

**POO**  
**LICENCE 3 INFO**  
**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
<i>reelle</i> :	(Complexe)	->	double
<i>imaginaire</i> :	(Complexe)	->	double
<i>module</i> :	(Complexe)	->	double
<i>ajouter</i> :	(Complexe, Complexe)	->	Complexe
<i>multiplier</i> :	(Complexe, Complexe)	->	Complexe
<i>afficher</i> :	(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.

Lamine Guéye  
 Université Gaston Berger  
 Saint-Louis  
 UFR SAT  
 Section Informatique

**POO**  
**LICENCE 3 INFO**  
**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>	<i>: (int)</i>	<i>-&gt; Ensemble</i>
<i>cardinal</i>	<i>: (Ensemble)</i>	<i>-&gt; int</i>
<i>ajouter</i>	<i>: (Ensemble, int)</i>	<i>-&gt; Ensemble</i>
<i>supprimer</i>	<i>: (Ensemble, int)</i>	<i>-&gt; Ensemble</i>
<i>contient</i>	<i>: (Ensemble, int)</i>	<i>-&gt; Ensemble</i>
<i>~Ensemble</i>	<i>: (Ensemble)</i>	<i>-&gt;</i>
<i>afficher</i>	<i>: (Ensemble)</i>	<i>-&gt;</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>-&gt; Matrice</i>
<i>~Matrice</i>	<i>: Matrice</i>	<i>-&gt;</i>
<i>GetElements</i>	<i>: Matrice x Entier x Entier</i>	<i>-&gt; Element</i>
<i>SetElements</i>	<i>: Matrice x Entier x Entier x Elément</i>	<i>-&gt; Matrice</i>
<i>produit</i>	<i>: Matrice x Matrice</i>	<i>-&gt; Matrice</i>
<i>afficher</i>	<i>: Matrice</i>	<i>-&gt;</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 :

```

Pile      :          -> Pile      // Construit une pile vide
~Pile     : Pile      ->           // Detruit une pile
empiler   : Pile x Element -> Pile    // Ajoute un élément en haut de la
                                              // pile
depiler   : Pile      -> Pile    // Suprime l'élément en haut de la pile
sommet    : Pile      -> Element // Retourne l'élément en haut de la
                                              // pile
est_vide  : Pile      -> bool   // Teste si une pile est vide ou non
afficher  : Pile      ->           // 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 (*début*) 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 :
- i) « init » initialisera le parcours d'un ensemble,
  - ii) « existe » testera s'il y a encore des éléments à traiter,
  - iii) « prochain » retournera le prochain élément à traiter.

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

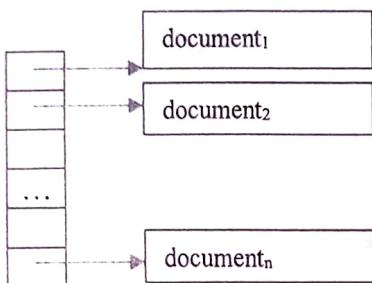
```
e.init();
while (e.existe())
    traiter(e.prochain());
```

- 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 ».

**Programmation Orientée Objet**  
**Contrôle Continu**  
**Licence 3 INFO**  
**Durée : 2h**  
**Documents non autorisés**

**Exercice 1**

- I. On considère une table représentée comme suit :



La table contient le nombre maximal d'éléments, le nombre courant et les éléments. Un document est constitué de 3 attributs : une clé (un entier) qui permet de l'identifier, une valeur (une chaîne de caractères correspondant au titre du document) et la liste de ses mots-clés. Ainsi, par exemple, la recherche d'un document dans la table se fera par la clé.

Les chaînes de caractères seront considérées au sens de c.

Les opérations possibles sur la table sont les suivantes :

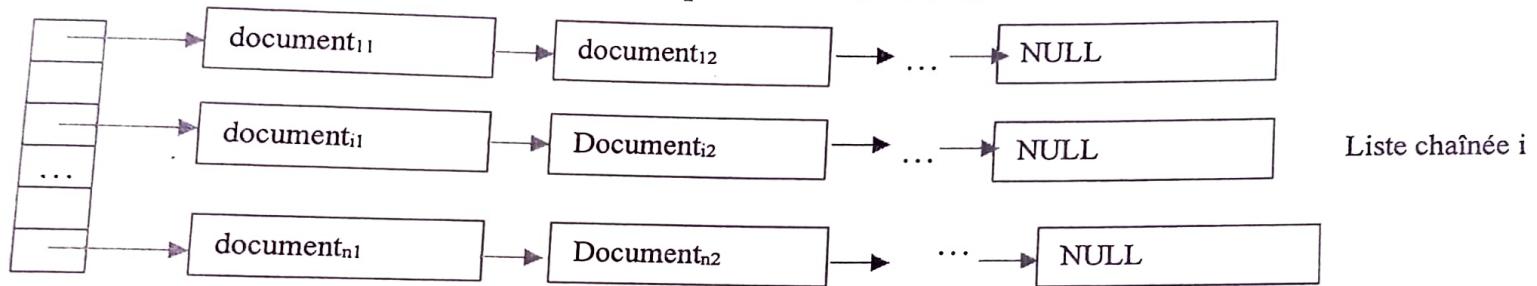
<i>ConstruitTable : int</i>	<i>-&gt; Table</i>	<i>// Construit une table vide</i>
		<i>// d'au plus max éléments</i>
<i>DétruitTable : Table -&gt;</i>		<i>// Détruit une table</i>
<i>insertion : Table x Document</i>	<i>-&gt; Table</i>	<i>// Insère le document de clé i et de</i>
		<i>// valeur v dans la table si ce</i>
		<i>// dernier ne s'y trouve pas</i>
<i>supprimer : Table x i</i>	<i>-&gt; Table</i>	<i>// Retire le document de clé i de</i>
		<i>// la table</i>
<i>rechercher : Table x i</i>	<i>-&gt; Booléen</i>	<i>// Recherche le document de clé i</i>
		<i>// dans la table</i>

1. Définissez dans un fichier « Table.h » une classe « Table » correspondant au type précisé ci-dessus.
2. Définissez les fonctions membres de la classe « Table » dans un fichier « Table.cpp ». L'affichage du contenu d'une table se fera au format « d1 -> d2 -> d3 » si d1, d2 et d3 sont les documents ajoutés successivement.
3. Ajoutez un constructeur de recopie à la classe « Table ».
4. Surdéfinissez l'opérateur d'affectation de la classe « Table ».

5. Modifiez la classe « Table » définie précédemment en implémentant les opérations « insertion », « supprimer » et « rechercher » sous la forme d'opérateurs surdéfinis « << », « >> » et « % » utilisables comme suit :

```
t << d1 << d5 << d3; // Ajoute les documents d1, d5 et d3 à la table t
t >> d4 >> d2; // Supprime les documents d4 et d2 de la table t
if (t % d3) // Teste si d3 est un document appartenant à la table t
    cout << "d3 est un document de la table t";
```

- II. On considère maintenant la table représentée comme suit :



La table contient toujours le nombre maximal d'éléments, le nombre courant et les éléments

1. Définissez dans un fichier « Table.h » une classe « Table » correspondant au type précisé ci-dessus.
2. Définissez les méthodes « ConstruitTable », « DetruitTable », « insertion » et « rechercher ». Pour l'ajout comme pour la recherche, le numéro de la liste chaînée utilisée est obtenu à partir de la clé par l'opération suivante : clé % max, où max est le nombre maximal d'éléments du tableau. L'ajout se fait en tête.
3. Vous testerez votre implémentation du type abstrait « Table » en construisant et manipulant des tables de documents dans le fichier « main.cpp ».

Programmation Orientée Objet  
Examen Première Session  
Licence  
Durée : 2h  
Documents non autorisés

Exercice 1

On veut écrire une application pour gérer une boîte aux lettres pouvant recevoir des courriers qui peuvent être des lettres ou des colis. Une lettre est caractérisée par : son poids (en grammes), le mode d'expédition (express ou normal), son adresse de destination, son adresse d'expédition, son format (A3 ou A4). Un colis est caractérisé par son poids (en grammes), le mode d'expédition (express ou normal), son adresse de destination, son adresse d'expédition, son volume (en litres).

On veut faire une implémentation de cette boîte aux lettres ainsi qu'un programme test (fonction « main ») en tenant compte des points suivants :

- On souhaite pouvoir afficher un courrier de la boîte aux lettres, que ce soit une lettre ou un colis ;
- On souhaite pouvoir calculer le prix du timbre d'un courrier de la boîte aux lettres, que ce soit une lettre ou un colis. Pour cela, on définit des règles de calcul du prix du timbre comme suit :
  - En mode expédition normal,
    - le montant nécessaire pour affranchir une lettre dépend de son format et de son poids : Formule : montant = tarif de base + 1.0 \* poids (kilos), où le tarif de base pour une lettre "A4" est de 250 et 350 pour une lettre "A3" ;
    - le montant nécessaire pour affranchir un colis dépend de son poids et de son volume : Formule :  $0,25 \times \text{volume} (\text{litres}) + \text{poids} \times 1.0$  ;
  - En mode expédition express
    - Les montants précédents sont doublés, quel que soit le type de courrier ;

On considère pour cela le fichier `main.cpp` fourni en annexe (qu'il s'agira de copier et de compléter)

1. Écrire les définitions des différentes classes (fichiers.h et fichiers.cpp pour la résolution de ce problème).
2. Vous allez maintenant définir la classe « Boîte aux lettres » caractérisée par :
  - Sa hauteur
  - Son volume
  - La liste de ses courriers

La classe « Boîte aux lettres » sera dotée des méthodes suivantes :

- Un constructeur prenant en paramètre une hauteur et un volume
- Un destructeur
- Une méthode permettant d'ajouter une lettre ou un colis à la boîte aux lettres
- Une méthode permettant d'afficher l'ensemble des courriers de la boîte aux lettres avec leurs caractéristiques et notamment le prix du timbre ;

Complétez votre main avec :

- une instruction permettant de créer une boîte aux lettres nommée "EXPEDITIF" avec une hauteur et volume que vous proposerez et y d'ajouter deux lettres et deux colis;
- une instruction permettant d'afficher l'ensemble des courriers de la boîte aux lettres avec leurs caractéristiques;

**Programmation Orientée Objet 1**

**Deuxième Session**

**Licence 3**

**Durée : 2h**

**Documents non autorisés**

**Exercice**

On veut écrire une application pour la gestion d'un institut s'occupant à la fois des étudiants et du personnel. Les étudiants ont chacun une moyenne annuelle, ils sont divisés en plusieurs groupes. Tout membre du personnel a un bureau. Dans le personnel, on distingue le personnel administratif du personnel enseignant. Chaque enseignant détient un numéro de casier. Tout membre du personnel reçoit un salaire à la fin du mois. Cependant, les vacataires, qui font partie du personnel enseignant, sont payés à l'heure et n'ont donc pas de salaire mensuel fixe. Chaque personne est désignée par un nom, une adresse et l'année où elle est arrivée à l'institut.

1. Dessiner la hiérarchie de classes correspondante.
2. Écrire les définitions des différentes classes ( fichiers.h) pour la résolution de ce problème.
3. Vous allez maintenant définir une classe « Institut » caractérisée par :
  - Le nom de l'institut
  - La liste des membres (étudiants et personnel) de l'institut

La classe « Institut » sera dotée des méthodes suivantes :

- Un constructeur prenant en paramètre une chaîne de caractères pour initialiser le nom de l'institut
- Un destructeur
- Une méthode permettant d'ajouter un membre à la liste des membres de l'institut
- Une méthode permettant d'afficher l'ensemble des membres avec leurs caractéristiques
- Une méthode permettant d'afficher le nombre moyen d'années pendant lesquelles les membres enregistrés ont fréquenté l'institut.

Complétez votre main avec :

- une instruction permettant de créer un Institut nommé "AVENIR" et ajoutant à ses membres un enseignant permanent, une secrétaire, un étudiant et un vacataire;
- une instruction permettant d'afficher l'ensemble des membres de l'institut avec leurs caractéristiques;
- une instruction permettant d'afficher le nombre moyen d'années pendant lesquelles les membres enregistrés ont fréquenté l'institut.