Joint Assignement
Advanced Programming - DS&A II

Pierrette Sotty
Lecturers : OHara Noel and Byrne Aîne

IT Carlow - February 16, 2018

Contents

Chapter 1

Introduction

1.1 Context

This joint project between Advanced programming and Discrete Structure & Algorithme II courses has been ordered by OHara Noel and Byrne Aîne and due by February 16, 2018.

1.2 Description of the problem

The goal of this assignement is to implement a vote counting application based on the following system:

We consider n Candidates who will be selected (from preference 1 to n) by each voter represented by a BallotPaper.

Once all the BallotPapers have been collected, the Candidates will be ranked by preferences and one of them is eliminated. The Candidate who receives the lowest amount of first preference votes will be eliminated.
At this point, the vote will be transferred from all the BallotPapers which name the "loser" as highest preference to the following highest preferred Candidate of each BallotPaper.
This operation is repeated until only one Candidate remains : the winner.

Chapter 2

Solution

2.1 Research

I started with a simple idea using arrays. The storage of all the elements would have been managed with vectors and maps.

This solution included,

- in VoteCounter implementation:

  ballots : vector of pointer of BallotPaper

  candidates : vector of pointer of Candidate

  votes : vector of vector of integer (as a 2D array)

- in Candidate class :

  preference : a map of pointer of Candidate indexed by integers

To compute the election I would have make a function that removes the candidate with the lowest amout of highest preference from the candidates' vector. This function would have been performed until it remains only one item in candidates.
Each time a candidate would have been ejected, the program should have to read all the ballots to count the votes again, without him. This means that the ballots would have been read $n-1$ times, with n the number of candidates.

Considering that reading the ballots more than once is not efficient, I looked for another way to solve this problem.

2.2 Solution used

I finally chose to use a tree I made from pointers. This structure allows to count the votes each candidate got by each preference. It also saves the specific ranking from all the ballots (without reading them more than once).

The level of a node in the tree matches with the preference accorded to the candidate of this node.

A branche represents a possible order of the candidates that occured in the BallotPaper collection.

2.2.1 Data structure

Candidate : represents a candidate.

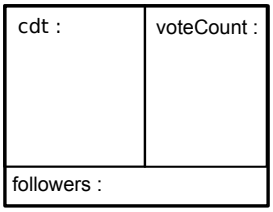
Node : contains a candidate, an amount of votes and an array of followers.

Tree : contains a dummy node, and manages the rest of the nodes from this root.

BallotPaper : contains the raking of each candidate for one ballot.

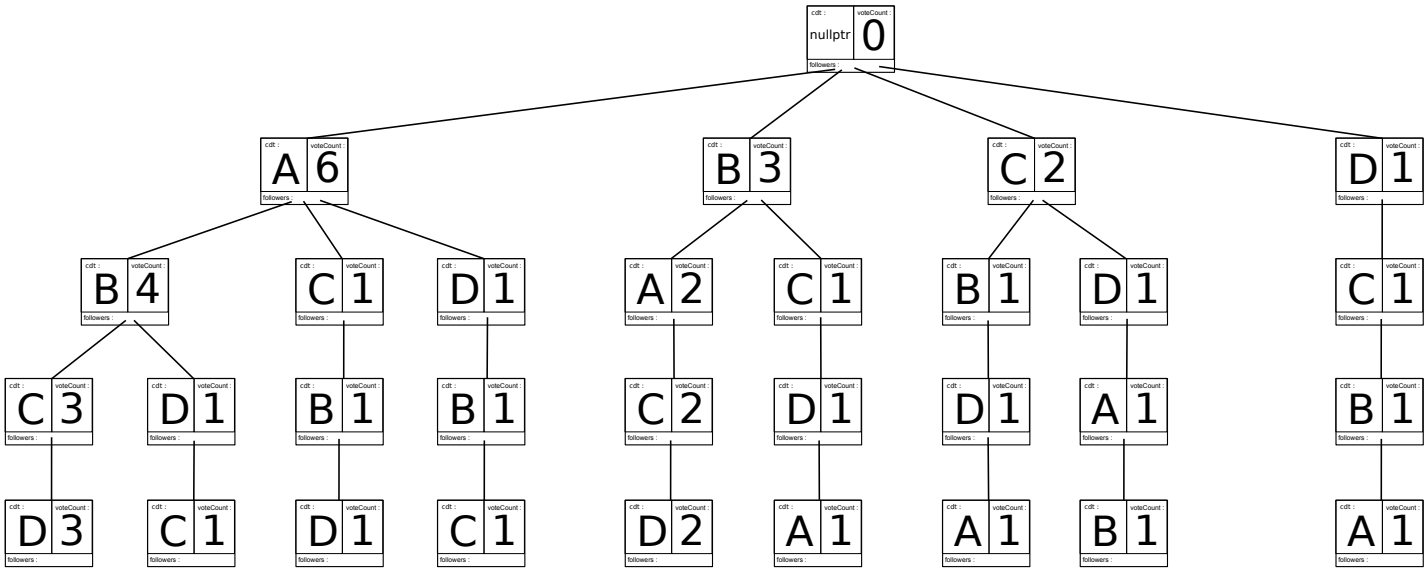
VoteCounter : contains the tree corresponding to the election and allows to use it.

2.2.2 Representation of a Node



cdt is a pointer of Candidate. *followers* is a collection of pointers of Node. *voteCount* is the number of ballots which names *cdt* (according to its preference and the previous candidates).

2.2.3 Representation of the Tree



This tree keeps a record of the following votes sample :

```
1  A;B;C;D    #List of candidates
2  1,2,3,4
3  1,2,3,4
4  1,2,3,4
5  1,2,4,3
6  1,3,2,4
7  1,3,4,2
8  2,1,3,4
9  2,1,3,4
10 4,1,2,3
11 4,2,1,3
12 3,4,1,2
13 4,3,2,1
```

With the first implementation idea, this sample would become this 2D array as votes :

Candidates	P 1	P 2	P 3	P 4
A	6	2	1	3
B	3	5	3	1
C	2	3	5	2
D	1	2	3	6

P = Preference

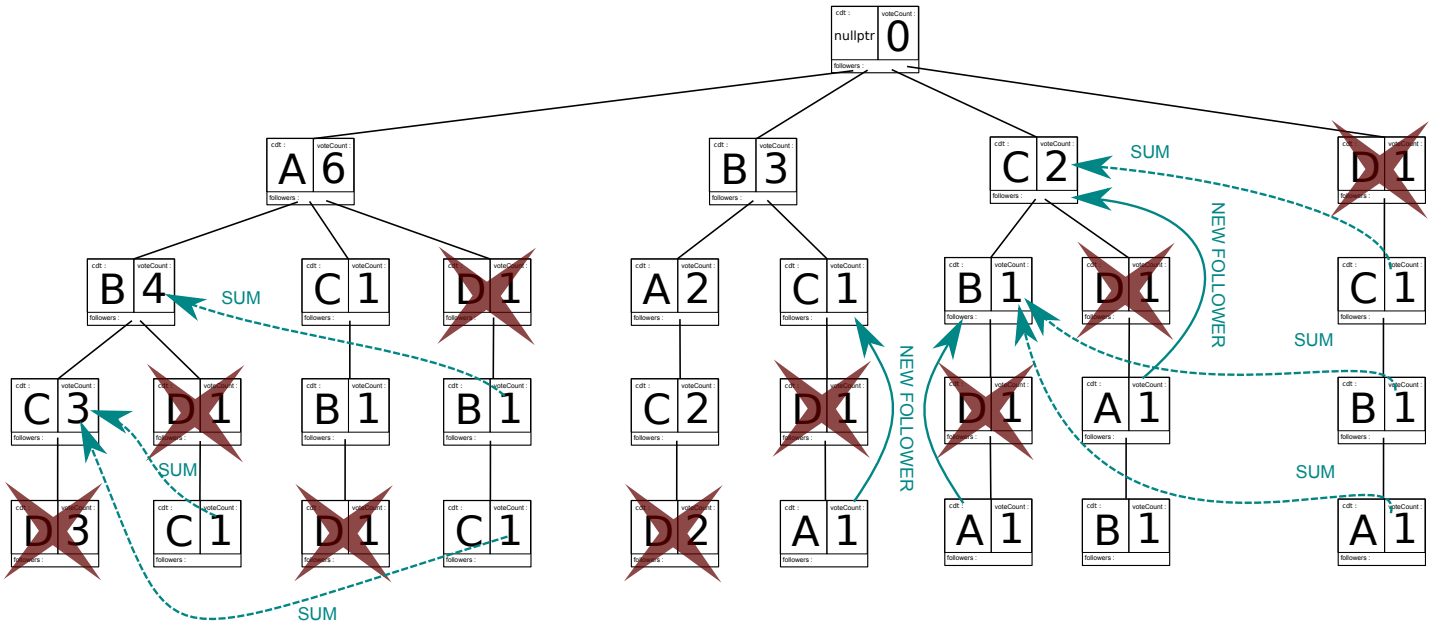
2.3 Tree fonctionning

2.3.1 Get a loser

The loser is named according to the *voteCount* of the nodes of the first level of the tree. The candidate with the lowest voteCount is return. If there equality between two candidates, the first seen is the loser.

2.3.2 Eject a candidate

The following picture shows the elimination of the candidate D from the previous example 2.2.3 on the page 5.



This operation is based on the functions *eject* and *addBranche* of the class Node called from the root of the tree. The function eject looks for the candidate through a breadth-first search. Once the loser is found, its node and its parent node merge their followers thanks to addBranche.

Each time a candidate is eliminated, the tree structure is modified to obtain the new results of the remaining candidates. The voteCount of the followers of the ejected candidate are added to the voteCount of its siblings (the node on the same level of the tree).

If the follower does not exist in the siblings list, it is added in the parent's followers collection.

To conserve the ranking from the ballot, the rest of the modified branche is merged according to the same pattern.

2.3.3 Get a winner

The most important function of the program is *getWinner* in the class Tree. This methode asks a loser to the root node, eject it and repeat while root has more than one item in its followers vector. The last candidate is designated as the winner.

2.4 Remark

I tried to develop a scalable solution that could take charge of a great quantity of data. I ran tests from little files with 4 candidates and 10 ballots to 7 candidates and over 100 ballots. The program responded well to all my tests and has always been able to display the tree in few seconds and to sort the winner quickly. I think my program is more efficient with the tree structure than the first implementation I planed.

Chapter 3

Results

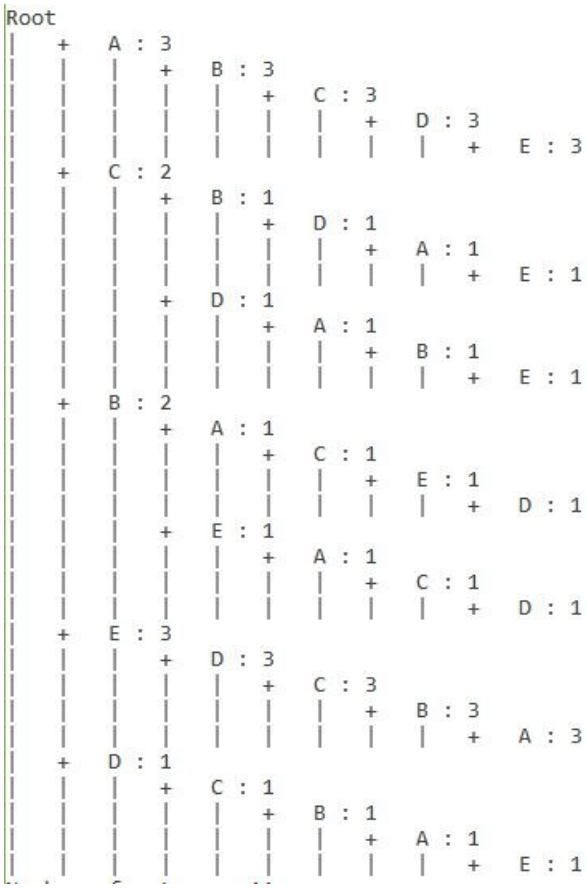
This section presents the results I obtain with my implementation of the solution and the inputs given for the demonstration. Input file, votes.txt :

```
1  A;B;C;D;E
2  1,2,3,4,5
3  4,2,1,3,5
4  2,1,3,5,4
5  5,4,3,2,1
6  1,2,3,4,5
7  3,1,4,5,2
8  5,4,3,2,1
9  5,4,3,2,1
10 4,3,2,1,5
11 3,4,1,2,5
12 1,2,3,4,5
```

3.1 Results of rounds

```
Number of voters : 11
Round #1 :
 - A : Unknown got 3 vote(s).
 - C : Unknown got 2 vote(s).
 - B : Unknown got 2 vote(s).
 - E : Unknown got 3 vote(s).
 - D : Unknown got 1 vote(s).
 -> D : Unknown is eliminated.
Round #2 :
 - A : Unknown got 3 vote(s).
 - C : Unknown got 3 vote(s).
 - B : Unknown got 2 vote(s).
 - E : Unknown got 3 vote(s).
 -> B : Unknown is eliminated.
Round #3 :
 - A : Unknown got 4 vote(s).
 - C : Unknown got 3 vote(s).
 - E : Unknown got 4 vote(s).
 -> C : Unknown is eliminated.
Round #4 :
 - A : Unknown got 7 vote(s).
 - E : Unknown got 4 vote(s).
 -> E : Unknown is eliminated.
Round #5 :
 ==> A : Unknown is the winner.
```

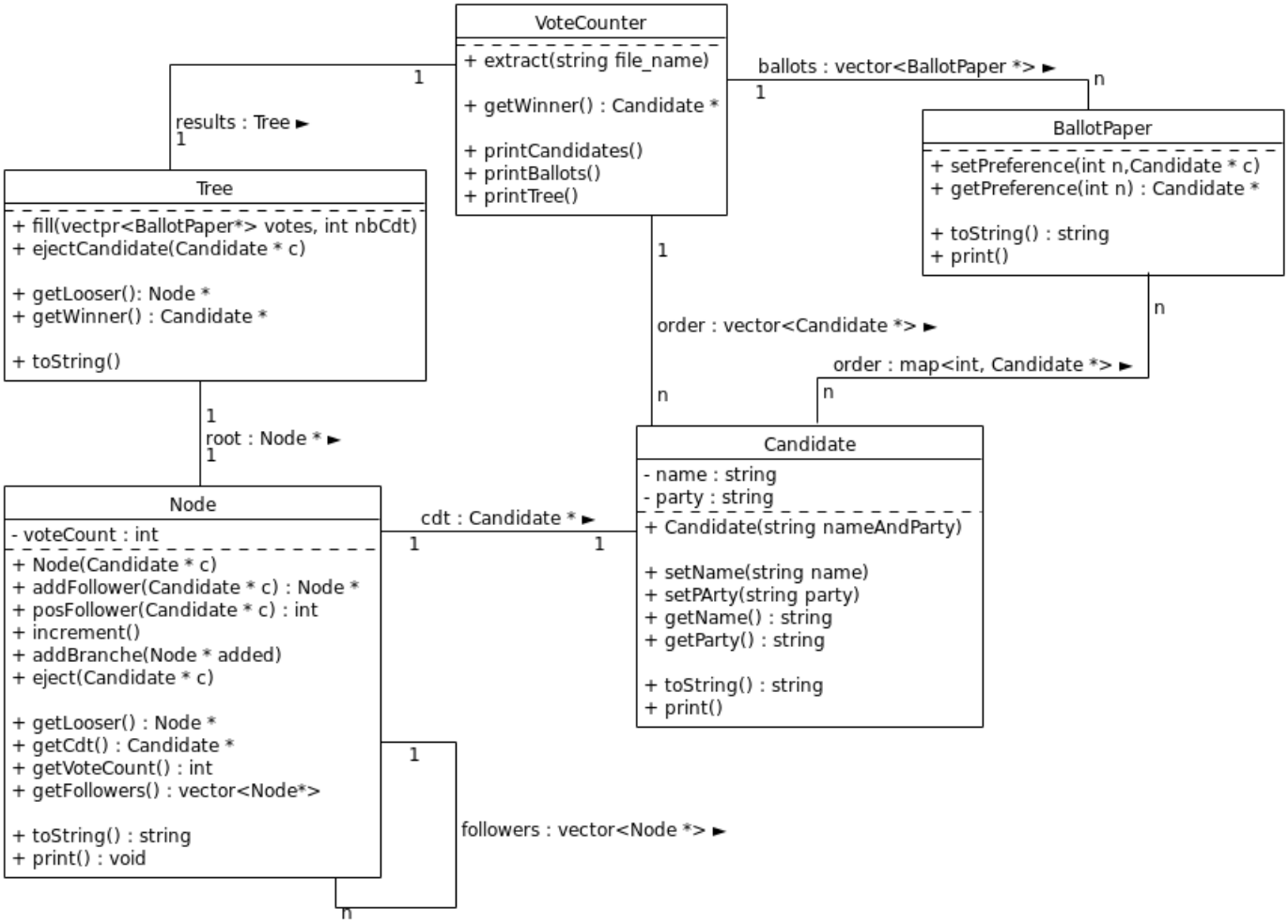
3.2 Tree displaying



Chapter 4

Code

4.1 Class diagram



4.2 Candidate

4.2.1 Header

```
/*
* Joint Assignement : C++ - DSA II
* Student : Pierrette Sotty
* Student number : C00223825
*/
#pragma once
#include <string>
#include <iostream>

using namespace std;

class Candidate
{
public:
    Candidate();
    Candidate(string nameAndParty);
    ~Candidate();

    void setName(string name);
    void setParty(string party);
    string getName();
    string getParty();
    string toString();
    void print();

private:
    string name;
    string party;
};
```

4.2.2 Implementation

```
/*
* Joint Assignement : C++ - DSA II
* Student : Pierrette Sotty
* Student number : C00223825
*/
#include "stdafx.h"
#include "Candidate.h"


Candidate::Candidate()
{
}

/**
 * initialize the candidate from nameAndParty
 *
 * @param string nameAndParty : represents the candidate
 * nameAndParty contains the name and the party of the candidate
 * they are (must be) separated by this character : ","
 */
Candidate::Candidate(string nameAndParty)
{
    int pos = nameAndParty.find(",");
    if (pos != -1)
    {
        this->setName(nameAndParty.substr(0, pos));
        this->setParty(nameAndParty.substr(pos + 1));
    }
    else {
```

```
30          this->setName(nameAndParty);
31          this->setParty("Unknown");
32      }
33  }
34
35  Candidate::~Candidate()
36  {
37  }
38
39  void Candidate::setName(string name)
40  {
41      this->name = name;
42  }
43
44  string Candidate::getName()
45  {
46      return this->name;
47  }
48
49  void Candidate::setParty(string party)
50  {
51      this->party = party;
52  }
53
54  string Candidate::getParty()
55  {
56      return this->party;
57  }
58
59
60  string Candidate::toString()
61  {
62      return this->getName() + "␣:␣" + this->getParty();
63  }
64
65  void Candidate::print()
66  {
67      cout << this->toString() << "\n";
68  }
```

4.3 BallotPaper

4.3.1 Header

```
1  /*
2  * Joint Assignement : C++ - DSA II
3  * Student : Pierrette Sotty
4  * Student number : C00223825
5  */
6  #pragma once
7  #include "Candidate.h"
8  #include <map>
9
10 using namespace std;
11
12 class BallotPaper
13 {
14 public:
15     BallotPaper();
16     ~BallotPaper();
```

```
17
18      void setPreference(int n, Candidate * c);
19      Candidate * getPreference(int n);
20      string toString();
21      void print();
22
23  private:
24      map<int, Candidate*> order; // with int preference from 1 to n
25  };
```

4.3.2 Implementation

```
1   #include "stdafx.h"
2   #include "BallotPaper.h"
3
4
5   BallotPaper::BallotPaper()
6   {
7   }
8
9
10  BallotPaper::~BallotPaper()
11  {
12  }
13
14  /**
15   * set the candidate at the n-th place of the ballot's array
16   * @param int n : preference for c
17   * @param Candidate * c : the candidate to add in the ballot
18   */
19  void BallotPaper::setPreference(int n, Candidate* c)
20  {
21      this->order[n] = c;
22  }
23
24  /**
25   * get the candidate who has the n-th preference
26   * @param int n : preference of the wanted candidate
27   */
28  Candidate* BallotPaper::getPreference(int n)
29  {
30      return this->order[n];
31  }
32
33  string BallotPaper::toString()
34  {
35      string str = "";
36      for (int i = 1; i < order.size() + 1; i++)
37      {
38          str += to_string(i) + "␣-␣" + order[i]->toString() + "\n";
39      }
40      return str;
41  }
42
43  void BallotPaper::print()
44  {
45      for (int i = 1; i < order.size() + 1; i++)
46      {
47          cout << i << "␣-␣";
48          order[i]->print();
49      }
```

```
}
```

4.4 VoteCounter

4.4.1 Header

```
/*
* Joint Assignement : C++ - DSA II
* Student : Pierrette Sotty
* Student number : C00223825
*/
#pragma once
#include "BallotPaper.h"
#include "Candidate.h"
#include "Node.h"
#include "Tree.h"
#include <vector>
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class VoteCounter
{
public:
    VoteCounter();
    ~VoteCounter();
    void extract(string file_path);
    Candidate * getWinner();

    void printCandidates();
    void printBallots();
    void printTree();

private:
    const string FILE_PATH = "../votes.txt";
    vector<BallotPaper*> ballots;
    vector<Candidate*> order;
    Tree results;
};
```

4.4.2 Implementation

```
/*
* Joint Assignement : C++ - DSA II
* Student : Pierrette Sotty
* Student number : C00223825
*/
#include "stdafx.h"
#include "VoteCounter.h"

/**
 * fill the tree with the extracted data
 */
VoteCounter::VoteCounter()
{
    this->extract(FILE_PATH);
    this->results.fill(this->ballots, this->order.size());
}
```

```
17
18
19  VoteCounter::~VoteCounter()
20  {
21  }
22
23  /**
24   * extract the candidates and the ballots from file_path
25   * add those data in ballots and order vectors
26   */
27  void VoteCounter::extract(string file_path)
28  {
29      string line;
30      ifstream myfile(file_path);
31      if (myfile.is_open())
32      {
33          // get the 1st line
34          getline(myfile, line);
35          int i;
36          int num = 0;
37          string nom;
38          // this line contains the candidates and their party
39          while (line.find(";") != -1)
40          {
41              i = line.find(";");
42              nom = line.substr(0, i);
43              line = line.substr(i + 1);
44              Candidate* c = new Candidate(nom);
45              order.push_back(c);
46              num++;
47          }
48          // last candidate
49          nom = line;
50          Candidate* c = new Candidate(nom);
51          order.push_back(c);
52
53          // get ballots
54          while (getline(myfile, line))
55          {
56              BallotPaper* ballot = new BallotPaper();
57              int cdt = 0;
58              int choice;
59              while (line.find(",") != -1)
60              {
61                  i = line.find(",");
62                  choice = stoi(line.substr(0, i));
63                  ballot->setPreference(choice, order[cdt]);
64
65                  line = line.substr(i + 1);
66                  cdt++;
67              }
68              ballot->setPreference(stoi(line), order[cdt]);
69              //ballot->toString();
70              //cout << '\n';
71              ballots.push_back(ballot);
72          }
73          myfile.close();
74      }
75
76      else cout << "Unable␣to␣open␣file";
77  }
```

```
78
79  /**
80   * get the winner of the election from the tree
81   * @return the candidate who wins
82   */
83  Candidate * VoteCounter::getWinner()
84  {
85      cout << "Number␣of␣voters␣:␣" << this->ballots.size() << "\n";
86      Candidate * winner = this->results.getWinner();
87      return winner;
88  }
89
90  void VoteCounter::printCandidates()
91  {
92      int len = order.size();
93      for (int i = 0; i < len; i++) {
94          cout << "Candidate␣#" << i << "␣:␣\n";
95          cout << order.at(i)->toString();
96          cout << "\n";
97      }
98      cout << "\n";
99  }
100
101 void VoteCounter::printBallots()
102 {
103     for (int i = 0; i < ballots.size(); i++)
104     {
105         cout << "Ballot␣#" << i << "␣:␣\n";
106         cout << ballots.at(i)->toString();
107         cout << "\n";
108     }
109 }
110
111 void VoteCounter::printTree()
112 {
113     this->results.print();
114 }
```

4.5 Tree

4.5.1 Header

```
1  /*
2  * Joint Assignement : C++ - DSA II
3  * Student : Pierrette Sotty
4  * Student number : C00223825
5  */
6  #pragma once
7  #include "Node.h"
8  #include "Candidate.h"
9  #include "BallotPaper.h"
10 #include <vector>
11
12 using namespace std;
13
14 class Tree
15 {
16 public:
17     Tree();
18     ~Tree();
```

```
19      void fill(vector<BallotPaper*> votes, int nbCdt);
20      void ejectCandidate(Candidate* c);
21      Node* getLoser();
22      Candidate * getWinner();
23      void print();
24
25  private:
26      Node * root;
27  };
```

4.5.2 Implementation

```
1   /**
2    * Joint Assignement : C++ - DSA II
3    * Student : Pierrette Sotty
4    * Student number : C00223825
5    */
6   #include "stdafx.h"
7   #include "Tree.h"
8
9   /// creates the root of the tree
10  Tree::Tree()
11  {
12      /// first node doesn't contain any Candidate *
13      root = new Node(nullptr);
14  }
15
16
17  Tree::~Tree()
18  {
19  }
20
21  /**
22   * fill(vector<BallotPaper*> votes, int nbCdt)
23   * @param vector<BallotPaper*> votes : liste of ballots
24   * @param int nbCdt : number of candidate
25   * fill the tree with the content of ballots
26   */
27  void Tree::fill(vector<BallotPaper*> votes, int nbCdt)
28  {
29      Node* current;
30      for (int i = 0; i < votes.size(); i++)
31      {
32          current = root;
33          for (int pref = 1; pref < nbCdt + 1; pref++)
34          {
35              current = current->addFollower(votes[i]->getPreference(pref));
36          }
37      }
38  }
39
40  /**
41   * eject c from the tree
42   * call the function eject(Candidate * c) of the Node* root
43   * @param Candidate * c : pointer on the rejected candidate
44   */
45  void Tree::ejectCandidate(Candidate * c)
46  {
47      this->root->eject(c);
48  }
49
```

```
50 /**
51  * Names a loser from the tree
52  * @return the loser designated by the root's function getLooser
53  */
54 Node * Tree::getLoser()
55 {
56     return root->getLoser();
57 }
58
59 /**
60  * Names a winner from the tree :
61  * While the root has more than 1 followers, one is eliminated
62  * The last candidate standing as root follower is the winner
63  * @return the winner of the election
64  */
65 Candidate * Tree::getWinner()
66 {
67     int tour = 1;
68     while (this->root->getFollowers().size() > 1)
69     {
70         cout << "Round␣#" << tour << "␣:\n";
71         for (int numCdt = 0; numCdt < this->root->getFollowers().size(); numCdt++)
72         {
73             cout << "␣-␣" << this->root->getFollowers()[numCdt]->toString() << "\n";
74         }
75         Node * looser = this->getLoser();
76         cout << "␣->␣" << looser->getCdt()->toString() << "␣is␣eliminated.\n";
77         //looser->getCdt()->toString();
78         this->ejectCandidate(looser->getCdt());
79         tour++;
80     }
81
82     Node * winner = root->getFollowers()[0];
83     cout << "Round␣#" << tour << "␣:\n";
84     cout << "␣==>␣" << winner->getCdt()->toString() << "␣is␣the␣winner.\n";
85     return winner->getCdt();
86 }
87
88 /// displays the tree in the console
89 void Tree::print()
90 {
91     cout << "Root\n";
92     this->root->print();
93 }
```

4.6 Node

4.6.1 Header

```
1  /*
2  * Joint Assignement : C++ - DSA II
3  * Student : Pierrette Sotty
4  * Student number : C00223825
5  */
6  #pragma once
7  #include "Candidate.h"
8  #include <vector>
9
10 using namespace std;
11
```

```
12 class Node
13 {
14 public:
15     Node(Candidate * c);
16     ~Node();
17     Node * addFollower(Candidate* c);
18     int posFollower(Candidate* c);
19     void increment();
20     void addBranche(Node * added);
21     void eject(Candidate * c);
22 
23     Node * getLoser();
24     Candidate * getCdt();
25     int getVoteCount();
26     vector<Node*> getFollowers();
27 
28     string toString();
29     string nodeString(int rank);
30     void print();
31 
32 private:
33     Candidate * cdt;
34     int voteCount;
35     vector<Node*> followers;
36 };
```

4.6.2 Implementation

```
1  /*
2  * Joint Assignement : C++ - DSA II
3  * Student : Pierrette Sotty
4  * Student number : C00223825
5  */
6  #include "stdafx.h"
7  #include "Node.h"
8  #include "Candidate.h"
9  #include <string>
10 
11 /**
12  * Initializes the Node containing c
13  * voteCount starts from 0
14  * @param Candidate * c : pointer on the candidate represented by this node
15  */
16 Node::Node(Candidate * c)
17 {
18     this->voteCount = 0;
19     this->cdt = c;
20 }
21 
22 
23 Node::~Node()
24 {
25 }
26 
27 /**
28  * add a follower candidate to the current node
29  * @param Candidate * c : the follower to add
30  * @return : the node containing the added follower
31  */
32 Node * Node::addFollower(Candidate * c)
33 {
```

```
34      Node* follower;
35      int pos = this->posFollower(c);
36      if (pos == -1) // if no follower recorded
37      {
38          follower = new Node(c);
39          this->followers.push_back(follower);
40      }
41      else
42      {
43          follower = this->followers[pos];
44      }
45      follower->increment();
46      return follower;
47  }
48
49  /**
50   * get the position of c in the followers vector of the current node
51   * if c is not in followers it returns -1
52   * @param Candidate * c : candidate whose the position is wanted
53   * @return : position of c
54   */
55  int Node::posFollower(Candidate * c)
56  {
57      int pos = -1;
58      for (int i = 0; i < this->followers.size(); i++)
59      {
60          if (this->followers[i]->cdt == c)
61          {
62              pos = i;
63          }
64      }
65      return pos;
66  }
67
68  /// increment voteCount
69  void Node::increment()
70  {
71      this->voteCount++;
72  }
73
74  /**
75   * sum 2 branches of a tree
76   * 1 starting from the current node
77   * 1 starting from the node called added
78   * @param Node * added : node to sum
79   * for each follower in followers of added
80   *      get the pos of the follower in this.followers
81   *      that contains the same candidate as added.followers[i]
82   *      this.followers[pos].voteCount += follower.voteCount
83   *      this.sumBanches(added.followers[i])
84   */
85  void Node::addBranche(Node * added)
86  {
87      for (int i = 0; i < added->followers.size(); i++)
88      {
89          int pos = this->posFollower(added->followers[i]->getCdt());
90          if (pos != -1)
91          {
92              this->followers[pos]->voteCount += added->followers[i]->getVoteCount();
93              this->followers[pos]->addBranche(added->followers[i]);
94          }
```

```
95          else
96          {
97              this->followers.push_back(added->followers[i]);
98          }
99      }
100 }
101
102 /**
103  * gets the followers of c in followers
104  * adds them to the current followers
105  * ejects c from the followers of the current followers
106  * delete c from the current followers
107  * @param Candidate * c : candidate that has to be ejected
108  */
109 void Node::eject(Candidate * c)
110 {
111     int pos = this->posFollower(c);
112     if (pos != -1)
113     {
114         this->addBranche(this->followers[pos]);
115         this->followers.erase(this->followers.begin()+pos);
116     }
117     for (int i = 0; i < this->followers.size(); i++)
118     {
119         followers[i]->eject(c);
120     }
121 }
122
123 /**
124  * names the loser candidate with the lowest voteCount in followers
125  * @return loser
126  */
127 Node * Node::getLoser()
128 {
129     Node * loser = followers[0];
130     for (int i = 0; i < this->followers.size(); i++)
131     {
132         if (followers[i]->getVoteCount() < loser->getVoteCount())
133             loser = followers[i];
134     }
135     return loser;
136 }
137
138
139 Candidate * Node::getCdt()
140 {
141     return this->cdt;
142 }
143
144
145 int Node::getVoteCount()
146 {
147     return this->voteCount;
148 }
149
150
151 vector<Node*> Node::getFollowers()
152 {
153     return this->followers;
154 }
155
```

```
156
157 string Node::toString()
158 {
159     string str = this->cdt->toString() + "␣got␣" + to_string(this->voteCount) + "␣vote(s).";
160     return str;
161 }
162
163 void Node::print()
164 {
165     /*
166     if (this->cdt != nullptr)
167     {
168         cout << "- " << this->voteCount << " : " ;
169         this->cdt->print();
170     }
171     for (int i = 0; i < this->followers.size(); i++)
172     {
173         followers[i]->print();
174     }
175     */
176     cout << this->nodeString(0);
177 }
178
179 /**
180  * create a string for the current node and its followers
181  * @param int rank : rank of the current node in the tree
182  * @return : the string representing the branche from this node in the tree
183  */
184 string Node::nodeString(int rank)
185 {
186     string str = "";
187     if (this->cdt != nullptr)
188     {
189         for (int i = 0; i < rank; i++)
190         {
191             str += "|␣␣␣";
192         }
193         str += "+␣␣␣" + this->cdt->getName() + "␣:␣" + to_string(this->voteCount) + "\n";
194     }
195     for (int i = 0; i < this->followers.size(); i++)
196     {
197         for (int i = 0; i < rank; i++)
198         {
199             str += "|␣␣␣";
200         }
201         str += followers[i]->nodeString(rank+1);
202     }
203     return str;
204 }
```

4.7 Main : Electoral System

```
1 /*
2 * Joint Assignement : C++ - DSA II
3 * Student : Pierrette Sotty
4 * Student number : C00223825
5 */
6 #include "stdafx.h"
7 #include "VoteCounter.h"
8 #include <iostream>
```

```
 9
10 using namespace std;
11
12
13 int main()
14 {
15     VoteCounter* vc = new VoteCounter();
16
17     vc->printTree();
18     vc->getWinner();
19
20     int num;
21     cin >> num;
22     return 0;
23 }
```