Projet d’Algorithmique et Structures de données 2

Sujet : Simuler un bureau de vote
Auteur : Hugo PIARD, Bertrand RIVARD

1. Description des différentes classes/Structures de données :

1. Classe Election :

La classe Election représente une élection et gère les opérations associées telles que l'ajout de candidats et d'électeurs, la gestion du bulletin blanc, et l'affichage des listes de candidats et d'électeurs. Trois constructeurs sont fournis : le premier initialise une élection avec un nom uniquement, le deuxième initialise une élection avec une liste de candidats et une liste d'électeurs, et le troisième initialise une élection avec uniquement une liste de candidats. La classe contient également des méthodes pour récupérer et définir le nom de l'élection, ajouter et retirer des candidats, afficher la liste des candidats, vérifier et ajouter des électeurs à la liste électorale, et afficher la liste des électeurs. Deux vecteurs sont utilisés pour stocker les candidats et les électeurs respectivement. De plus, un pointeur est utilisé pour représenter le bulletin blanc, permettant ainsi de gérer ce cas spécifique de vote.

2. Classe Electeur :

La classe Electeur représente un électeur participant à une élection. Elle hérite de la classe Personne. Les électeurs ont des caractéristiques spécifiques telles que la durée pendant laquelle ils doivent rester dans l'espace électoral et leur choix de vote. Le constructeur initialise un électeur avec un nom, un prénom, une valeur représentant sa sensibilité politique, et une durée. Les méthodes permettent d'accéder et de définir la durée, d'accéder et de définir le choix de vote de l'électeur, et d'ajouter des bulletins de vote à sa liste de bulletins. La classe contient également des méthodes pour ajouter une liste de bulletins de vote et pour afficher cette liste. Un vecteur est utilisé pour stocker les bulletins de vote de l'électeur, et un pointeur vers Personne (candidats) est utilisé pour représenter le choix de vote.

3. Classe Espace :

La classe Espace représente un espace dans le bureau de vote. C’est la classe mère des classes TableDeDecharge, Isoloir et TableDeVote. Elle est associée à une élection spécifique et à un nom, une durée dans pendant laquelle chaque électeur doit rester dans l’espace, un électeur présent dans l’espace (=NULL si il n’y a personne) et une file d’attente représenté par la classe standard queue. Les méthodes permettent d'ajouter un électeur à l'espace, de sortir un électeur de l'espace, d'afficher la file d'attente, d'afficher les informations sur l'espace, de récupérer le nom de l'espace, la durée pendant laquelle une personne reste dans l'espace, la personne actuellement dans l'espace, la file d'attente de l'espace, et l'élection associée à l'espace. Il y a aussi des méthodes pour définir la personne actuellement dans l'espace, définir l'élection associée à l'espace et ajouter une personne à la file d'attente de l'espace.

4. Classe TableDeDecharge :

La classe TableDeDecharge représente une table de décharge où les électeurs peuvent prendre des bulletin pour voter. Elle hérite de la classe Espace, ce qui signifie qu'elle partage certaines fonctionnalités avec celle-ci. Le constructeur initialise une table de décharge avec une durée, un nom, une élection associée, ainsi que des probabilités pour les bulletins blancs et nuls. Les méthodes permettent d'obtenir et de définir les probabilités de bulletins blancs et nuls, de récupérer la sensibilité politique maximale entre un électeur et un candidat, d'enregistrer le bulletin d'un électeur, de trouver le choix final (le vote) de l'électeur en fonction de sa sensibilité politique, et de rechercher le bulletin correspondant à la sensibilité politique de l'électeur. Deux variables sont utilisées pour stocker les probabilités de bulletins blancs et nuls, ainsi qu'une variable pour stocker la distance maximale entre les sensibilités politiques d'un électeur et d'un candidat.

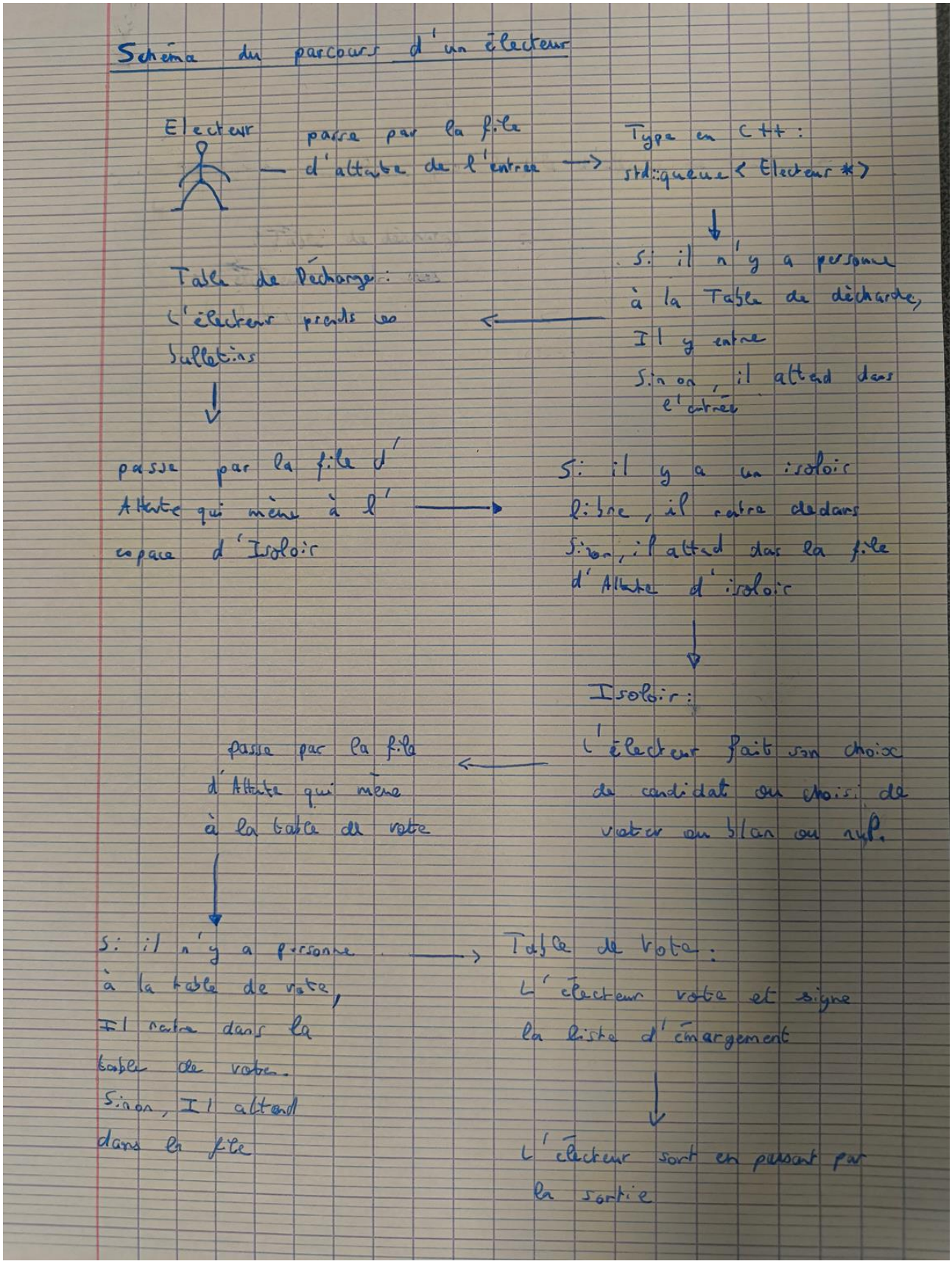
5. Classe Isoloir :

La classe Isoloir représente l’espace où il y a plusieurs isoloirs où les électeurs. Elle hérite de la classe espace. Le constructeur initialise un espace isoloir avec une durée, un nom, une élection associée, ainsi que le nombre d'isoloirs présent dans l’espace d’isoloir. Les méthodes permettent de vérifier si les différents isoloirs sont tous vides ou pleins, d'ajouter un électeur à l'espace isoloir, de sortir un électeur de l'isoloir, d'afficher les informations sur l'espace (y compris la personne à l'intérieur et la durée), et de récupérer la liste des électeurs dans les isoloirs. Un entier est utilisé pour stocker le nombre d'isoloirs et une file d'attente est utilisée pour représenter les différents isoloirs.

6. Classe TableDeVote :

La classe TableDeVote représente une table de vote où les électeurs peuvent enregistrer leur vote. Elle hérite de la classe Espace, ce qui signifie qu'elle partage certaines fonctionnalités avec celle-ci. Le constructeur initialise une table de vote avec un nombre de dés, un nom et une élection associée. La méthode vote() permet d'enregistrer le vote d'un électeur. Pour stocker les votes, une structure de données est utilisée : une table de hachage (unordered_map) où les clés sont des objets de type Personne (représentant les candidats) et les valeurs sont des entiers représentant le nombre de votes pour chaque candidat.

2. illustration du parcours d'un électeur dans un bureau de vote :



3- Complexité des opérations sur les listes :

1. La liste d’émargement :

Notre liste d’émargement est un tableau statique de booléens. Cette structure de données nous offre une complexité de O(1), donc un accès en temps constant. Chaque indice du tableau représente l'identifiant d’un électeur (chacun possédant un identifiant unique), et est associé à un booléen qui devient vrai lorsque l’électeur signe la liste d’émargement. Nous avons opté pour ce tableau car grâce aux identifiants, nous avons un accès direct aux éléments, assurant ainsi une efficacité constante, ce qui constitue un avantage significatif.
De plus, nous avons choisi un tableau statique car le nombre d’électeurs est connu à l'avance, étant donné qu'ils doivent nécessairement être présents dans le vecteur électeur pour pouvoir voter. La taille du tableau est donc logiquement égale à celle du vecteur électeur.
Cependant, un problème survient dans la fonction personneToElecteur. En effet, lors de la création d'un électeur à partir d'une personne, l'identifiant de l'électeur est incrémenté de 1 par rapport à la dernière personne créée. Ainsi, la taille réelle de notre tableau est la somme des tailles des vecteurs personnes et électeur. Cette décision favorise le coût temporel au détriment de la consommation de mémoire.
Pour résoudre cette problématique, nous pourrions envisager la création de nouveaux constructeurs dans la classe Personne.

2. La liste électorale :

La liste électorale est un vecteur de pointeurs vers des électeurs. Cette structure nous offre un accès rapide aux éléments. Nous avons donc implémenté un algorithme dichotomique qui, grâce à la récursivité, garantit une complexité en O(log(n)), avec n étant la taille du vecteur personne. Notre algorithme compare la personne recherchée avec celle située au milieu du vecteur, puis relance la recherche soit à droite, soit à gauche, en fonction du résultat de cette comparaison. Ainsi, l'algorithme parcourt au maximum la moitié du tableau, ce qui assure une efficacité constante. Ce type d'algorithme est nettement plus rapide qu'une recherche complète dans le tableau.
Nous avons opté pour l'implémentation de notre liste électorale sous forme de vecteur, car cela permet un accès plus rapide aux éléments par rapport à une liste. Une fois de plus, nous avons privilégié le coût temporel dans cette décision.

4- SDA et SDC pour un isoloir :

Notre code ne comporte pas d’isoloirs à proprement parler. Pour nous, l’isoloir est simplement une file d’attente dans laquelle chaque électeur demeure pendant un laps de temps sans forcément effectuer une action. En effet, tous les choix des bulletins sont gérés dans la table de déchargement lorsque l’électeur sélectionne ses bulletins. Il nous a semblé plus logique et plus simple de centraliser toute la gestion au même endroit. Notre espace isoloir peut être représenté comme le x-ième emplacement dans la file, mais étant donné que notre espace d’isoloir est une file, on ne peut pas vraiment y accéder directement. Cependant, cela ne pose pas de problème car chaque électeur

reste le même temps dans l'espace d'isoloir, donc il occupera le premier "isoloir" vide disponible, et le suivant occupera le suivant. Lorsque son temps sera écoulé, il sortira et le suivant prendra sa place en suivant le principe de FIFO (first in, first out).

Nous avions initialement envisagé de créer une classe Isoloir, mais nous allons plutôt définir sa SDA (Structure de Données Abstraite) et sa SDC (Structure de Données Concrète), car il est difficile de formaliser une SDC et une SDA pour notre structure d'isoloir.

1. SDA :

On va faire une SDA objet pour la classe Isoloir

-Isoloir(*in* booléen estOccupé)
.sortie : i est un isoloir vide

-estVide() → booléen
.sortie : revoie vrai si l'isoloir est vide

-ajouterElecteur(*in* pointeur_vers_électeur e)
.pré : il faut que l'isoloir soit vide
.sortie : ajoute l'électeur e dans l'isoloir

-sortirElecteur() → pointeur_vers_électeur
.pré : il faut que l'isoloir contienne une personnes
.sortie : renvoie l'électeur qui est présent dans l'isoloir

2. SDC :

Pour la classe SDC (Structure de Données Concrètes) Isoloir, nous avons opté pour un attribut pointeur_vers_électeur pour stocker l'électeur présent dans l'isoloir et un booléen pour indiquer si l'isoloir est occupé. Cette approche permet un accès direct à l'élément recherché si le pointeur est disponible.

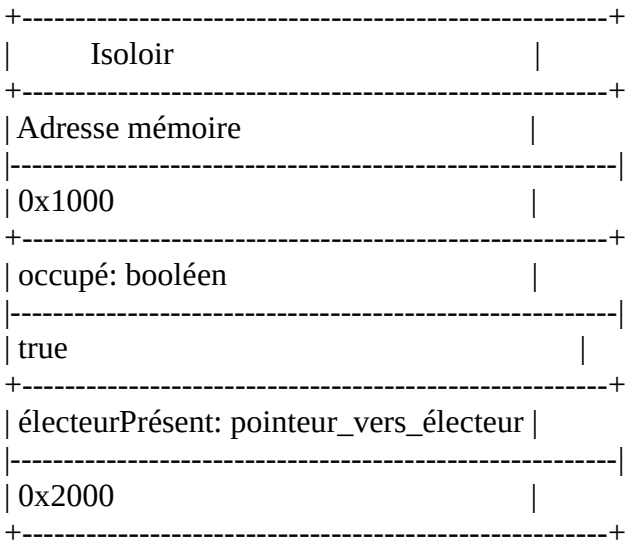
- Isoloir() : Constructeur par défaut
  occupé = faux
  électeurPrésent = NULL

- estVide() → booléen :
  Retourner (occupé == faux)

- ajouterElecteur(in pointeur_vers_électeur e) :
  Si (estVide()):
  électeurPrésent = e
  occupé = vrai
  Sinon:
  Erreur ("L'isoloir est déjà occupé")

- sortirElecteur() → pointeur_vers_électeur :
  Si (estVide()):
  Erreur ("L'isoloir est vide")
  Sinon:
  occupé = faux
  Retourner électeurPrésent

voici comment pourrait être représenté en mémoire la classe Isoloir :



5- SDA et SDC pour un espace d'isoloirs :

1. SDA :

- Isoloir(*out* isoloirs i, *in* int nbIsoloir, *in* Election elec, *in* int durée, *in* chaîne nom)
  .sortie : i est un espace d'isoloirs vide

- estVide(*in* isoloirs i) → booléen
  .sortie : retourne vrai si tous les isoloirs sont vide

-estPlein(*in* isoloirs i) → booléen
  .sortie : retourne vrai si tous les isoloirs sont plein

-aouterElecteur(*inout* isoloirs i, *in* Electeur* pers)
  .sortie : l'électeur pers est rajouté a l'espace d'isoloirs

-sortirElecteur(*inout* isoloirs i) → booléen
  .pré : il faut que l'espace d'isoloirs ne soit pas vide
  .sortie : on sort la première personne de l'espace d'isoloirs

2. SDC :

L'espace d'isoloir est implémenté par une queue de pointeurs vers électeurs issu de la bibliothèque standard c++. Pour notre espace d'isoloir nous n'avions juste besoin d'accéder qu'a la tête de la file puisque vu que chaque électeur a le même temps de passage dans l'isoloir alors le premier entrée est le premier sortie. Donc on utilise le principe FIFO, qui est donc le même principe utilisé avec la structure de donnée queue. Nous ne savons pas exactement comment sont implémenté les opérations de la queue dans notre code. Cependant vu que dans notre code nous essayons juste d'accéder a la tête de la queue et rajouter a la fin de la queue nous en déduisons que les opérations doivent être en temps constant.

Méthode Isoloir(De, nom, elec, nbIsoloir) :

Espace::Isoloir(De, nom, elec)
nbIsoloir ← nbIsoloir // Initialisation du nombre d'isoloirs
listeIsoloir ← nouvelle Queue()
Espace::setElecteurEnCours(listeIsoloir.front())

Méthode estVide() → Booléen :
Retourner listeIsoloir.empty();

Méthode estPlein() → Booléen :
Retourner taille de listeIsoloir>= nbIsoloir;

Méthode ajouterElecteur(elec)
Si l'espace d'isoloir est plein :
Ajouter l'électeur en cours à la queue d'attente.
Sinon :
Définir la durée de l'électeur.
Ajouter l'électeur à la queue d'isoloirs.

Méthode sortirElecteur() → Electeur :
Si l'espace d'isoloir est vide :
Lancer une exception indiquant que l'espace d'isoloir est vide.
Sinon :
Retirer et renvoyer le premier électeur de la queue d'isoloirs.

Adresse mémoire	Nom de la variable	Valeur
0x1000	listeIsoloir	Adresse de la file de pointeurs vers des électeurs
	pointeurs	
0x2000	Électeur* (1er)	Adresse de l'électeur 1
0x2004	Électeur* (2ème)	Adresse de l'électeur 2
0x2008	Électeur* (3ème)	Adresse de l'électeur 3
...	...	...

6- Trace d’exécution issue du logiciel :

```
ELECTION : Euro 2024
Liste des candidats :
Candidat N°0: François bul 10
Candidat N°1: Grégory yap 3
Candidat N°2: Xavier nel 5
Candidat N°3:   blanc 0
Candidat N°4: Albert bic 7


OUVERTURE DU BUREAU

T = 0
  ENTREE
Albert entre
  DECHARGE
      Albert entre
  ISOLOIR
  VOTE

T = 1
  ENTREE
  DECHARGE
       Albert prend François bul 10
       Albert prend Grégory yap 3
       Albert prend Xavier nel 5
  ISOLOIR
  VOTE

T = 2
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE

T = 3
  ENTREE
Bernard entre
  DECHARGE
Albert sort
taille file Attente isoloir :1
  ISOLOIR
      Albert entre
  VOTE

T = 4
  ENTREE
  DECHARGE
      Bernard entre
  ISOLOIR
  VOTE

T = 5
  ENTREE
Christian entre
  DECHARGE
```

```
T = 5
  ENTREE
Christian entre
  DECHARGE
       Bernard prend François bul 10
       Bernard prend Grégory yap 3
       Bernard prend Xavier nel 5
       Bernard prend Albert bic 7
  ISOLOIR
  VOTE

T = 6
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE

T = 7
  ENTREE
  DECHARGE
Bernard sort
taille file Attente isoloir :1
  ISOLOIR
      Bernard entre
  VOTE

T = 8
  ENTREE
  DECHARGE
      Christian entre
  ISOLOIR
  VOTE

T = 9
  ENTREE
Dominique entre
  DECHARGE
       Christian prend Grégory yap 3
       Christian prend François bul 1
       Christian prend Xavier nel 5
  ISOLOIR
      Albert choisi Albert bic 7
      Albert sort
  VOTE
     Albert entre

T = 10
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE
     Albert vote
```

```
T = 15
  ENTREE
Evelyne entre
  DECHARGE
Dominique sort
taille file Attente isoloir :1
  ISOLOIR
      Dominique entre
  VOTE
     Bernard vote

T = 16
  ENTREE
  DECHARGE
      Evelyne entre
  ISOLOIR
  VOTE

T = 17
  ENTREE
François entre
  DECHARGE
       Evelyne prend Grégory yap 3
       Evelyne prend François bul 10
       Evelyne prend Xavier nel 5
       Evelyne prend Albert bic 7
  ISOLOIR
      Christian choisi Grégory yap 3
      Christian sort
  VOTE

T = 18
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE
      Bernard sort
  SORTIE
      Bernard sort

T = 19
  ENTREE
Grégory entre
  DECHARGE
Evelyne sort
taille file Attente isoloir :1
  ISOLOIR
      Evelyne entre
  VOTE
     Christian entre

T = 39
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE
     Grégory entre

T = 40
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE
     Grégory vote

T = 41
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE

T = 42
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE

T = 43
  ENTREE
  DECHARGE
  ISOLOIR
  VOTE
      Grégory sort
  SORTIE
      Grégory sort

RESULTAT DU BUREAU :

nb electeurs : 7
nb vote : 7
Participation : 100.00%
Abstention : 0.00%

Dépouillement :

Albert bic 7 : 1(20.00% )
vote blanc : 1(20.00% )
Xavier nel 5 : 0(0.00% )
Grégory yap 3 : 3(60.00% )
François bul 10 : 1(20.00% )
vote nul : 1 (20.00% )
```

7- Regard critique sur la complétude et sa correction :

1. Complétude :

Nous avons essayé d'implémenter au maximum toutes les fonctionnalité de l'application. Mais malgré tout nous n'avons pas pu tout faire au maximum car cela demandait trop de temps. Par exemple chaque bulletin peut être pris infiniment par les électeurs dans la limite de 1 par électeur. Cela nous a sembler plus réaliste de laisser un nombre de bulletin infinie car lors d'une véritable élection c'est comme cela que tout est gérer. Pour résumer nous avons supposé qu'il y aurait suffisamment de bulletins pour tous les électeurs.
Une autre limite de notre code concerne la gestion des votes blancs. Concrètement, nos bulletins sont représentés par des pointeurs vers des personnes (les candidats), et nous avons créé une "personne" fictive pour représenter le vote blanc. Cependant, si nous retirons cette option de la liste des candidats, les électeurs ne pourront plus voter blanc. Nous aurions pu opter pour une approche plus flexible dans la gestion du vote blanc, peut-être en l'implémentant comme une variable globale.

Nous avons également envisagé d'ajouter une touche de réalisme à notre programme en introduisant des éléments aléatoires. Cependant, nous avons seulement réussi à rendre aléatoire l'entrée des électeurs. Pour ce qui est des candidats, ils sont saisis manuellement, ce qui, bien que non aléatoire, demeure assez réaliste. L'entrée des électeurs est certes réaliste mais il a fallut restreindre cette aléatoire pour évité de se retrouver une fois sur deux avec deux électeurs qui votent en tout. Pour cela nous avons définie des coefficient visant a réduire cette chance.

2. Bugs :

Nous avons effectué des tests approfondis sur notre logiciel et sommes convaincus qu'il ne présente aucun bug majeur, même dans des situations variées telles que des vecteurs de personnes ou d'électeurs vides. Cependant, malgré nos efforts, il est possible que quelques bugs nous aient échappé, et l'aléatoire peut parfois entraîner des problèmes imprévus. Cependant, nous considérons que ces incidents restent minimes.

Il est important de reconnaître qu'aucun logiciel informatique n'est jamais parfait, et qu'il existe toujours des améliorations potentielles à apporter, même mineures. Par exemple, en ce qui concerne la réutilisation de notre logiciel, nous pourrions envisager des ajustements, tels que la possibilité d'alimenter le vecteur Personne de manière différente, comme en demandant à l'utilisateur d'entrer les données manuellement ou en développant une fonction capable de remplir le vecteur à partir d'une certaine structure de données.

Bien que nous ayons consacré beaucoup d'efforts au développement de notre application, nous sommes conscients qu'il reste encore du travail à faire pour l'améliorer davantage.