Université Paris Diderot
M2 Modélisation Aléatoire

Programmation C++
Année 2013-2014

TP n°3

Page du cours : `http://www.liafa.univ-paris-diderot.fr/~carton/Enseignement/C++/`

Exercice 1 Écrire une fonction `void echanger(int& a,int& b)` qui échange les valeurs des deux variables `a` et `b`.

Exercice 2 Implémenter une classe `Matrice` qui gère des matrices. La classe contiendra les champs privés :

- `int nbLigne;`
- `int nbColonne;`
- `double* matrice;`

Et les méthodes publiques :

- `Matrice(int nbLigne,int nbColonne) //Le constructeur`
- `~Matrice() //Le destructeur`
- `void setCase(int ligne,int colonne,double valeur)`
- `Matrice(Matrice & matrice)// le constructeur de recopie`
- `int getNbLignes()`
- `int getNbColonne()`
- `int getCase(int ligne,int colonne)`
- `Matrice multiplie(Matrice & matrice)`
- `Matrice estMultipliable(Matrice & matrice)`
- `Matrice ajoute(Matrice & matrice)`
- `Matrice estMemeDim(Matrice & matrice)`

`matrice` est un tableau qui contiendra les coefficients de la matrices. `nbLigne` est le nombre de ligne de la matrice. `nbColonne` le nombre de colonne. Le tableau doit donc être de taille `nbLigne*nbColonne`. Le coeficient i, j de la matrice sera contenu dans la case `(i-1)*nbColonne+j-1` du tableau `matrice`.

Lors de la construction d'une nouvelle liste avec le constructeur, La place nécessaire pour le tableau `matrice` sera allouée avec `new`. Il faut implémenter le destructeur de tel façon qu'il dessaloue la place avec `delete`.

Les fonctions `multiplie` et `ajoute` renvoient le produit ou la somme de deux matrices.

Les fonctions `estMultipliable` et `estMemeDim` dise si l'insance est multipliable avec l'argument respectivement de même dimension.

Exercice 3 Implémentez une classe `Liste` qui gère des listes d'entiers. La classe contiendra les champs privés :

- `int taille;`
- `int tailleMax;`
- `int* tableau;`

Et les méthodes publiques :

- `Liste() //Le constructeur`
- `~Liste() //Le destructeur`
- `void ajoute(int a)`
- `Liste recopie()`
- `int getTaille()`
- `void setCase(int indice,int valeur)`
- `int getCase(int a)`

`tableau` contiendra les entiers de la liste et `taille` le nombre d'entiers qu'elle contient. Lors de la construction d'une nouvelle liste avec le constructeur, la place allouée pour le tableau sera de 100 cases, elle sera étendu si nécessaire. `tailleMax` indiquera en permanence le nombre de cases allouées pour le tableau, (ces cases sont toutes vides dans une liste nouvellement crée), donc pour une nouvelle liste `tailleMax` vaut 100.

La méthode `ajoute` ajoute un entier à la fin de la liste (à droite). Si le `tableau` est plein il faut penser à allouer un tableau plus grand et à recopier dans ce tableau les valeurs de l'ancien tableau. Ensuite il faut libérer la mémoire occupée par l'ancien tableau. Le nombre de cases allouées au `tableau` (valeur de `tailleMax`) sera toujours multiple de 100. (Si la liste contient 235 entiers alors `taille` vaut 235 et `tailleMax` vaut 300.

La méthode `recopie` fabrique une nouvelle liste qui contient la même liste d'entiers que l'instance.

La méthode `setCase` (res. `getCase`) modifie (res. renvoie) la valeur de la case `indice` de la liste. Si l'utilisateur utilise l'indice d'une case supérieur ou égale à `taille` alors la méthode `setCase` (res. `getCase`) ne fait rien. (On verra plus tard comment gérer une mauvaise utilisation de la part de l'utilisateur).

La méthode `getTaille` renvoie la taille de la liste.

Le destructeur desalloue la place occupé par le `tableau`.