

POO
LICENCE 3 INFO – L3 MIAGE
TD1

Exercice 1

Soit la structure suivante définissant une matrice :

```
struct Matrice {  
    int nb_lignes ;  
    int nb_colonnes ;  
    float **elements ;  
};
```

Ecrire les fonctions suivantes :

1. *creer_matrice*, elle reçoit en argument le nombre de lignes et le nombre de colonnes, crée dynamiquement une matrice, l'initialise à 0 et retourne le pointeur sur la matrice,
2. *détruire_matrice*, qui permet de libérer l'espace occupé par une matrice,
3. *afficher_matrice* : affiche les différents éléments de la matrice,
4. *transposee_matrice* : calcule le transposé de la matrice
5. *matrice_produit* : reçoit en argument les pointeurs des deux matrices, calcule leur produit, et retourne le pointeur sur la matrice résultat du produit.
6. une fonction main() pour tester ces différentes fonctions.

Exercice 2

On considère le type abstrait suivant permettant de manipuler des nombres complexes. Les opérations permises sur des objets de ce type seront les suivantes :

Complexe :	(double, double)	->	Complexe
reelle :	(Complexe)	->	double
imaginaire :	(Complexe)	->	double
module :	(Complexe)	->	double
ajouter :	(Complexe, Complexe)	->	Complexe
multiplier :	(Complexe, Complexe)	->	Complexe
afficher :	(Complexe)	->	

Plus précisément, les opérations « *reelle* » et « *imaginaire* » permettront d'obtenir les parties réelle et imaginaire d'un complexe donné. Les suivantes permettront de déterminer le module d'un complexe donné, d'ajouter ou multiplier deux complexes. Pour finir, la dernière opération permettra d'afficher un nombre complexe donné au format « *x + y.i* » où « *x* » et « *y* » sont ses parties réelle et imaginaire.

- 1) Ecrivez dans un fichier « *complexe.h* » la définition d'une classe « *Complexe* » correspondant exactement au type défini précédemment (sans surdéfinitions d'opérateurs).
- 2) Ecrivez dans un fichier « *complexe.cpp* » les définitions des fonctions membres de la classe « *Complexe* ». Afin de visualiser les appels au constructeur, ce dernier devra afficher à chaque fois ses paramètres.

- 3) Ecrivez dans le même fichier source une fonction « main » permettant de tester l'implémentation du type abstrait « Complexe ». Expliquez les divers appels au constructeur lors de son exécution.
- 4) Modifiez la classe « Complexe » définie précédemment en implémentant les opérations « ajouter » et « multiplier » sous la forme d'opérateurs surdéfinis « + » et « * ».
- 5) Surdéfinissez l'opérateur « << » de manière à pouvoir écrire un complexe sur un flot de sortie de type « ostream ».
- 6) Modifiez la fonction « main » écrite précédemment pour tester les surdéfinitions des divers opérateurs.

Exercice 3 (TP)

On considère le type abstrait suivant permettant de manipuler des rectangles. Les opérations permises sur des objets de ce type seront les suivantes :

Rectangle :	(unsigned int, unsigned int)	->	Rectangle
GetLargeur:	(Rectangle)	->	unsigned int
GetHauteur:	(Rectangle)	->	unsigned int
perimetre :	(Rectangle)	->	unsigned int
surface :	(Rectangle)	->	unsigned int
SetLargeur :	(Rectangle , unsigned int)	->	Rectangle
SetHauteur :	(Rectangle , unsigned int)	->	Rectangle
Compare :	(Rectangle , Rectangle)	->	booléen
afficher :	(Rectangle)	->	

Plus précisément, les opérations « GetLargeur » et « GetHauteur » permettront d'obtenir la largeur et la hauteur d'un rectangle donné. Les opérations « perimetre » et « surface » permettront de déterminer le périmètre et la surface d'un rectangle donné. Les opérations « SetLargeur », « SetHauteur » permettront respectivement de modifier la largeur et la hauteur d'un rectangle et « Compare » de comparer deux rectangles. Pour finir, la dernière opération permettra d'afficher un rectangle en utilisant un signe.

- 7) Ecrivez dans un fichier « Rectangle.h » la définition d'une classe « Rectangle » correspondant exactement au type défini précédemment (sans surdéfinitions d'opérateurs).
- 8) Ecrivez dans un fichier « Rectangle.cpp » les définitions des fonctions membres de la classe « Rectangle ». Afin de visualiser les appels au constructeur, ce dernier devra afficher à chaque fois ses paramètres.
- 9) Ecrivez dans un fichier source « main.cpp » une fonction « main » permettant de tester l'implémentation du type abstrait « Rectangle ». Expliquez les divers appels au constructeur lors de son exécution.
- 10) Modifiez la classe « Rectangle » définie précédemment en implémentant l'opération « Compare » sous la forme d'un opérateur surdéfini « == ».
- 11) Surdéfinissez l'opérateur « << » de manière à pouvoir écrire un rectangle sur un flot de sortie de type « ostream ».
- 12) Modifiez la fonction « main » écrite précédemment pour tester la surdéfinition de l'opérateur.

Exercice 4 (TP)

On considère le type abstrait suivant permettant de manipuler des cercles. Les opérations permises sur des objets de ce type seront les suivantes :

Cercle	:	(unsigned int, unsigned int, unsigned int)	->	Rectangle
<i>GetRayon</i>	:	(Cercle)	->	<i>unsigned int</i>
<i>GetCentre</i> :		(Cercle)	->	Centre
<i>perimetre</i>	:	(Cercle)	->	<i>unsigned int</i>
<i>surface</i>	:	(Cercle)	->	<i>unsigned float</i>
<i>SetRayon</i>	:	(Cercle , <i>unsigned int</i>)	->	Cercle
<i>SetCentre</i>	:	(Cercle , Centre)	->	Cercle
<i>Compare</i>	:	(Cercle Cercle)	->	booléen
<i>afficher</i>	:	(Cercle)	->	

Plus précisément, les opérations « GetRayon » et « GetCentre » permettront d'obtenir le rayon et le centre d'un cercle donné. Les opérations « perimetre » et « surface » permettront de déterminer le périmètre et la surface d'un cercle donné. Les opérations « SetRayon », « SetCentre» permettront respectivement de modifier le rayon et le centre d'un cercle et « Compare » de comparer deux cercles. Pour finir, la dernière opération permettra d'afficher un cercle en utilisant un signe.

- 13) Écrivez dans un fichier « Cercle.h » la définition d'une classe « Cercle » correspondant exactement au type défini précédemment (sans surdéfinitions d'opérateurs).
- 14) Écrivez dans un fichier « Cercle.cpp » les définitions des fonctions membres de la classe « Cercle ». Afin de visualiser les appels au constructeur, ce dernier devra afficher à chaque fois ses paramètres.
- 15) Écrivez dans un fichier source « main.cpp » une fonction « main » permettant de tester l'implémentation du type abstrait « Cercle ». Expliquez les divers appels au constructeur lors de son exécution.
- 16) Modifiez la classe « Cercle » définie précédemment en implémentant l'opération « Compare » sous la forme d'un opérateur surdéfini « == ».
- 17) Surdéfinissez l'opérateur « << » de manière à pouvoir écrire un cercle sur un flot de sortie de type « ostream ».
- 18) Modifiez la fonction « main » écrite précédemment pour tester la surdéfinition de l'opérateur.

POO
LICENCE 3 INFO – L3 MIAGE
TD2

Exercice 1

En langage C, il n'existe pas de type chaîne de caractères. On considère qu'une chaîne de caractères est un tableau de caractères (se terminant par le caractère nul). Définir une classe nommée *str* offrant des possibilités plus proches d'un véritable type chaîne. Pour cela, il faudra prévoir comme données membres :

- la longueur de la chaîne,
- l'adresse d'une zone allouée dynamiquement, destinée à recevoir la suite de caractères (il ne sera pas nécessaire d'y ranger le caractère nul de fin, puisque la longueur de la chaîne est définie par ailleurs).

Le contenu d'un objet de type *str* pourra donc évoluer par un simple jeu de gestion dynamique.

On munira la classe *str* des constructeurs suivants :

- *str()* : initialise une chaîne vide ;
- *str (char *)* : initialise la chaîne avec la chaîne (au sens de C) dont on lui fournit l'adresse en argument
- *str (str &)* : constructeur de recopie

et d'un destructeur.

On définira les opérateurs :

- *=* pour l'affectation entre objets de type *str* (penser à l'affectation multiple)
- *==* pour examiner l'égalité de deux chaînes
- *+* pour réaliser la concaténation de deux chaînes. Si *a* et *b* sont de type *str*, *a + b* sera une nouvelle chaîne formée de la concaténation de *a* et *b*.
- *[]* pour accéder à un caractère de rang donné d'une chaîne (les affectations de la forme *a[i] = 'x'* devront pouvoir fonctionner).

On ajoutera une fonction d'affichage.

Donnez un exemple d'appel de chaque fonction membre dans une fonction *main* (qu'on supposera dans un fichier *main.cpp*).

N.B. Il ne faudra pas utiliser la classe *string* de la bibliothèque standard.

Exercice 2

Vous allez définir un type de données abstrait permettant de manipuler des ensembles d'entiers sans répétition (un ensemble est dit sans répétition s'il ne peut pas contenir deux fois le même élément). Les opérations permises sur des objets de ce type seront les suivantes :

<i>Ensemble</i>	:	(int)	->	<i>Ensemble</i>
<i>cardinal</i>	:	(<i>Ensemble</i>)	->	<i>int</i>
<i>ajouter</i>	:	(<i>Ensemble, int</i>)	->	<i>Ensemble</i>
<i>supprimer</i>	:	(<i>Ensemble, int</i>)	->	<i>Ensemble</i>
<i>contient</i>	:	(<i>Ensemble, int</i>)	->	<i>Ensemble</i>
<i>~Ensemble</i>	:	(<i>Ensemble</i>)	->	
<i>afficher</i>	:	(<i>Ensemble</i>)	->	

Plus précisément, ces opérations devront permettre :

- i) de créer un ensemble pouvant contenir au plus un nombre N d'éléments passé en argument,
 - ii) de déterminer le cardinal d'un ensemble,
 - iii) d'ajouter ou de supprimer un élément (un entier) d'un ensemble,
 - iv) de tester si un ensemble contient un élément donné,
 - v) de détruire un ensemble,
 - vi) d'afficher le contenu d'un ensemble au format $E = [x_1, x_2, \dots, x_n]$ où x_1, x_2, \dots, x_n sont ses éléments.
- 1) Ecrivez dans un fichier « ensemble.h » la définition d'une classe « Ensemble » correspondant exactement au type défini précédemment (sans surdéfinitions d'opérateurs). Les éléments d'un ensemble donné devront être mémorisés dans un tableau d'entier alloué dynamiquement.
 - 2) Ecrivez dans un fichier « ensemble.cpp » les définitions des fonctions membres de la classe « Ensemble ». Afin de visualiser les appels aux constructeur et destructeur, ces derniers devront afficher l'adresse des objets les ayant appelé.
 - 3) Ecrivez dans le même fichier source une fonction « main » permettant de tester l'implémentation du type abstrait « Ensemble ».
 - 4) Ecrivez un constructeur de recopie pour la classe « Ensemble ».
 - 5) Mettez en évidence ses principales conditions d'appels automatiques. Vous pourrez programmer pour cela une fonction « f » bien choisie.
 - 6) Modifiez la classe « Ensemble » définie précédemment en implémentant les opérations « ajouter », « supprimer » et « contient » sous la forme d'opérateurs surdéfinis « << », « >> » et « % » utilisables comme suit :


```

Ensemble e(10);      // Ensemble d'au plus 10 éléments
e << 2 << 10 << 3; // Ajoute 2, 10 et 3 à l'ensemble e
e >> 10 >> 2;      // Supprime 2 et 10 de l'ensemble e
if (e % 3)            // Teste si 3 est élément de e
    cout << "3 est element de E";
      
```
 - 7) Implémentez une surdéfinition de l'opérateur d'affectation « = » pour la classe « Ensemble ».
 - 8) Proposez des surdéfinitions des opérateurs « + » et « * » correspondant aux opérations d'union et d'intersection.

- 9) Ajoutez à la définition de la classe « Ensemble » des fonctions membres « init », « existe » et « prochain » permettant de parcourir l'ensemble des éléments d'un ensemble pour leur appliquer le même traitement. Plus précisément :
- « init » initialisera le parcours d'un ensemble,
 - « existe » testera s'il y a encore des éléments à traiter,
 - « prochain » retournera le prochain élément à traiter.

Si « e » est un objet de la classe « Ensemble », ces fonctions pourront s'utiliser comme suit :

```
e.init();
while (e.existe())
    traiter(e.prochain());
```

- 10) A partir des fonctions précédentes, écrivez une fonction « somme » retournant la somme des entiers d'un ensemble donné ; vous ne définirez pas cette nouvelle fonction comme une fonction membre de la classe « Ensemble ».

Exercice 3

On considère le type abstrait suivant permettant de manipuler des matrices. Les opérations permises sur des objets de ce type seront les suivantes :

Sorte Matrice
Utilise Elément, Entier

Opérations

<i>Matrice</i>	<i>: (int, int)</i>	<i>-> Matrice</i>
<i>Matrice</i>	<i>: Matrice</i>	<i>-></i>
<i>GetElements</i>	<i>: Matrice x Entier x Entier</i>	<i>-> Element</i>
<i>SetElements</i>	<i>: Matrice x Entier x Entier x Elément</i>	<i>-> Matrice</i>
<i>produit</i>	<i>: Matrice x Matrice</i>	<i>-> Matrice</i>
<i>afficher</i>	<i>: Matrice</i>	<i>-></i>

Plus précisément, ces opérations devront permettre :

- de créer une matrice à partir du nombre de lignes et du nombre colonnes passés en argument,
 - de détruire une matrice,
 - de récupérer l'élément à la ligne i et à la colonne j de la matrice,
 - d'écrire l'élément à la ligne i et à la colonne j de la matrice,
 - de faire le produit de deux matrices,
 - d'afficher le contenu d'une matrice sous sa forme habituelle.
- Ecrivez dans un fichier « Matrice.h » la définition d'une classe « Matrice » correspondant exactement au type défini précédemment (sans surdéfinitions d'opérateurs).
 - Ecrivez dans un fichier « Matrice.cpp » les définitions des fonctions membres de la classe « Matrice ». Afin de visualiser les appels aux constructeur et destructeur, ces derniers devront afficher l'adresse des objets les ayant appelé.

- 3) Ecrivez un constructeur de recopie pour la classe « Matrice ». Pour mettre en évidence une de ses conditions d'appel automatique, vous écrirez une fonction « somme » qui reçoit comme argument une matrice et calcule la somme de ses éléments. Dire comment se rendre compte de cet appel lors de l'exécution.
- 4) Implémentez une surdéfinition de l'opérateur d'affectation « = » pour la classe « Matrice ».
- 5) Ecrivez dans un fichier « main.cpp » une fonction « main » permettant de tester l'implémentation du type abstrait « Matrice ». Cette fonction fera appel aux différentes fonctions membres.

Exercice 4

Un enseignant d'une université est reconnu par son nom, son prénom et son diplôme. Il peut enseigner jusqu'à cinq matières.

- 1) Ecrire une classe « Enseignant » (Enseignant.h et Enseignant.cpp) avec les fonctions membres suivantes :

- Un constructeur
- Un constructeur de recopie
- Un destructeur
- Une méthode qui permet de récupérer le nom
- Une méthode qui permet de récupérer le prénom
- Une méthode qui permet de récupérer le diplôme
- Une méthode qui attribue une matière à un enseignant
- Une méthode qui affiche l'enseignant avec les matières qu'il enseigne
- Une surdéfinition de l'opérateur d'affectation

Les chaînes de caractères seront définies au sens de C.

- 2) Ecrire une fonction main qui crée des enseignants, leur affecte un certain nombre de matières et les affiche.
- 3) Rajouter une situation d'invocation du constructeur de recopie et une affectation d'un enseignant à un autre.

Exercice 5

Les opérations possibles sur une pile sont les suivantes :

<i>Pile</i>	:	<i>Pile</i>	->	<i>Pile</i>	// Construit une pile vide
<i>~Pile</i>	:	<i>Pile</i>	->		// Détruit une pile
<i>empiler</i>	:	<i>Pile x Element</i>	->	<i>Pile</i>	// Ajoute un élément en haut de la pile
<i>depiler</i>	:	<i>Pile</i>	->	<i>Pile</i>	// Supprime l'élément en haut de la pile
<i>sommet</i>	:	<i>Pile</i>	->	<i>Element</i>	// Retourne l'élément en haut de la pile
<i>est_vide</i>	:	<i>Pile</i>	->	<i>bool</i>	// Teste si une pile est vide ou non
<i>afficher</i>	:	<i>Pile</i>	->		// Affiche le contenu d'une pile

1. Définissez dans un fichier « pile.h » une classe « Pile » correspondant au type précisé ci-dessus. La pile, dans ce cas, sera réalisée sous forme de liste simplement chaînée.

La donnée pourra être considérée comme un entier.

Indication : déclarer dans un premier temps une structure de liste chaînée (*Pile_elt*) et

ensuite comme donnée membre de la liste le pointeur sur la tête de la liste (*tete*).

2. Définissez les fonctions membres de la classe « Pile » dans un fichier « pile.cpp ». L'affichage du contenu d'une pile se fera au format « e3 / e2 / e1 » si e1, e2 et e3 sont les éléments empilés successivement. Vous testerez votre implémentation du type « Pile » en construisant et manipulant des piles d'entiers (dans le fichier « main.cpp »).

Exercice 6

On considère de nouveau le type abstrait de « Ensemble » de l'exercice 2.

- 1) Ecrivez dans un fichier « Ensemble.h » la définition d'une classe « Ensemble » correspondant exactement au type défini précédemment (sans surdéfinitions d'opérateurs). L'ensemble, dans ce cas, sera réalisé sous forme de liste simplement chaînée. La donnée est considérée comme un entier.

Indication : déclarer dans un premier temps une structure de liste chaînée (*element*) et ensuite comme données membres de la liste le pointeur sur le début de la liste (*debut*) et le pointeur sur l'élément courant (*courant*).

- 2) Ecrivez dans un fichier « ensemble.cpp » les définitions des fonctions membres de la classe « Ensemble ».
- 3) Ecrivez un constructeur de recopie pour la classe « Ensemble ».
- 4) Implémentez une surdéfinition de l'opérateur d'affectation « = » pour la classe « Ensemble ».
- 5) Ecrivez dans le même fichier source une fonction « main » permettant de tester l'implémentation du type abstrait « Ensemble ». Vous mettrez en évidence une situation d'invocation du constructeur de recopie et ferez l'affectation entre ensembles.
- 6) Modifiez la classe « Ensemble » définie précédemment en implémentant les opérations « ajouter », « supprimer » et « contient » sous la forme d'opérateurs surdéfinis « << », « >> » et « % » utilisables comme suit :

```
Ensemble e; // Création d'un ensemble vide  
e << 2 << 10 << 3; // Ajoute 2, 10 et 3 à l'ensemble e  
e >> 10 >> 2; // Supprime 2 et 10 de l'ensemble e  
if (e % 3) // Teste si 3 est élément de e  
    cout << "3 est element de E";
```

- 7) Ajoutez à la définition de la classe « Ensemble » des fonctions membres « init », « existe » et « prochain » permettant de parcourir l'ensemble des éléments d'un ensemble pour leur appliquer le même traitement. Plus précisément :
- « init » initialisera le parcours d'un ensemble,
 - « existe » testera s'il y a encore des éléments à traiter,
 - « prochain » retournera le prochain élément à traiter.

Si « e » est un objet de la classe « Ensemble », ces fonctions pourront s'utiliser comme suit :

```
e.init();
while (e.existe())
    traiter(e.prochain());
```

- 8) A partir des fonctions précédentes, écrivez une fonction « somme » retournant la somme des entiers d'un ensemble donné ; vous ne définirez pas cette nouvelle fonction comme une fonction membre de la classe « Ensemble ».

POO
Licence INFO 3 – L3 MIAGE
TD3

Exercice 1

Créer un patron de classes permettant de représenter des « vecteurs dynamiques », c'est-à-dire des vecteurs dont la dimension peut ne pas être connue lors de la compilation. On prévoira que les éléments de ces vecteurs puissent être de type quelconque.

On surdéfinira convenablement [] pour qu'il permette l'accès aux éléments du vecteur (aussi bien en consultation qu'en modification) et on s'arrangera pour qu'il n'existe aucun risque de "débordement d'indice". En revanche, on ne cherchera pas à régler les problèmes posés éventuellement par l'affectation ou la transmission par valeur d'objets du type concerné.

On veut maintenant réaliser un patron de classes permettant de manipuler des vecteurs dont les éléments sont de type quelconque mais pour lesquels la dimension, supposée être cette fois une expression constante, apparaîtra comme un paramètre du patron.

Exercice2 (TP)

On veut implémenter un site de documents. Le site doit avoir les propriétés suivantes :

- Ajouter un document au site
- Supprimer un document du site
- Copier tous les documents d'un site vers un autre
- Rechercher un document dans le site (par mots-clés)

Ecrire une classe « Site » qui sera muni d'un constructeur, d'un destructeur et éventuellement d'autres méthodes jugées utiles.

Un document sera défini par son titre, sa liste de mots-clés.

Définir les classes « Document » et « Site »

Rajouter une fonction « main » qui utilise les documents et le site.

Exercice 3 (TD / TP)

Un salarié d'une école est reconnu par son nom et son prénom. On distingue deux sortes de salariés :

- les administratifs
- les professeurs

Un professeur a un diplôme et peut enseigner jusqu'à dix matières. Un administratif a une fonction. Dans les deux cas, on doit pouvoir consulter par affichage les caractéristiques d'un salarié.

La classe « Salarie » sera munie d'un constructeur, d'un destructeur et de deux méthodes qui permettent de récupérer respectivement le nom et le prénom.

La classe « Administrateur » sera munie d'un constructeur, d'un destructeur et d'une méthode qui permet de récupérer la fonction.

La classe « Professeur » sera munie d'un constructeur, d'un destructeur, d'une méthode qui permet de récupérer le diplôme, d'une méthode qui permet de récupérer la i-ème matière enseignée et finalement d'une méthode qui permet d'ajouter une matière à la liste des matières enseignées.

Les chaînes de caractères seront dans considérées au sens de C

On vous demande d'écrire en C++ les classes (fichiers « .h » et « .cpp ») « Salarie », « Administratif » et « Professeur ».

Rajouter une fonction « main » qui utilise ces différentes classes.

Exercice 4 (TP)

Ecrire une classe qui hérite de la classe str définie dans la fiche de td précédente, et rajoute une couche permettant de gérer le formatage du texte. Le formatage peut être italique (*Italic*), gras (**Bold**) et coloré (*Couleur*).

La classe comprendra les fonctions membre suivantes :

- un constructeur par défaut ;
- un constructeur à partir d'une chaîne de caractères (de type char *) et éventuellement les options de formatage ;
- un constructeur à partir d'une chaîne de caractères (de type str) et éventuellement les options de formatage ;
- un constructeur de recopie ;
- un destructeur
- une surcharge de l'opérateur d'affectation pour le cas d'une copie de chaîne formatée ;
- une surcharge de l'opérateur d'affectation pour le cas d'une copie de chaîne non formatée (classe de base);
- une méthode qui permet de mettre en italique ;
- une méthode qui permet de mettre en gras (**Bold**) ;
- une méthode qui permet de colorier ;
- un affichage de la chaîne et des informations de formatage, en utilisant la notation HTML, à savoir :
 - <i> pour la mise en italique et </i> à la fin;
 - pour la mise en gras et à la fin ;
 - pour la couleur et à la fin.

Exercice 5 (TD)

Soit une classe « vect » permettant de manipuler des « vecteurs dynamiques » d'entiers (c'est à dire dont la dimension peut être fixée au moment de l'exécution) dont la déclaration (fournie dans le fichier vect.h) se présente comme suit :

```
class vect {  
    int n ;                                // Nombre d'éléments  
    int * adr ;                            // adresse zone dynamique contenant les éléments  
public :  
    vect(int) ;                          // constructeur (précise la taille du vecteur)  
    ~vect() ;                            // destructeur
```

```
int & operator [] (int) ; // accès à un élément par son indice.
```

```
}
```

Créer une classe vectb, dérivée de vect, permettant de manipuler des vecteurs dynamiques, dans lesquels on peut fixer les « bornes » des indices, lesquelles seront fournies au constructeur de vectb. La classe vect apparaîtra ainsi comme un cas particulier de vectb (un objet de type vect étant un objet de type vectb dans lequel la limite inférieure de l'indice est 0).

On ne se préoccupera pas, ici, des problèmes éventuellement posés par la recopie ou l'affectation d'objets de type vectb.

Exercice 6 (TD)

On considère la classe Tableau suivante :

```
class Tableau {  
    int * tab ;  
    int max ;  
public :  
    Tableau (int l) {  
        max = l ;  
        tab = new int[max] ;  
    }  
    int & operator [] (int i) {  
        // contrôle de la valeur de i  
        return tab[i] ;  
    }  
}
```

Une pile est un tableau particulier dont le comportement est « dernier entré, premier sorti ». Ecrire à partir de la classe « Tableau » une classe « Pile » avec comme donnée membre le niveau de remplissage de la pile et comme fonctions membres : le constructeur, une fonction qui permet de savoir si la pile est vide, un empilement et un dépilement.

Exercice 7

On considère la classe « Liste » suivante :

```
struct Place {  
    void* element;  
    Place* suivant;  
};  
  
class Liste {  
    int m_longueur;  
    Place* m_premier;  
    Place* m_courant;  
public:  
    Liste();  
    Liste(Liste&);  
    ~Liste();  
    Liste& operator=(Liste&);  
    // Ajoute un élément à la i-ème position  
    // avec : 0 < i <= m_longueur + 1
```

```

int ajouter(int i, void* pe);
// Supprime l'element a la i-eme position
// avec : 0 < i <= m_longueur
int supprimer(int i);
void* ieme(int i);
int longueur() { return(m_longueur); }
void init() { m_courant = m_premier->suivant; }
int existe() { return(m_courant != 0); }
void* prochain() {
    void* temp = m_courant->element;
    m_courant = m_courant->suivant;
    return(temp);
}
};


```

En partant de définitions d'une pile et d'une file à partir d'une liste, on veut créer à partir de cette classe « Liste », les classes « Pile » et « File ».

1. Ecrire dans les fichiers « Pile.h » et « Pile.cpp » les déclarations et définitions des fonctions membres suivantes : un constructeur, un destructeur, une fonction d'empilement, une fonction qui permet de récupérer le sommet de la pile, une fonction de dépilement, une fonction qui teste si la pile est vide et finalement une fonction qui récupère la longueur de la pile.
2. Ecrire dans les fichiers « File.h » et « File.cpp » les déclarations et définitions des fonctions membres suivantes : un constructeur, un destructeur, une fonction d'enfillement, une fonction qui permet de récupérer la tête de la file, une fonction de défilement, une fonction qui teste si la file est vide et finalement une fonction qui récupère la longueur de la file.
3. Expliciter clairement les choix que vous avez faits pour l'implémentation des fonctions membre de 1. et de 2. Donnez ensuite un exemple d'utilisation de 1. et de 2. en montrant que cela correspond bien aux choix que vous avez faits.

Exercice 8 (TD /TP)

Soit une classe définie comme suit :

```

Class Compte {
    long numero;
    double solde;
public :
    Compte(long);
    void affiche_toi();
    void versement(double);
    bool retrait (double);
}


```

1. Déclarer la classe dans un fichier « Compte.h » et définir ses fonctions membres dans un fichier « Compte.cpp ». Pour attribuer un numéro à un compte, on utilisera dans un premier temps une variable statique de la classe Compte par exemple numero_a utiliser, qui sera initialisée à 1000 par exemple et qui sera incrémentée de 1 à chaque création de compte. (On n'essaiera pas de récupérer les numéros de compte détruits).
2. Créer une fonction « main » qui permet de tester ces fonctions.

3. Déclarer et définir les classes « CompteCour » et « ComptEpargne ». Le Compte Courant est un Compte sur lequel on autorise un découvert. On devra donc redéfinir la fonction retrait pour les objets CompteCour. Le ComptEpargne est un Compte qui produit des intérêts, il possède un taux de rémunération et une méthode de calcul des intérêts dont nous considérerons qu'elle est le produit du solde par le taux d'intérêt. Eventuellement certains attributs privés de Compte pourront changer de statut.
4. Réécrire une fonction « main » qui permet de tester ces nouvelles fonctions.