Sorbonne Université **M1 Mathématiques**
2023–2024 **MU4M056 : Programmation en C++**

TP noté 1 : Jeu Mastermind

L'objectif de ce TP est d'implémenter en C++ le jeu du mastermind avec différentes classes. On veillera à inclure dans chaque fichier toutes les bibliothèques nécessaires et les options de compilation nécessaires. **Il est également impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant.**

1 Détails du jeu

1.1 Explications

Le Mastermind est un jeu de société pour deux joueurs, un Codificateur et un Décodeur. Le but est de deviner, par déductions successives, une combinaison ordonnée de 4 chiffres parmi 6. Il se présente généralement sous la forme d'un plateau perforé de 10 rangées de 4 trous pouvant accueillir des pions de 6 valeurs possibles. Dans notre cas, nous supposons ques les 4 chiffres à deviner sont tous différents.

Le jeu se déroule de la façon suivante : le Codificateur cache derrière un écran la combinaison de son choix. Son adversaire, le Décodeur, est chargé de déchiffrer ce code secret. Il doit le faire en 10 coups au plus. À chaque tour, le Décodeur doit remplir une rangée selon l'idée qu'il se fait de la combinaison dissimulée. Une fois la combinaison proposée par le Décodeur, le Codificateur indique au Décodeur :
— le nombre de bonnes valeurs bien placées (BVBP),
— le nombre de bonnes valeurs mal placées (BVMP).

1.2 Exemple

Les 6 valeurs possibles de notre jeu sont $\{0, 1, \ldots, 5\}$. Faisons un exemple du déroulement d'une partie. Pour commencer, le Codificateur choisi la combinaison $(4, 5, 0, 1)$ et la cache derrière un écran. Le Décodeur doit maintenant arriver à deviner cette combinaison et il procède donc par tentatives successives, de la façon illustrée en Figure 1.

Combinaison à deviner	4	5	0	1	
Premier essai	0	1	2	3	BVBP : 0, BVMP : 2
Deuxième essai	2	4	5	1	BVBP : 1, BVMP : 2
Troisième essai	1	0	5	4	BVBP : 0, BVMP : 4
Quatrième essai	5	4	1	0	BVBP : 0, BVMP : 4
Cinquième essai	4	5	0	1	BVBP : 4, BVMP : 0

FIGURE 1 – Partie de Mastermind

Premier essai : $(0, 1, 2, 3)$
Il y a deux valeurs bonnes. Le Décodeur en déduit que 4 et 5 sont dans le code secret.

Deuxième essai : $(2, 4, 5, 1)$
Il y a trois valeurs bonnes. Le Décodeur en déduit que 0 est dans le code.

Troisième essai : $(1, 0, 5, 4)$
Le code est composé des chiffres 0, 1, 4 et 5. Mais ils sont tous mal placés. On sait alors que le 5 n'est pas en troisième position et que le zéro n'est ni en première ni en deuxième position.

Quatrième essai : $(5, 4, 1, 0)$
Tous les chiffres sont encores mal placés. Cela indique que le 0 est forcément en troisième position. Comme à l'essai 2, on avait une valeur bien placée, on sait maintenant que c'est le 1 et qu'il est en dernière position. Il ne reste plus qu'à échanger le 4 et le 5 pour trouver la solution.

Cinquième essai : $(4, 5, 0, 1)$
Le Décodeur a deviné la solution en 5 coups !

2 Questions

2.1 Premières méthodes et premiers tests

Dans notre cas, le code secret à deviner est un vecteur d'entiers de taille 4, rempli avec des entiers pris dans l'ensemble $\{0, 1, \ldots, 5\}$. Cependant, nous souhaitons généraliser le jeu à une taille quelconque n et à un ensemble de valeurs $\{0, 1, \ldots, v_{max}\}$ où v_{max} est un entier quelconque supérieur ou égal à n.

Nous imposons la classe suivante dans un fichier `mastermind.hpp` :

```
class CodeSecret{
    private:
        int n;  //taille du code
        int v_max; // valeur maximale de l'ensemble des valeurs possibles
        std::vector<int> code; // code secret
    public:
       // A COMPLETER
};
```

1. Compléter tous les fichiers avec les inclusions de fichiers et de bibliothèques nécessaires.

2. Écrire un constructeur par défaut `CodeSecret()` qui définit un code secret vide avec une seule valeur possible.

3. Écrire un constructeur qui prend en argument un vecteur d'entiers et un entier v_{max} et crée un code secret avec une taille égale à la taille du vecteur donné, une valeur maximale de l'ensemble qui est égale à v_{max} et un vecteur de valeurs égal au vecteur donné.

4. Écrire un constructeur qui prend en argument un vecteur d'entiers et crée un code secret avec une taille égale à la taille du vecteur donné, une valeur maximale de l'ensemble qui est égale à la valeur maximale du vecteur et un vecteur de valeurs égal au vecteur donné.
Pour cela, on pourra utiliser `std::max_element` (qui prend en argument deux itérateurs) de `<algorithm>` qui renvoie un itérateur sur la case qui correspond au plus grand élément.

5. Ajouter un accesseur à la taille du code secret `size()` , un accesseur à la valeur maximale v_{max} de l'ensemble `value_max()` et un accesseur aux éléments de `code` à l'aide des crocherts `[]` . Il s'agit ici de surcharger `int operator[] (int i) const` .

6. Surcharger l'opérateur `<<` de telle sorte qu'un code soit écrit avec le format suivant :

```
c_1 c_2 ... c_n
```

(une ligne d'entiers représentants les valeurs du vecteur séparées par des espaces).

7. Tester la première partie du code présent dans le fichier `test_jeu.cpp` qui doit fonctionner et comparer au résultat attendu.

Attention : Tant que vous n'obtenez pas les bons résultats à la question précédente, il est inutile de continuer.

2.2 Comparaison des codes

À présent, nous souhaitons comparer le code secret donné par le Codificateur et le code proposé par le Décodeur. Pour cela, nous allons ajouter de nouvelles méthodes à notre classe `CodeSecret` .

8. Écrire le code de la méthode

```
int BonneValeurBonEndroit(CodeSecret &prop) const;
```

qui prend en argument le code proposé par le Décodeur et qui renvoie un entier qui correspond au nombre de bonnes valeurs au bon endroit.

9. Écrire le code de la méthode

```
int BonneValeurMauvaisEndroit(CodeSecret &prop) const;
```

qui prend en argument le code proposé par le Décodeur et qui renvoie un entier qui correspond au nombre de bonnes valeurs au mauvais endroit. On rappelle ici que l'on suppose que l'on n'a pas de redondance dans les chiffres du code secret. On pourra utiliser `std::find` (qui prend en argument deux itérateurs et une valeur à trouver) de `<algorithm>` qui renvoie un itérateur sur le premier élément égal à la valeur recherchée dans l'intervalle. Si aucun élément n'est trouvé, il retourne un itérateur sur la fin de notre intervalle.

10. Tester la seconde partie du code présent dans le fichier `test_jeu.cpp` qui doit fonctionner et comparer au résultat attendu.

Attention : Tant que vous n'obtenez pas les bons résultats à la question précédente, il est inutile de continuer.

2.3 Décodeur

On veut à présent représenter le joueur que l'on appelle Décodeur. Pour cela, on crée la classe `Decodeur` donnée par :

```
class Decodeur{
    private:
        std::string nom; // nom du joueur
        int etape; // numéro d'étape à laquelle se situe le joueur
public:
        Decodeur(const std::string &nom);
```

```
        CodeSecret ProposerCode(int n, int v_max);
8   };
```

11. Écrire le code du constructeur qui initialise le nom du joueur et le numéro de l'étape à 0.

12. Ajouter deux accesseurs, un pour le nom du joueur `name()` et un pour le numéro de l'étape `step()` .

13. Écrire le code de la méthode

```
    CodeSecret ProposerCode(int n, int v_max);
```

qui simule une étape dans la partie du joueur. Ainsi cette méthode demande au joueur d'entrer un code et le retourne.
Remarque : Il faut faire attention au nombre de valeurs données, vérifier que les valeurs sont bien dans le bon intervalle et que l'on n'a pas de redondance. On pensera aussi à augmenter le numéro de l'étape.

14. Tester la troisième partie du code présent dans le fichier `test_jeu.cpp` qui doit fonctionner.

Attention : Tant que vous n'obtenez pas les bons résultats à la question précédente, il est inutile de continuer.

2.4 Mastermind

On souhaite à présent pouvoir jouer au jeu Mastermind. Pour cela, on crée une classe `Mastermind` :

```
    class Mastermind{
2       private:
            int nb_essai; // nombre d'essai total dans la partie
4           CodeSecret code; // code secret à deviner
        public:
6           Mastermind(int nb, CodeSecret C);
    };
```

15. Écrire le constructeur qui initialise le nombre d'essai et le code secret.

16. Écrire le code de la méthode,

```
    bool jouer(Decodeur& D) const;
```

qui prend en argument un joueur D, le fait jouer une partie de Mastermind et renvoie vrai si le joueur a gagné et faux sinon.

17. Compléter le fichier `test_jeu.cpp` pour tester la fonction précédente avec 10 essais maximum et le code secret c2 défini dans la première partie du fichier `test_jeu.cpp` .

18. On souhaiterait maintenant pouvoir jouer en solitaire et donc ne pas connaître le code secret. Pour cela, ajouter un constructeur à la classe `CodeSecret` ,

```
CodeSecret(int n, int v_max);
```

qui génère un code aléatoire de taille n avec des valeurs comprises dans l'ensemble $\{0, 1, \dots v_{max}\}$. On fera attention à ce qu'il n'y ait pas de redondance dans le code secret. On pourra utiliser `uniform_int_distribution<int> Loi(a,b)` de `<random>` pour générer des entiers aléatoires compris entre a et b.

19. Lancer alors un nouveau jeu de mastermind avec un code secret généré aléatoirement.

Fin du sujet