Sorbonne Université **M1 Mathématiques**
2019–2020 **MU4MA056 : Programmation en C++**

TP noté 1 : Mastermind

IMPORTANT : à l'issue du TP, chaque groupe envoie par email trois fichiers `mastermind.hpp` , `mastermind.cpp` et `test_mastermind.cpp` . **Il est impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant de chacun des membres du groupe.**

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, avec répétition possible, 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.

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. A 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 placés, que nous allons indiquer rr (*right-right*) ;
— le nombre de bonnes valeurs mal placés, que nous allons indiquer rw (*right-wrong*).

Un petit exemple illustratif. Les 6 valeurs possibles sont $\{0, 1, \ldots, 5\}$. Voyons un exemple de déroulement de partie. Pour commencer, le Codificateur choisi la combinaison $(1, 5, 0, 0)$ 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	1	5	0	0	
Premier essai	3	3	4	4	$rr : 0, rw : 0$
Deuxième essai	0	2	1	5	$rr : 0, rw : 3$
Troisième essai	2	2	5	0	$rr : 1, rw : 1$
Quatrième essai	0	0	5	1	$rr : 0, rw : 4$
Cinquième essai	1	5	0	0	$rr : 4, rw : 0$ *Gagné !*

FIGURE 1 – Partie

Premier essai : $(3, 3, 4, 4)$
Aucune valeur est bonne, le Codificateur indique $rr : 0$ et $rw : 0$. Le Décodeur en déduit que 3 et 4 ne sont pas dans la combinaison à deviner.
Deuxième essai : $(0, 2, 1, 5)$
0, 1 et 5 sont trois bonnes valeurs, mais au mauvais endroit, le Codificateur indique $rr : 0$ et $rw : 3$. Le Décodeur en déduit qu'il y a une répétition dans la combinaison cachée.
Troisième essai : $(2, 2, 0, 5)$
Le 5 est une bonne valeur au mauvais endroit et le 0 est une bonne valeur au bon endroit, le Codificateur indique $rr : 1$ et $rw : 1$. Le Décodeur en déduit que le 2 n'est pas la valeur répétée.

Quatrième essai : $(0, 0, 5, 1)$
Les valeurs sont toutes bonnes mais mal placées, le Codificateur indique donc $rr : 0$ et $rw : 4$. Le Décodeur en déduit que les 0 occupent la 3ème et la 4ème position, il aura gagné en maximum 2 coups !

Cinquième essai : $(1, 5, 0, 0)$
Les pions sont tous de la bonne valeur et au bon endroit, le Codificateur indique donc $rr : 4$ et $rw : 0$. Le Décodeur a deviné la combinaison en 5 coups.

Implémentation en C++. Une combinaison de 4 valeurs parmi 6 valeurs possibles sera un vecteur d'entiers de taille 4, rempli avec des entiers pris dans l'ensemble $\{0, 1, \ldots, 5\}$. Aussi, nous allons généraliser le jeu à une taille quelconque `n` et nous n'imposons pas de contraintes sur l'ensemble des valeurs possibles.

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

```
class Combination{
private:
    int n; // taille de la combinaison
    std::vector<int> values; // combinaison (de taille n)
public:
    ...
    std::set<int> unique_values() const;
    int count_value(int) const;
    int count_right_place(int, const Combination&) const;
    int count_rr(const Combination&)  const;
    int count_rw(const Combination&)  const;

    Response one_try(const Combination&) const;
    bool won(const Combination&) const;
};
```

Attention : **On veillera à étiqueter `const` toutes les méthodes et les arguments nécessaires.**

1. Recopier ce code dans un fichier `mastermind.hpp` et ajouter les `include` nécessaires. Préparer également un fichier `mastermind.cpp` avec les `include` nécessaires.

2. Écrire un constructeur par défaut qui définit une combinaison vide.

3. Écrire un constructeur qui prend en argument un vecteur d'entiers et crée une combinaison avec une taille égale à la taille du vecteur donné et un vecteur de valeurs égal au vecteur donné.

4. Ajouter un accesseur à la taille de la combinaison `size()` et un accesseur aux éléments de `values` à l'aide des crochets `[]` (il s'agit de surcharger `int operator[](int i) const`).

5. Surcharger les opérateurs `<<` et `>>` de telles sortes qu'une combinaison soit écrite ou lue avec le format suivant :

```
V_1 V_2  ... V_n
```

(une ligne d'entiers représentants les valeurs, séparés par des espaces).

6. Écrire un programme complet `test_mastermind.cpp` qui fait les choses suivantes :

— on teste le constructeur par défaut, en affichant une combinaison `C1` vide ;
— on teste le constructeur avec un vecteur $v = (2, 0, 1, 2, 4)$, en affichant la combinaison `C2` ;
— on lit le fichier `combinaison_4.dat` dans une combinaison `a_deviner` et on affiche la combinaison `a_deviner` .

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

Il est question maintenant de nous donner les moyens de comparer la combinaison donnée par le Codificateur avec les combinaisons qui seront proposées par le Décodeur.

Nous ajoutons pour cela au fichier `mastermind.hpp` la structure `Response` qui encode une réponse du Codificateur à une proposition du Décodeur.

```
struct Response{
        int rr;
        int rw;
};
```

7. Écrire le code de la méthode `count_rr` , qui prend en argument une combinaison et renvoie un entier correspondant au nombre de bonnes valeurs au bon endroit.

Compter le nombre de bonnes valeurs au mauvais endroit se révèle plus difficile, puisqu'il faudra faire attention à ne pas compter les bonnes valeurs au bon endroit et que la répétition de valeur est admise dans le jeu. Cette question sera donc découpée en plusieurs points, suivant les formules que nous donnons ici-bas.

Étant donnée une combinaison C, on indique avec $\mathcal{V}(C)$ l'ensemble des valeurs de la combinaison C, c'est-à-dire l'ensemble des valeurs qui apparaissent dans C, comptées une seule fois [1]. Soit C la combinaison à deviner choisie par la Codificateur et D une combinaison proposée par le Décodeur. Nous voulons calculer $rw_C(D)$, le nombre de pions de la combinaison D qui sont de la bonne valeur mais au mauvais endroit, lorsqu'on compare la combinaison D avec la combinaison C.

Pour chaque $v \in \mathcal{V}(D)$, on définit

— $n_D(v)$: le nombre d'apparitions de la valeur v dans la combinaison D ;
— $rr_C(v, D)$: le nombre de fois où v apparaît au même endroit dans les deux combinaisons C et D (bonne valeur au bon endroit pour la valeur v).

Alors la formule que nous cherchons s'écrit :

$$rw_C(D) = \sum_{v \in \mathcal{V}(D)} \min(n_C(v), n_D(v)) - rr_C(v, D).$$

8. Écrire le code de la méthode `unique_values()` [2] qui renvoie l'ensemble des valeurs présentes dans une combinaison, sans répétition ($\mathcal{V}(C)$ dans les formules). On pourra utiliser la fonction `std::count` [3] de la bibliothèque `<algorithm>` .

9. Écrire le code de la méthode `count_value` qui prend en argument un entier `v` et qui renvoie un entier correspondant au nombre de fois que la valeur v apparaît dans une combinaison ($n_C(v)$ dans les formules).

1. Pour $v = (2, 0, 1, 2, 4)$, cet ensemble est $\{0, 1, 2, 4\}$.
2. Nous renvoyons vers la Section 4.5 du polycopié de cours pour la documentation de l'objet `std::set` et nous rappelons que l'ajout d'un élément se fait via la méthode `std::insert` .
3. Cette fonction prend en arguments un itérateur vers le début et un itérateur vers la fin de l'objet à parcourir, et la valeur à chercher pour le comptage. Elle renvoie le nombre de fois que la valeur apparaît dans l'objet.

10. Écrire le code de la méthode `count_right_place` qui prend en arguments un entier `v` et une combinaison `D` et renvoie un entier correspondant au nombre de fois que la valeur v apparaît au bon endroit dans la combinaison D ($rr_C(v, D)$ dans les formules).

11. Écrire le code de la méthode `count_rw` qui prend en argument une combinaison et renvoie un entier correspondant au nombre de bonnes valeurs au mauvais endroit ($rw_C(D)$ dans les formules).

12. Écrire le code de la méthode `one_try` qui prend en argument une combinaison et renvoie une réponse sous forme de `Response` .

13. Écrire le code de la méthode `won` qui prend en argument une combinaison et qui renvoie vrai ou faux, selon que le combinaison passée en argument soit gagnante ou pas.

14. Compléter le fichier `test_mastermind.cpp` en testant ces méthodes :

- créer une combinaison `C3` à partir du vecteur $(2, 2, 4, 3, 1)$;
- afficher le nombre de bonnes valeurs à la bonne place de `C3` par rapport à `C2` (1) ;
- afficher le nombre de fois que 2 apparaît dans `C3` (2) ;
- afficher le nombre de fois que 2 apparaît au bon endroit dans `C3` par rapport à `C2` (1) ;
- afficher le nombre de bonnes valeurs au mauvais endroit de `C3` par rapport à `C2` (3) ;

Attention : tant que vous n'obtenez pas les résultats attendus aux questions précédentes, il est inutile de continuer.

Il s'agit maintenant de jouer au Mastermind. Nous allons pour cela créer une classe Mastermind.

```
class Mastermind{
private:
    int n_guess_max; // nombre maximal d'essais
    int n_guess_count; // nombre d'essais effectués par le Décodeur
    Combination hidden; // combinaison à deviner

public:
    Mastermind(int M, Combination C): n_guess_max(M), n_guess_count(0), hidden(C){};
};
```

15. Écrire le code de la méthode `Combination user_guess() const` qui affiche un message avec la valeur du compteur du Décodeur, lui demande de rentrer une combinaison en ligne de commande, de la même taille que la combinaison à deviner (`hidden`), et renvoie la combinaison proposée par le Décodeur.

16. Écrire le code de la méthode `bool play()` qui demande au Décodeur de proposer une combinaison, tant qu'il n'a pas deviné et tant que le compteur n'a pas dépassé le nombre maximal d'essais possibles. Pour chaque proposition, on affiche une réponse. Cette méthode renvoie `true` si le Décodeur a gagné (deviné la combinaison cachée en moins de `n_guess_max` essais) et `false` sinon.

17. Dans le fichier `test_mastermind.cpp` créer un jeu de Mastermind avec 10 essais maximaux et la combinaison `a_deviner` , et vérifier que le jeu marche.

Culture générale : En 1977 Donald Knuth, auteur du livre de référence *The Art of Computer Programming* et créateur du langage TEX, a démontré qu'on peut s'assurer de gagner en moins de 5 essais, voir ici le détail de l'algorithme : https ://github.com/nattydredd/Mastermind-Five-Guess-Algorithm.