TP C++ 4 : Classes

M1 Mathématiques Appliquées

2019-2020

Remarque préliminaire : Il est fortement recommandé de valider les questions de manière incrémentale à l'aide d'un programme principal.

Exercice 1 : Jeu du morpion

Le but de cet exercice est créer une classe qui simule une partie de morpion. Cette classe aura pour membres données une tableau bi-dimensionnel de taille 3x3 représentant la grille de jeu. Chaque case de la grille pourra prendre les valeurs `vide,joueur1,joueur2` selon qu'elle sera non occupée ou prise par l'un des joueurs. Une instance de la classe sera capable de simuler une partie en changeant l'état de la grille. Elle devra donc savoir à tout moment quel joueur sera le prochain à jouer, et elle devra aussi être capable de déterminer un éventuel vainqueur.

a) Définir un type énumération `typecase` contenant les valeurs `vide,joueur1,joueur2`.

b) Définir la classe `Morpion` et son constructeur. Par défaut, les valeurs de la grille de morpion seront initialisées à `vide`, et le joueur 1 commencera la partie.

c) Définir une fonction membre publique `jouer` prenant en paramètres deux entiers. Cette fonction devra vérifier que les deux entiers correspondent bien à des indices de ligne et de colonne de la grille; si tel est le cas, la fonction devra vérifier si la case n'est pas déjà occupée. Dans le cas où la case possède la valeur `vide`, la fonction y affectera alors la valeur du joueur dont ce sera le tour. Dans tous les autres cas, la fonction devra afficher un message d'erreur et ne modifier ni la grille ni le joueur dont c'était le tour.

d) Écrire une fonction membre publique `gagnant` qui renverra une valeur de type `typecase`. Cette valeur sera `joueur1` ou `joueur2` si l'un des deux joueurs a gagné[1] et `personne` sinon.

e) On considère la partie suivante:

[1] c'est-à-dire qu'il ou elle possède trois cases sur la même ligne, la même colonne ou en diagonale.

```
Morpion unepartie;
// Joueur1 joue (0,0)
unepartie.jouer(0,0);
// Joueur2 joue (0,2)
unepartie.jouer(0,2);
// Joueur1 joue (1,0)
unepartie.jouer(1,0);
// Joueur2 joue (1,2)
unepartie.jouer(1,2);
// Joueur1 joue (2,0)
unepartie.jouer(2,0);
// Joueur2 joue (2,2)
unepartie.jouer(2,2);
```

Si aucune vérification n'a été faite dans l'écriture de la classe, cette partie peut se dérouler sans problème, et les deux joueurs peuvent alors gagner. Modifier (si besoin) la classe `Morpion` de sorte que cela ne puisse pas se produire.

Exercice 2 : Matrices et allocation dynamique

Dans cet exercice, on se propose de créer une classe `Matrice` permettant de coder une matrice d'éléments de type `double`) sous la forme d'un tableau unidimensionnel. Si $M = [M_{ij}]_{\substack{0 \leq i \leq n-1 \\ 0 \leq j \leq m-1}}$ est une matrice à n lignes et m colonnes, l'élément (i, j) de la matrice sera rangé en position $i \times m + j$ dans le tableau.

Pour l'ensemble de l'exercice, on utilisera un fichier en-tête `Matrice.h` contenant les définitions et déclarations propres à la classe `Matrice`, qui sera incorporé dans le fichier principal via la directive `#include`, ainsi qu'un fichier `Matrice.cpp` contenant le code des différentes fonctions membres de la classe. Lors de la compilation, on effectuera donc les instructions suivantes (où `testMatrice.cpp` représente le programme principal) :

```
g++ -c Matrice.cpp testMatrice.cpp
g++ -o testMatrice Matrice.o testMatrice.o
```

a) Définir la classe `Matrice`, avec ses constructeur et destructeur. Les dimensions de la matrice seront fixées lors de la construction de l'objet et passées en paramètres au constructeur. Elles seront aussi des membres données (privés) de la classe. Le constructeur pourra être appelé avec une seule dimension, auquel cas il s'agira du nombre de lignes de la matrice, le nombre de colonnes étant fixé à 1.

b) Définir une fonction membre `getElement` qui prenne en arguments deux entiers (i, j) et renvoie le coefficient (i, j) de la matrice liée à l'objet pour lequel cette fonction est appelée. Définir également une fonction membre `setElement` qui prenne en argument deux entiers i et j ainsi qu'une valeur v de type `double` et affecte la valeur v au coefficient (i, j) de la matrice.

c) Écrire une fonction membre `afficher` permettant d'afficher le contenu d'une matrice sur la sortie standard. L'affichage sera fait ligne par ligne.

d) Écrire une fonction membre `multiplication` prenant en paramètres deux références sur des instances de la classe `Matrice`, ainsi que leur nombre de colonnes (qui sera supposé être le même). La fonction effectuera le produit de la matrice correspondant à l'objet par lequel elle est appelée avec la matrice correspondant au premier argument de la fonction `multiplication`, et stockera le résultat dans la matrice liée au second argument.

e) Écrire une fonction `nb_instances` qui indique combien d'objets de la classe `Matrice` existent au moment où cette fonction est appelée. Ce nombre doit donc être accessible par chaque instance.