Voting Machine Software for Community College President Elections

Overscores
Katelyn McQueen, Dylan Berry, Jay D. Reyes
Fall 2023
Santiago Canyon College - Prof. Ahmed Alweheiby

Table Of Contents

Summary

This software provides a solution for managing the results of an election.

Introduction

Project Aims and Objectives

This software aims to deliver a simple and easy-to-use interface for community college elections in the RSCCD school district. In addition, this was a great showcase for our skill and knowledge as developers in the language C++.

Software problem statement

We were tasked with designing election result display software to a given specification.

Software Description

Software overview

This project reads a text file that is formatted with information of candidates and their votes. The user can then use the provided menu to view the data and find the winner of the election.

Software Assumptions

The one assumption we make is that the data would already be cleaned (e.g. no duplicates and with similar spacing) and in a specific format. In our program, menu choices are tested to be a valid selection.

Software Requirements

Hardware Requirements

The program has been tested on both windows 10 and linux. All other configurations are untested. We believe our code should be portable to any implementation with standard library facilities. The program is a text based cli program processing relatively small datasets, the

performance and memory requirements are therefore small and should be performant on most modern systems.

Software Requirements

- Functional Requirements:
    - Print all candidates
    - Print a candidate's votes by campus
    - Print a candidate's total votes
    - Print the winner
    - Print final results

- Non-Functional Requirements:
    - Reliability - provided you run the software on a Windows 10 or Linux machine built in the last 10 years with C++ on it, you should run in to no issues at all
    - Security - all relevant information is private and only accessible through the appropriate channels given in the program.
    - Scalability - the worst Big O notation is O(n), so it runs on linear time, meaning it will scale quite well with larger and larger lists of candidates. I do imagine it slowing down near the 250 thousand mark, but there should never be more than 100 candidates at a time in the RSCCD school district. Thus, scalability is not an issue.
    - Space: Our program uses up no more than 5 kilobytes of space, so space will not be an issue on any system built in the last 20 years.
    - Ethical Requirements: All data is sourced and handled responsibly with no misuse.

System Design and Implementation

System Design (UML)

PersonType:
  member variables:
    - fName : string - a person's first name
    - lName : string - a person's last name
    - SSN : int - a person's social security number
  member functions:

    + PersonType() : PersonType - default constructor. Sets SSN to 0.

    + PersonType(firstName : string, lastName : string, socialSecurityNumber : int) : PersonType - initializes a person with the given values

+ setPerfonInfo(firstName : string, lastName : string, socialSecurityNumber: int) : void - re-sets the provided values
(perfon is likely a typo, use the (likely) incorrect name from the specification unless and until it's fixed)
(the specification doesn't explicitly call for a return type. I've assumed void, but bool is a possible alternative)

+ getFirstName() const : string - return the person's first name

+ getLastName() const : string - return the person's last name

+ getSSN() const : int - returns the person's social security number

+ printName() : void - print the person's first and last name
(formatted as "lName, fName)
(void is assumed again)

+ printPersonInfo() : void - prints the full set of information about a person
(calls printSSN() to format the social security number)
(formatted as ###-##-#### fName lName)
(void is assumed again)

+ printSSN() : void - formats the social security number and prints the formatted string (###-##-####)
(void is assumed)

+ ~PersonType() - destructor. Left empty

CandidateType:
inherits from PersonType
member variables:
- totalVote : int - total number of votes
- campusVotes : int[NUM_OF_CAMPUSES] - an array of votes by campus
member functions:
+ CandidateType() : CandidateType - default constructor, initialize all votest to 0

+ updateVotesByCampus(campusNumber : int, numVotes : int) : void - updates the number of votes to the selected campus to the number provided

+ getTotalVotes() const : int - returns total number of votes

+ getVotesByCampus(campusNumber : int) const : int - returns the number of votes for a particular campus

+ printCandidateInfo() : void - prints the candidate info
(formated as "###-##-####-lName, fName")

+ printCandidateTotalVotes() : void - prints the total number of votes
(formatted as "Lastname, Firstname \n\t" "Total Votes (all campuses): ##")
(in other words, it has a linebreak followed by a tab)

+ printCandidateCampusVotes(campusNumber : int) : void - prints the candidates votes for the specified campus
(formated as "Lastname, Firstname \n\t" "Campus # total vote: ##)

+ ~CandidateType() - default destructor

CandidateList:
Inherits Node.h and CandidateType.h
Member Variables:

- Node* First - pointer of type node that points to first node in the list
- Node* Last - pointer of type Node that points to last node in the list
- Int count - integer of total number of Node objects in the list

Member Functions:

+ CandidateList() - default constructor
+ Void addCandidate(CandidateType newCandidate) - adds a new candidate to the list
+ CandidateType getWinner() const - returns the winner of the election
+ Bool searchCandidate(int ssn) const - returns the candidate with the associated social security number
+ Void printCandidateName(int ssn) const - prints the candidate associated with the social security number's name
+ Void printAllCandidates() const - prints all info associated with all candidates
+ Void printCandidateCampusVotes(int ssn, int division) const - print the candidate with the associated social security number's votes for the specified campus
+ Void printCandidateTotalVotes(int ssn) const - prints the sum of all the votes for all campuses for the candidate associated with the social security number
+ Void destroyList() - helper function for the destructor
+ ~CandidateList() - destructor, runs through the list and destroys all node objects

Algorithm Design

- Classes:
    - PersonType
        - Base Class
        - O(1)
    - CandidateType
        - Inherits PersonType

        - O(n)
    - Node
        - Containers for data in CandidateList
        - Inherits CandidateType
        - O(1)
    - CandidateList
        - Inherits Node
        - Manages all functionality related to the linked list
        - O(n)

Inheritance Order:
PersonType -> CandidateType -> Node -> CandidateList

Testing and Results

Testing

Software was tested with a mixture of unit testing and manual trial and error. Our software uses the Doctest testing framework. As the project went on we relied more on manual testing largely due to deadlines.

Results

Our program's output is largely correct. There are two known bugs. If an incorrect social security number is entered on certain menu selections the program will erroneously print out more than one status indicator. The other error has to do with the provided specification. It is stated in the comments of the provided driver program that menu option 5 should print out all candidates and their vote totals. However, there isn't a neat interface into the classes in the provided specifications to achieve that. The provided implementation calls printAllCandidates, however that function is not intended to print vote totals. The main menu driver program at that point has access to a prebuilt candidate list object which doesn't expose a method externally to traverse the list and print the votes for all candidates. Our implementation opts to display the per campus results for the winning candidate.

Final Deliverables

Source Code

CandidateList.cpp

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

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

void CandidateList::addCandidate(CandidateType newCandidate)
{
	if (count == 0)
	{
		first = new Node(newCandidate, nullptr);
		last = first;
	}
	else
	{
		Node* newNode = new Node(newCandidate, nullptr);
		last->setLink(newNode);
		last = newNode;
	}
	count++;
}

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

	Node* checkNode = first;

	while (checkNode != nullptr && ssn != checkNode->getCandidate().getSSN())
		checkNode = checkNode->getLink();
```

```
	bool success = checkNode != nullptr;

	if(!success)
	{
		std::cout << "=> SSN is not in the list" << std::endl;
	}

	return success; //Returns whether the iterator reached the end of the list.
}

CandidateType CandidateList::getWinner() const
{
	CandidateType winner = first->getCandidate();

	for (Node* checkNode = first; checkNode != nullptr; checkNode = checkNode->getLink())
//Scan the whole linklist
	{
		if (checkNode->getCandidate().getTotalVotes() > winner.getTotalVotes())
		{
			winner = checkNode->getCandidate(); //Finds the highest number.
		}
	}

	return winner;
}

void CandidateList::printCandidateName(int ssn) const
{
	if (count == 0)
	{
		std::cout << "=> List is Empty" << std::endl;
	}
	else
	{
		Node* nodePtr = first;

		while (nodePtr != nullptr && nodePtr->getCandidate().getSSN() != ssn)
			nodePtr = nodePtr->getLink();

		if (nodePtr == nullptr)
		{
			std::cout << "=> SSN is not in the list" << std::endl;
		}
```

```
		else
		{
			nodePtr->getCandidate().printName();
		}
	}
}

void CandidateList::printAllCandidates() const
{
	if (count == 0) {
		std::cout << "=> List is empty." << std::endl;
	}
	else {
		Node* currentNode = first;
		while (currentNode != last) { //might not print last
			currentNode->getCandidate().printCandidateInfo();
			currentNode = currentNode->getLink();
		}

	}
}

void CandidateList::printCandidateCampusVotes(int ssn, int division) const
{
	if (count == 0)
	{
		std::cout << "=> List is Empty" << std::endl;
	}
	else
	{
		Node* currentNode = first;

		while (currentNode != nullptr && currentNode->getCandidate().getSSN() != ssn)
			currentNode = currentNode->getLink();

		if (currentNode == nullptr)
		{
			std::cout << "=> SSN is not in the list" << std::endl;
		}
		else
		{
			currentNode->getCandidate().printCandidateCampusVotes(division);
			std::cout << std::endl << std::endl;
		}
```

```
        }
}

void CandidateList::printCandidateTotalVotes(int ssn) const
{
        if (count == 0)
        {
                std::cout << "=> List is Empty" << std::endl;
        }
        else
        {
                Node* currentNode = first;

                while (currentNode != nullptr && currentNode->getCandidate().getSSN() != ssn)
                        currentNode = currentNode->getLink();

                if (currentNode == nullptr)
                {
                        std::cout << "=> SSN is not in the list" << std::endl;
                }
                else
                {
                        currentNode->getCandidate().printCandidateTotalVotes();
                        std::cout << std::endl << std::endl;
                }
        }
}

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

        for (int i = 1; i < count; ++i) {
                delete currentNode;
                currentNode = currentNode->getLink();
        }
        //very elegant, i like this a lot -j
}

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

CandidateList.h

```
#include"Node.h"
#include"CandidateType.h"

class CandidateList
{
private:
	Node* first;
	Node* last;
	int count;
public:
	CandidateList();
	void addCandidate(CandidateType newCandidate);
	CandidateType getWinner() const;
	bool searchCandidate(int ssn) const;
	void printCandidateName(int ssn) const;
	void printAllCandidates() const;
	void printCandidateCampusVotes(int ssn, int division) const;
	void printCandidateTotalVotes(int ssn) const;
	void destroyList();
	~CandidateList();
};
```

CandidateType.cpp

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

CandidateType::CandidateType() : PersonType()
{
	for (int i = 0; i < NUM_OF_CAMPUSES; ++i) {
		votesByCampus[i] = 0;
	}
}

CandidateType::CandidateType(std::string firstName, std::string lastName, int
socialSecurityNumber, int* voteByCampusPointer) : PersonType(firstName, lastName,
socialSecurityNumber) //ok this probably works but still check it out
{
	for (int i = 0; i < NUM_OF_CAMPUSES; ++i) {
		votesByCampus[i] = *(voteByCampusPointer + i);
	}
}

void CandidateType::updateVotesByCampus(int campusNum, int votes)
```

```
{
	votesByCampus[campusNum] = votes;
}

int CandidateType::getTotalVotes() const
{
	int total = 0;

	for (int v : votesByCampus)
		total += v;


	return total;
}

int CandidateType::getVotesByCampus(int campusNum) const
{
	return votesByCampus[campusNum];
}

std::string CandidateType::formatCandidateInfo() const
{
	std::string output;

	output += formatSSN();
	output += " - ";
	output += formatName();

	return output;
}

void CandidateType::printCandidateInfo()
{
	std::cout << formatCandidateInfo() << std::endl;
}

std::string CandidateType::formatCandidateTotalVotes() const
{
	std::string output;

	output += "Total votes ( all campuses ) : ";
	output += std::to_string(getTotalVotes());
```

```
	return output;
}

void CandidateType::printCandidateTotalVotes()
{
	std::cout << formatName() << std::endl;
	std::cout << "    " << formatCandidateTotalVotes() << std::endl;
}

std::string CandidateType::formatCandidateCampusVotes(int campusNum) const
{
	std::string output;

	output += "Campus ";
	output += std::to_string(campusNum);
	output += " total votes: ";
	output += std::to_string(*(votesByCampus + campusNum - 1));

	return output;
}

void CandidateType::printCandidateCampusVotes(int campusNum)
{
	std::cout << formatName() << std::endl;
	std::cout << "    " << formatCandidateCampusVotes(campusNum) << std::endl;
}


CandidateType::~CandidateType()
{

}
```

CandidateType.h

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

#ifndef CANDIDATE_TYPE_H
#define CANDIDATE_TYPE_H

const int NUM_OF_CAMPUSES = 4;

class CandidateType : public PersonType
{
```

```
private:
	int votesByCampus[NUM_OF_CAMPUSES];

	std::string formatCandidateInfo() const;
	std::string formatCandidateTotalVotes() const;
	std::string formatCandidateCampusVotes(int campusNum) const;

public:
	CandidateType();
	CandidateType(std::string firstName, std::string lastName, int socialSecurityNumber, int*
voteByCampusPointer);
	void updateVotesByCampus(int campusNum, int votes);
	int getTotalVotes() const;
	int getVotesByCampus(int campusNum) const;
	void printCandidateInfo();
	void printCandidateTotalVotes();
	void printCandidateCampusVotes(int campusNum);
	~CandidateType();
};

#endif
```

InputHandler.h

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


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.setPerfonInfo(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::string fileName;

  std::cout << "Please enter the name of an input file: ";

  std::getline(std::cin, fileName);

  //this is hacky and temporary please fix
        infile.open(fileName);

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

        createCandidateList(infile, candidateList);

        infile.close();
}
```

Main.cpp

```
/*
    NAME HEADER
*/

#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);
                  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);
                  candidateList.printCandidateTotalVotes(ssn);
}
                cout << endl << endl;
                break;

                // Print winner
            case 4:
                ssn = candidateList.getWinner().getSSN();
                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 totall votes and name of each candidate
                        cout << endl;
                        cout << "FINAL RESULTS" << endl;
                        cout << "-------------" << endl;
                        ssn = candidateList.getWinner().getSSN();
    if(ssn != 0)
    {
                          cout << "\nElection winner: ";
                          candidateList.printCandidateName(ssn);
                          for (int i = 1; i <= NUM_OF_CAMPUSES; ++i)
                                  candidateList.printCandidateCampusVotes(ssn, i);
                          candidateList.printCandidateTotalVotes(ssn);
    }
                        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;
}
```

Node.h

```
#include"CandidateType.h"
```

```
#ifndef NODE_H
#define NODE_H

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
};

#endif
```

PersonType.cpp

```
#include "PersonType.h"


PersonType::PersonType()
{
  SSN = 0;
}
PersonType::PersonType(std::string firstName, std::string lastName, int socialSecurityNumber)
{
  lName = firstName;
  fName = lastName;
  SSN = socialSecurityNumber;
}

void PersonType::setPerfonInfo(std::string firstName, std::string lastName, int
socialSecurityNumber)
{
  fName = firstName;
  lName = lastName;
  SSN = socialSecurityNumber;
}
```

```
std::string PersonType::getFirstName() const
{
  return fName;
}

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

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

void PersonType::printName()
{
    std::cout << formatName() << std::endl;
}

void PersonType::printPersonInfo()
{
    std::cout << formatPersonInfo() << std::endl;
}

void PersonType::printSSN()
{
    std::cout << formatSSN() << std::endl;
}

std::string PersonType::formatSSN() const
{
    std::string strSSN = std::to_string(SSN);

    std::string output;

    output += strSSN.substr(0, 3) + "-"; //3 Digits
    output += strSSN.substr(3, 2) + "-"; //2 Digits
    output += strSSN.substr(5, 4);       //4 Digits

    return output;
}
```

```
std::string PersonType::formatName() const
{
   return lName + ", " + fName;
}

std::string PersonType::formatPersonInfo() const
{
   std::string output;

   output += formatSSN();
   output += " ";
   output += lName;
   output += ", ";
   output += fName;

   return output;
}

PersonType::~PersonType()
{
}
```

PersonType.h

```
#include<string>
#include<iostream>

#ifndef PERSON_TYPE_H
#define PERSON_TYPE_H

class PersonType
{
  private:
    std::string fName;
    std::string lName;
    int SSN;
  protected:
    std::string formatName() const;
    std::string formatPersonInfo() const;
    std::string formatSSN() const;
   public:
    PersonType();
    PersonType(std::string firstName, std::string lastName, int SSN);
    void setPerfonInfo(std::string fName, std::string lName, int SSN);
    std::string getFirstName() const;
```

```
    std::string getLastName() const;
    int getSSN() const;
    void printName();
    void printPersonInfo();
    void printSSN();
    ~PersonType();
};

#endif
```

Tests.cpp

```
//tells the testing library to implement a main function to serve as a test driver
#define DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
//include the testing library header
#include"doctest.h"
#include"PersonType.h"
#include"CandidateType.h"
#include"CandidateList.h"
#include<string>

using namespace std;

//one of our tests. checks that the default constructor works. also relies on the get functions
behaving
TEST_CASE("test constructor") //the string is the name of our test
{
  PersonType Person;
  //this is supposed to catch if an exception is thrown,
  //but it just crashes anyway. oh well
  //
  //CHECK_THROWS(Person.fName); //check that this throws an exception
  CHECK(Person.getSSN() == 0); //check that our ssn is initialized to zero
  CHECK(Person.getFirstName() == "");
  CHECK(Person.getLastName() == "");
}

TEST_CASE("test name change")
{
  //every subcase runs this setup code first
  PersonType Person;
  //REQUIRE(Person.getFirstName == "");//require bails out of the test if it fails
  //REQUIRE(Person.getLastName == "");
  //REQUIRE(Person.getSSN == 0);
  SUBCASE("Change details")
```

```
{
  string firstName = "Katelyn";
  string lastName = "McQueen";
  int SSN = 123456789;
  Person.setPerfonInfo(firstName, lastName, SSN);
  CHECK(Person.getFirstName() == firstName);
  CHECK(Person.getLastName() == lastName);
  CHECK(Person.getSSN() == SSN);
}
SUBCASE("Change details")
{
  string firstName = "Jay";
  string lastName = "Reyes";
  int SSN = 192837465;
  Person.setPerfonInfo(firstName, lastName, SSN);
  CHECK(Person.getFirstName() == firstName);
  CHECK(Person.getLastName() == lastName);
  CHECK(Person.getSSN() == SSN);
}
SUBCASE("Change details")
{
  string firstName = "Dylan";
  string lastName = "Berry";
  int SSN = 987654321;
  Person.setPerfonInfo(firstName, lastName, SSN);
  CHECK(Person.getFirstName() == firstName);
  CHECK(Person.getLastName() == lastName);
  CHECK(Person.getSSN() == SSN);
}
SUBCASE("Change details twice")
{
  string firstName = "Dylan";
  string lastName = "Berry";
  int SSN = 987654321;
  Person.setPerfonInfo(firstName, lastName, SSN);
  CHECK(Person.getFirstName() == firstName);
  CHECK(Person.getLastName() == lastName);
  CHECK(Person.getSSN() == SSN);

  firstName = "Katelyn";
  lastName = "McQueen";
  SSN = 987654321;
  Person.setPerfonInfo(firstName, lastName, SSN);
  CHECK(Person.getFirstName() == firstName);
```

```
    CHECK(Person.getLastName() == lastName);
    CHECK(Person.getSSN() == SSN);
  }
}

TEST_CASE("add votes")
{
  CandidateType katelyn;
  katelyn.setPerfonInfo("Katie", "McQueen", 123456789);
  CHECK(katelyn.getTotalVotes() == 0);
  katelyn.updateVotesByCampus(0, 100);
  katelyn.updateVotesByCampus(1, 100);
  katelyn.updateVotesByCampus(2, 100);
  katelyn.updateVotesByCampus(3, 100);
  CHECK(katelyn.getTotalVotes() == 400);
}

TEST_CASE("search Candidate")
{
  CandidateType katelyn;
  katelyn.setPerfonInfo("Katie", "McQueen", 123456789);
  CandidateType dylan;
  dylan.setPerfonInfo("Dylan", "Berry", 12);
  CandidateList list;
  list.addCandidate(katelyn);
  list.addCandidate(dylan);
  CHECK(!list.searchCandidate(0));
  CHECK(list.searchCandidate(123456789));
  CHECK(list.searchCandidate(12));
  CHECK(!list.searchCandidate(13));
  CHECK(!list.searchCandidate(15));
}
```

Documentation

For Part A of this project, you will need to implement the PersonType Class. Make sure you follow the
same format discussed in class by having an .h file and .cpp file
PersonType class
Member Variables:
A person’s first name fName stored as a string
A person’s last name lName stored as a string
A person’s social security number SSN stored as an int

Default Constructors: Initializes the social security number to a default value of 0.
(why there is no need to initialize the first and last names?)

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

Function setPerfonInfo 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.
Destructor Left empty

A CandidateType defines a candidate that runs for the presidential election of the student government.
There are four community college campuses participating the election process. Each campus manages
its own voting process.
You will need to implement the CandidateType class that inherits from the PersonType class.
Since the CandiateType inherits from the PersonType class, MAKE SURE you implement your code
efficiently by -re-suing the functions that are already available in the PersonType.
CandiateType class

Global constant Declare a global constant integer, right before the class

definition, to store the number of participating campuses.
Name your constant: NUM_OF_CAMPUSES
Member variables An integer that stores the total number of votes
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
Initializes all the array elements 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.

Destructor (empty)

For this part, you will need to complete the class CandidateList that creates a singly-linked list of nodes
containing objects of the class CandidateType and a Pointer to the next node.

The CandidateList interface will have a class Node (implementation will be provided) that creates nodes
storing a CandidateType object and a pointer link that points to the net node. The CandidateList class
will contain a pointer that points to the first node in the list, and will contain a pointer that points to the
last node of the list. CandidateList class will also contain an int count to keep track of the number of
nodes in the list. Below is a visual illustration of the CandidateList class.
CandidateList Class
Member variables A pointer named first that points to the first node.
A pointer named last that points to the last node.
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 numbe of votes, and returns the social
security number assoicated 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.
Tranverses 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 printCandidteName 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 funcion 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.

References

Mr Alweiheiby for the Node implementation, input handler, and Main.cpp file for the final project.

Cppreference.com for language reference