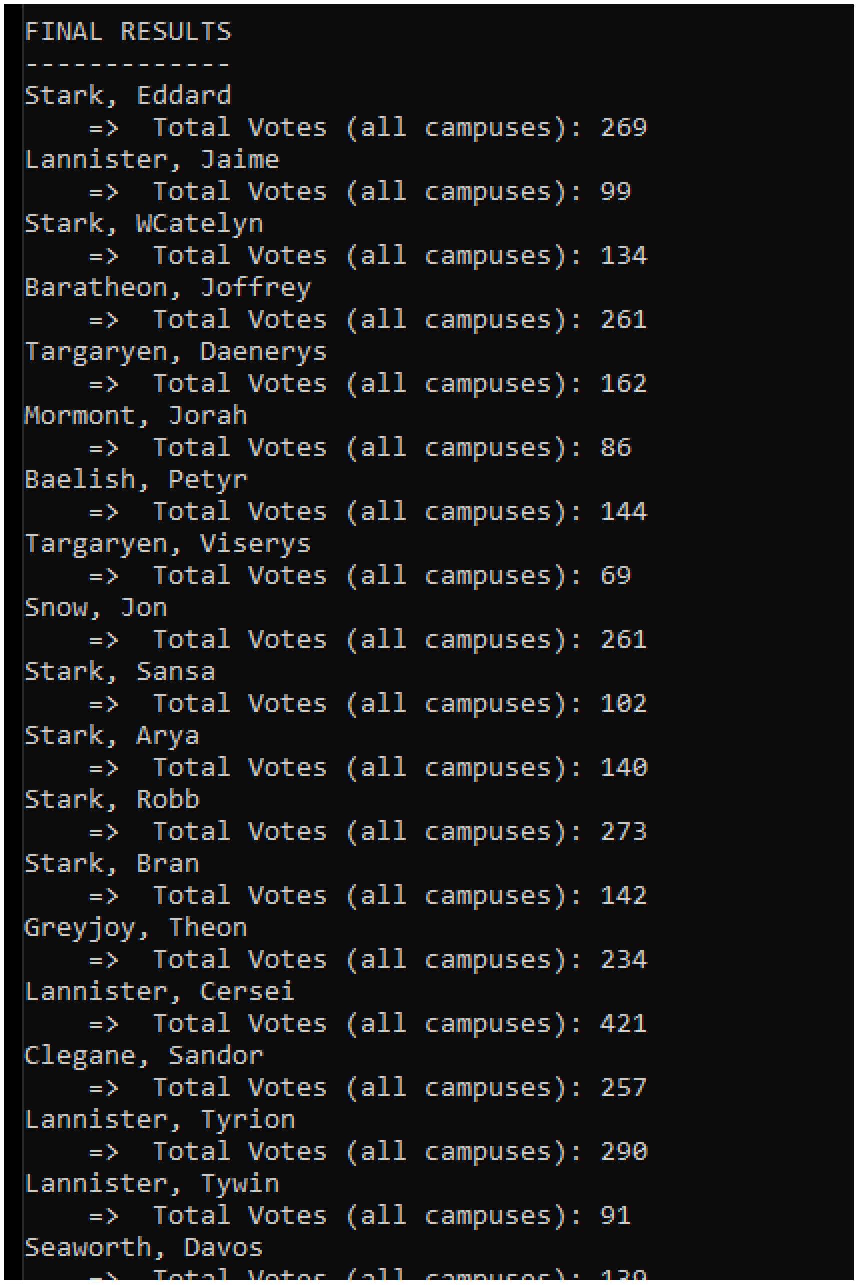
Testing:

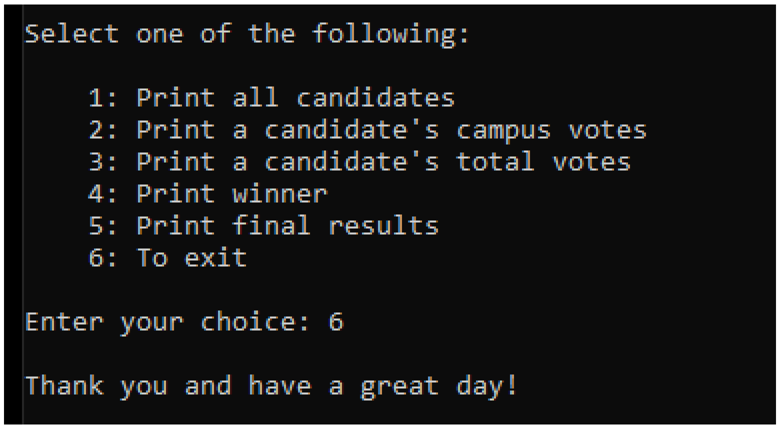
We tested each class and function to make sure it worked as expected. The ones we had to check most carefully were the CandidateList functions that looped through the whole list. Those were the most difficult to grasp conceptually, but we verified that the loop is well-formed and doesn't close early. As we worked on the project, assuming user input would be perfect, there was no need to try and break the code through invalid inputs. The working code's results can be seen in the next section, where the program behaves as expected and responds appropriately to each user input.

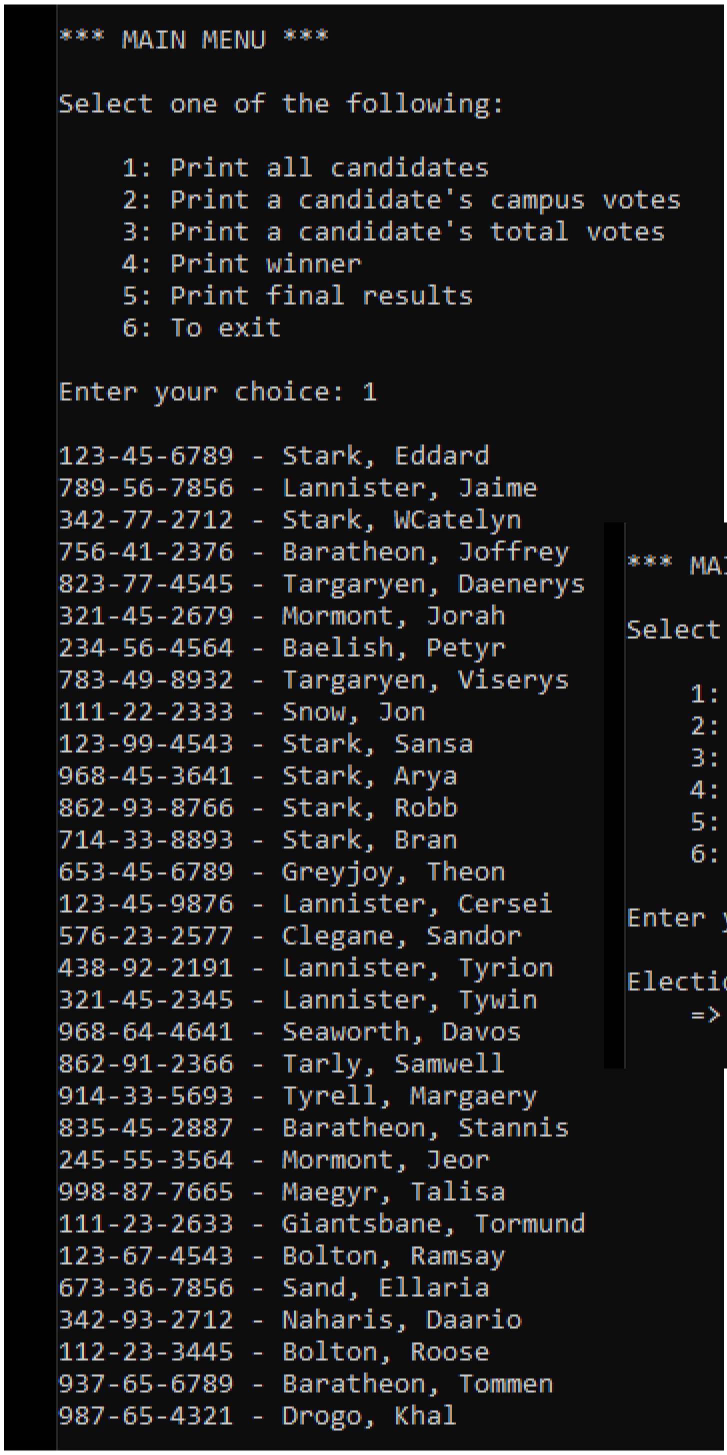
Results:



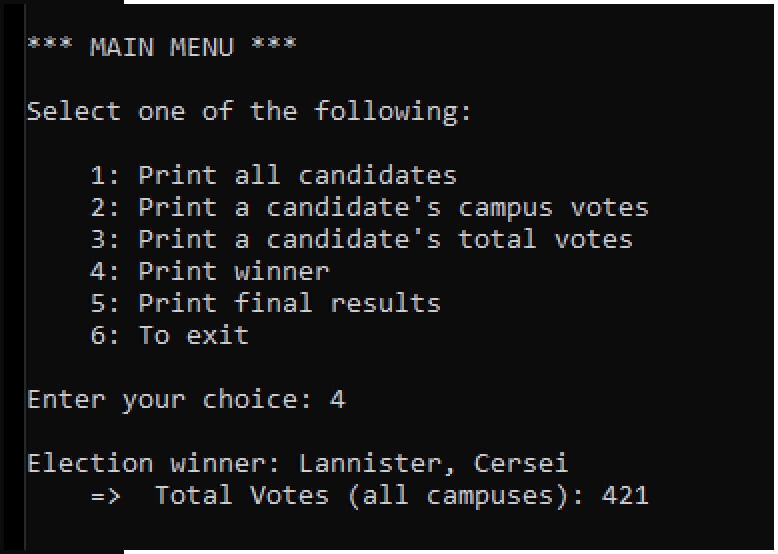
```
*** MAIN MENU ***

Select one of the following:

    1: Print all candidates
    2: Print a candidate's campus votes
    3: Print a candidate's total votes
    4: Print winner
    5: Print final results
    6: To exit

Enter your choice: 3

Enter candidate's social security number (no dashes): 123674543

Bolton, Ramsay
    =>  Total Votes (all campuses): 101
```


```
FINAL RESULTS
-------------
Stark, Eddard
    =>  Total Votes (all campuses): 269
Lannister, Jaime
    =>  Total Votes (all campuses): 99
Stark, WCatelyn
    =>  Total Votes (all campuses): 134
Baratheon, Joffrey
    =>  Total Votes (all campuses): 261
Targaryen, Daenerys
    =>  Total Votes (all campuses): 162
Mormont, Jorah
    =>  Total Votes (all campuses): 86
Baelish, Petyr
    =>  Total Votes (all campuses): 144
Targaryen, Viserys
    =>  Total Votes (all campuses): 69
Snow, Jon
    =>  Total Votes (all campuses): 261
Stark, Sansa
    =>  Total Votes (all campuses): 102
Stark, Arya
    =>  Total Votes (all campuses): 140
Stark, Robb
    =>  Total Votes (all campuses): 273
Stark, Bran
    =>  Total Votes (all campuses): 142
Greyjoy, Theon
    =>  Total Votes (all campuses): 234
Lannister, Cersei
    =>  Total Votes (all campuses): 421
Clegane, Sandor
    =>  Total Votes (all campuses): 257
Lannister, Tyrion
    =>  Total Votes (all campuses): 290
Lannister, Tywin
    =>  Total Votes (all campuses): 91
Seaworth, Davos
    =>  Total Votes (all campuses): 139
```


```
Select one of the following:

    1: Print all candidates
    2: Print a candidate's campus votes
    3: Print a candidate's total votes
    4: Print winner
    5: Print final results
    6: To exit

Enter your choice: 6

Thank you and have a great day!
```



```
*** MAIN MENU ***

Select one of the following:

    1: Print all candidates
    2: Print a candidate's campus votes
    3: Print a candidate's total votes
    4: Print winner
    5: Print final results
    6: To exit

Enter your choice: 1

123-45-6789 - Stark, Eddard
789-56-7856 - Lannister, Jaime
342-77-2712 - Stark, WCatelyn
756-41-2376 - Baratheon, Joffrey
823-77-4545 - Targaryen, Daenerys
321-45-2679 - Mormont, Jorah
234-56-4564 - Baelish, Petyr
783-49-8932 - Targaryen, Viserys
111-22-2333 - Snow, Jon
123-99-4543 - Stark, Sansa
968-45-3641 - Stark, Arya
862-93-8766 - Stark, Robb
714-33-8893 - Stark, Bran
653-45-6789 - Greyjoy, Theon
123-45-9876 - Lannister, Cersei
576-23-2577 - Clegane, Sandor
438-92-2191 - Lannister, Tyrion
321-45-2345 - Lannister, Tywin
968-64-4641 - Seaworth, Davos
862-91-2366 - Tarly, Samwell
914-33-5693 - Tyrell, Margaery
835-45-2887 - Baratheon, Stannis
245-55-3564 - Mormont, Jeor
998-87-7665 - Maegyr, Talisa
111-23-2633 - Giantsbane, Tormund
123-67-4543 - Bolton, Ramsay
673-36-7856 - Sand, Ellaria
342-93-2712 - Naharis, Daario
112-23-3445 - Bolton, Roose
937-65-6789 - Baratheon, Tommen
987-65-4321 - Drogo, Khal
```



```
*** MAIN MENU ***

Select one of the following:

    1: Print all candidates
    2: Print a candidate's campus votes
    3: Print a candidate's total votes
    4: Print winner
    5: Print final results
    6: To exit

Enter your choice: 4

Election winner: Lannister, Cersei
    =>  Total Votes (all campuses): 421
```

Final Deliverables

Documentation

<u>PersonType class</u>
Member Variables:
firstName - A person's first name stored as a string
lastName - A person's last name stored as a string
ssn - A person's social security number stored as an int

Default Constructor:
Initializes the social security number to a default value of 0.
No need to initialize strings because they are objects and have a default constructor

Overloaded Constructor
Parameters: first name, last name, and social security
number.
Initializes all member variables to the give values.

Function: *setPersonInfo*
Parameters: first name, last name, and social security
number.
Re-sets the first name, the last name, and the social
security number of a person to the new values passed.

Function: *getFirstName*
Returns the person's first name

Function: *getLastName*
Returns the person's last name

Function: *getSSN*
Returns the person's social security number.

Function: *printName*
Prints the person's last and first name in the following
format:
lName, fName

Function: *printPersonInfo*
Calls the function printSSN() to format the social security number. Prints the person's social security number, first name and last name in the following format

###-##-#### fName lName

Function: *printSSN*
Formats the social security number by separating the number with dashes and outputs the formatted string.

<u>CandidateType class</u>
Global constant: *NUM_OF_CAMPUSES*

Member variables
totalVotes - An integer that stores the total number of votes
votesPerCampus - An array of integers that has capacity of NUM_Of_CAMPUSES.

The campuses will be labeled by numbers (i.e., Santiago Canyon College = 1, Santa Ana College = 2, and so on) where one corresponds to index 0 of the array, 2 corresponds to index 1 of the array, etc.

Default constructor
Initializes the total number of votes to 0

Function: *updateVotesByCampus*
Parameters: The Campus number and the number of votes for that campus.
Updates the total number of votes and the number of votes for the campus specified

Function: *getTotalVotes*
Return the total number of votes

Function: *getVotesByCampus*
Parameter: The Campus number.
Returns the votes for the specified Campus.

Function: *printCandidateInfo*
Prints information about the candidate in the following format:

###-##-#### - lName, fName

Note: the “#” should be replaced with integers.

Function: *printCandidateTotalVotes*
Prints the candidate’s total votes in the following format:

Lastname, Firstname
Total Votes (all campuses) : #

Note that “#” should be replaced with an integer.

Function: *printCandidateCampusVotes*
Parameters: The Campus number.
Prints the candidate’s vote for the specified Campus:

Lastname, Firstname
Campus # total votes: ##
Note that “#” should be replaced with an integer.

<u>CandidateList Class</u>
Member variables
first - A pointer named first that points to the first node.
last - A pointer named last that points to the last node.
count - An integer variable named count that stores the number of nodes in the list.

Default Constructor
Initializes all member variables.

Function *addCandidate*
Parameters: An object of the CandidateType class.
Inserts nodes to the back of the list. You have a pointer pointing to the back of the list; therefore, there is NO need to traverse the list.

Function *getWinner*
Traverses the list to find the candidate who has the highest number of votes, and returns the social security number associated with that candidate.
If the list is empty, output the error message:
"=> List is empty" and return 0;

Function *searchCandidate*
Parameters: A social security number.
Transverses the list to find the candidate with the given social security number and returns true if the candidate is found and false otherwise.
Use a while loop so that you can stop when the candidate is found -> You are not allowed to use "break" or "continue".
If the list is empty, output the error message
"=> List is empty".
If the candidate was not found, output the error message:
"=> SSN not in the list".

Function *printCandidateName*
Parameters: A social security number.
Traverse the list to find the candidate with the given social security number and prints out the name using the printName function of the PersonType class.
Use a while loop so that you can stop the loop when the candidate is found -> you are NOT allowed to use "break" or "continue".
If the list is empty, output the error message "=>l List is empty."
If the candidate was not found, output the error message "=> SSN not in the list."

Function *printAllCandidates*
Traverse the list to print all candidates using the printCandidateInfo function of the CandidatetType class.
If the list is empty, output the error message "=> List

is empty.”

Function *printCandidateCampusVotes*
Parameters: A social security number and a division number. Prints out all the division votes for a given candidate, using the getVotesByCampus function of the CandidateType class.
Use a while loop so that you can stop the loop when the candidate is found -> You are NOT allowed to use “break” or “continue”
If the list is empty, output the error message “=> List is empty.”

Function *printCandidateTotalVotes*
Parameters: A social security number
Traverses the list to find the candidate with the given social security number and prints out the total number of votes using the getTotalVotes function of the CandidateType class.
Use a while loop so that you can stop the loop when the candidate is found -> You are NOT allowed to use “break” or “continue”
If the list is empty, output the error message “=> List is empty.”

Function *destroyList*
Traverse the list to delete each node and reset all member variables to their default value.

Destructor
Calls the function destroyList.

Source Code:

Main.cpp

```
#include "InputHandler.h"
```

```
using namespace std;
void displayMenu();
void processChoice(CandidateList& candidateList);

int main()
{
   //Create the list
   CandidateList candidateList;
   //fill the list with candidates data
   readCandidateData(candidateList);
   //Make a choice
   displayMenu();
   //Process the choice
   processChoice(candidateList);
   cout << endl;
   system("Pause");
   return 0;
}

void displayMenu()
{
   cout << "\n*** MAIN MENU ***\n";
   cout << "\nSelect one of the following:\n\n";
   cout << "     1: Print all candidates" << endl;
   cout << "     2: Print a candidate's campus votes" << endl;
   cout << "     3: Print a candidate's total votes" << endl;
   cout << "     4: Print winner" << endl;
   cout << "     5: Print final results" << endl;
   cout << "     6: To exit" << endl;
}

void processChoice(CandidateList& candidateList)
{
   int choice;
   cout << "\nEnter your choice: ";
   cin >> choice;
   while (choice != 6)
   {
         string fName, lName;
         int campus = 0,
        ssn = 0;
switch (choice)
{
        // Print all candidates
case 1:
        cout << endl;
        candidateList.printAllCandidates();
        cout << endl;
        break;
        // Print a candidates's campus votes
case 2:
        cout << "\nEnter candidate's social security number (no dashes): ";
                cin >> ssn;
        cout << endl;
        if (candidateList.searchCandidate(ssn))
        {
                candidateList.printCandidateName(ssn);
                cout << endl;
                for (int i = 1; i <= NUM_OF_CAMPUSES; ++i)
                        candidateList.printCandidateCampusVotes(ssn, i);
        }
        cout << endl;
        break;
        // Print a candidate's total votes
case 3:
        cout << "\nEnter candidate's social security number (no dashes): ";
                cin >> ssn;
        cout << endl;
        if (candidateList.searchCandidate(ssn))
        {
                candidateList.printCandidateName(ssn);
                cout << endl;
                candidateList.printCandidateTotalVotes(ssn);
        }
        cout << endl << endl;
        break;
        // Print winner
case 4:
        ssn = candidateList.getWinner();
        if (ssn != 0)
```

```
				{
					cout << "\nElection winner: ";
					candidateList.printCandidateName(ssn);
					cout << endl;
					candidateList.printCandidateTotalVotes(ssn);
				}
				else
				{
					cout << "\n	=> There are no candidates.";
				}
				cout << endl << endl;
				break;
		case 5: // prints total votes and name of each candidate
				cout << endl;
				cout << "FINAL RESULTS" << endl;
				cout << "-------------" << endl;
				candidateList.printFinalResults(); // function doesn't exist
				cout << endl;
				break;
		default:
				cout << "\n	=> Sorry. That is not a selection. \n";
				cout << endl;
		}
		cout << endl;
		displayMenu();
		cout << "\nEnter your choice: ";
		cin >> choice;
	}
	if (choice == 6)
		cout << "\nThank you and have a great day!" << endl;
}
```

PersonType.h

```
#ifndef PERSONTYPE_H
#define PERSONTYPE_H

#include <string>
```

```
class PersonType
{
private:
    std::string firstName; // first name
    std::string lastName; // last name
    int ssn; // social security number

public:

    // Constructors

    PersonType();

    PersonType(const std::string& firstName, const std::string& lastName, int ssn);


    // Accessors

    const std::string& getFirstName() const;

    const std::string& getLastName() const;

    int getSSN() const;


    // Mutators

    void setPersonInfo(const std::string& firstName, const std::string& lastName, int ssn);


    // Methods

    void printName() const;

    void printPersonInfo() const;

    void printSSN() const;


    // Destructor
    ~PersonType() {}

};

#endif
```

CandidateType.h

```
#ifndef CANDIDATETYPE_H
#define CANDIDATETYPE_H

#include "PersonType.h"

const int NUM_OF_CAMPUSES = 4;

class CandidateType : public PersonType
{
private:

    int totalVotes; // total votes for the candidate
    int votesPerCampus[NUM_OF_CAMPUSES]; // array of votes per campus

public:

    // Constructors

    CandidateType();


    // Destructor

    ~CandidateType() {}


    // Accessors

    int getTotalVotes() const;
    int getVotesByCampus(int campusNumber) const;


    // Mutators

    void updateVotesByCampus(int campusNumber, int numOfVotes);


    // Members

    void printCandidateCampusVotes(int campusNumber) const;

    void printCandidateInfo() const;

    void printCandidateTotalVotes() const;

};

#endif // end CANDIDATETYPE_H
```

CandidateList.h

```
#ifndef CANDIDATELIST_H
#define CANDIDATELIST_H

#include "CandidateType.h"

class CandidateList
{
    class Node
    {
    public:
            Node() : link(nullptr) {}
            Node(const CandidateType& votes, Node* theLink)
                    : candidate(votes), link(theLink) {}
            Node* getLink() const { return link; }
            CandidateType getCandidate() const { return candidate; }
            void setCandidate(const CandidateType& votes) { candidate = votes; }
```

```
        void setLink(Node* theLink) { link = theLink; }
        ~Node() {}
    private:
        CandidateType candidate;
        Node* link; //pointer that points to next node
    };

private:

    Node* first;
    Node* last;
    int count;

public:

    //Constructor

    CandidateList();


    // Destructor

    ~CandidateList();


    // Accessors

    int getWinner() const;

    bool searchCandidate(int ssn) const;


    // Mutators

    void addCandidate(const CandidateType& candidate);

    void destroyList();


    // Members
    void printAllCandidates() const;

    void printCandidateCampusVotes(int ssn, int campusNumber) const;

    void printCandidateName(int ssn) const;

    void printCandidateTotalVotes(int ssn) const;

    void printFinalResults() const;

};

#endif // !CANDIDATELIST_H
```

InputHandler.h

```
#ifndef INPUTHANDLER_H
#define INPUTHANDLER_H

#include <fstream>
#include <string>
#include "CandidateList.h"
#include <iostream>
#include <filesystem>

void createCandidateList(std::ifstream& infile, CandidateList& candidateList)
{
    int ssn = 0;
    int allVotes[NUM_OF_CAMPUSES];
    std::string  fName, lName;

    infile >> ssn;

    while (ssn != -999)
    {
        CandidateType newCandidate;

        infile >> fName;
```

```
            infile >> lName;
            newCandidate.setPersonInfo(fName, lName, ssn);

            for (int i = 0; i < NUM_OF_CAMPUSES; ++i)
            {
                    infile >> allVotes[i];
                    newCandidate.updateVotesByCampus(i, allVotes[i]);
            }

            candidateList.addCandidate(newCandidate);

            infile >> ssn;
    }
}

void readCandidateData(CandidateList& candidateList)
{
    std::ifstream infile;

    std::cout << std::filesystem::current_path().string() << std::endl;

    infile.open("candidate_data.txt");

    if (!infile)
    {
            std::cerr << "Input file does not exist." << std::endl;
            exit(1);
    }

    createCandidateList(infile, candidateList);

    infile.close();
}

#endif // !INPUTHANDLER_H
```

PersonType.cpp

```
#include "PersonType.h"
#include <iostream>
#include <string>

// Constructors

PersonType::PersonType()
{
   this->ssn = 0;
   // firstName and lastName are strings, the default string constructor is called
   // this inits those vars to empty strings
}

PersonType::PersonType(const std::string& firstName, const std::string& lastName, int ssn)
{
   this->firstName = firstName;
   this->lastName = lastName;
   this->ssn = ssn;
}


// Accessors

const std::string& PersonType::getFirstName() const
{
   return firstName;
}

const std::string& PersonType::getLastName() const
{
   return lastName;
}

int PersonType::getSSN() const
{
   return ssn;
}


// Mutators
```

```
void PersonType::setPersonInfo(const std::string& firstName, const std::string& lastName, int
ssn)
{
   this->firstName = firstName;
   this->lastName = lastName;
   this->ssn = ssn;
}


// Methods

void PersonType::printName() const
{
   std::cout << lastName << ", " << firstName;
}

void PersonType::printPersonInfo() const
{
   printSSN();
   std::cout << " " << firstName << " " << lastName;
}

void PersonType::printSSN() const
{
   std::string s = std::to_string(ssn);

   s.insert(3, "-");
   s.insert(6, "-");

   std::cout << s;
}
```

CandidateType.cpp

```
#include "CandidateType.h"
#include <string>
#include <iostream>

// Constructor
```

```
CandidateType::CandidateType()
{
   totalVotes = 0;
   for (int i = 0; i < NUM_OF_CAMPUSES; ++i)
   {
         votesPerCampus[i] = 0;
   }
}


// Accessors

int CandidateType::getTotalVotes() const
{
   return totalVotes;
}

int CandidateType::getVotesByCampus(int campusNumber) const
{
   return votesPerCampus[campusNumber - 1];
}


// Mutators

void CandidateType::updateVotesByCampus(int campusNumber, int newNumOfVotes)
{
   int prevNum = votesPerCampus[campusNumber - 1];

   votesPerCampus[campusNumber - 1] = newNumOfVotes;

   totalVotes -= prevNum;
   totalVotes += newNumOfVotes;
}


// Members

void CandidateType::printCandidateCampusVotes(int campusNumber) const
```

```
{
   printName();
   std::cout << std::endl;
   std::cout << "          =>  Campus " << campusNumber << " total votes: " <<
getVotesByCampus(campusNumber) << std::endl;
}

void CandidateType::printCandidateInfo() const
{
   printSSN();
   std::cout << " - ";
   printName();
   std::cout << std::endl;
}

void CandidateType::printCandidateTotalVotes() const
{
   printName();
   std::cout << std::endl;
   std::cout << "          =>  Total Votes (all campuses): " << getTotalVotes() << std::endl;
}
```

CandidateList.cpp

```
#include "CandidateList.h"
#include <iostream>
#include <string>

//Constructor

CandidateList::CandidateList()
{
   first = nullptr;
   last = nullptr;
   count = 0;
}
```

```
// Destructor

CandidateList::~CandidateList()
{
   destroyList();
}


// Accessors

int CandidateList::getWinner() const
{
   if (count == 0)
   {
         std::cout << "=> List is empty" << std::endl;
         return 0; // error code for the driver program
   }

   Node* current = first;
   Node* largest = current;

   while (current != nullptr) // while not at the end of the list
   {
         if (current->getCandidate().getTotalVotes() > largest->getCandidate().getTotalVotes())
         {
                 largest = current;
         }

         current = current->getLink(); // go to next candidate
   }

   return largest->getCandidate().getSSN();
}

bool CandidateList::searchCandidate(int ssn) const
{
   if (count == 0)
   {
         std::cout << "=> List is empty" << std::endl;
         return false;
```

```
    }

    Node* current = first;

    while (current->getCandidate().getSSN() != ssn) // while current isn't the desired candidate
    {
        current = current->getLink(); // go to the next candidate
    }

    if (current == nullptr) // candidate wasn't found (node at end of the array has null link)
    {
        std::cout << "=> SSN not in the list" << std::endl;
        return false;
    }
    else
    {
        return true;
    }
}


// Mutators

void CandidateList::addCandidate(const CandidateType& candidate)
{
    Node* newNode = new Node(candidate, nullptr);

    if (count == 0)
    {
        first = newNode; // only update the first node if list was empty
    }
    else
    {
        last->setLink(newNode); // links old last element to new last element
    }

    last = newNode; // updates last node pointer on the list

    count++;
}
```

```
void CandidateList::destroyList()
{
   Node* toDestroy = first;

   while (toDestroy != nullptr) // can also loop from 0 to i < count
   {
         Node* next = toDestroy->getLink();
         delete toDestroy;
         toDestroy = next;
   }

   first = nullptr;
   last = nullptr;
   count = 0;
}


// Members

void CandidateList::printAllCandidates() const
{
   if (count == 0)
   {
         std::cout << "=> List is empty." << std::endl;
         return;
   }

   Node* current = first;

   while (current != nullptr)
   {
         current->getCandidate().printCandidateInfo();
         current = current->getLink();
   }
}

void CandidateList::printCandidateCampusVotes(int ssn, int campusNumber) const
{
   if (count == 0)
```

```
    {
        std::cout << "=> List is empty" << std::endl;
        return;
    }

    Node* current = first;

    while (current->getCandidate().getSSN() != ssn) // while current isn't the desired candidate
    {
        current = current->getLink(); // go to the next candidate
    }

    if (current == nullptr) // candidate wasn't found (node at end of the array has null link)
    {
        std::cout << "=> SSN not in the list" << std::endl;
        return;
    }

    CandidateType candidate = current->getCandidate();

    std::cout << "        =>  Campus " << campusNumber << " total votes: " <<
candidate.getVotesByCampus(campusNumber) << std::endl;
}

void CandidateList::printCandidateName(int ssn) const
{
    if (count == 0)
    {
        std::cout << "=> List is empty" << std::endl;
        return;
    }

    Node* current = first;

    while (current->getCandidate().getSSN() != ssn) // while current isn't the desired candidate
    {
        current = current->getLink(); // go to the next candidate
    }

    if (current == nullptr) // candidate wasn't found (node at end of the array has null link)
    {
            std::cout << "=> SSN not in the list" << std::endl;
            return;
    }

    CandidateType candidate = current->getCandidate();

    candidate.printName(); // PersonType class function
}

void CandidateList::printCandidateTotalVotes(int ssn) const
{
    if (count == 0)
    {
            std::cout << "=> List is empty" << std::endl;
            return;
    }

    Node* current = first;

    while (current->getCandidate().getSSN() != ssn) // while current isn't the desired candidate
    {
            current = current->getLink(); // go to the next candidate
    }

    if (current == nullptr) // candidate wasn't found (node at end of the array has null link)
    {
            std::cout << "=> SSN not in the list" << std::endl;
            return;
    }

    CandidateType candidate = current->getCandidate();

    std::cout << "          =>  Total Votes (all campuses): " << candidate.getTotalVotes() <<
std::endl;
}

void CandidateList::printFinalResults() const
{
    if (count == 0)
    {
        std::cout << "=> List is empty" << std::endl;
        return;
    }

    Node* current = first;

    while (current != nullptr)
    {
        current->getCandidate().printCandidateTotalVotes();
        current = current->getLink();
    }
}
```

References

Gaddis, Tony. *Starting out with C++, From Control Structures through Objects*. Ninth ed., Pearson Education, Inc., 2018.