# INF1010

## *Programmation Orientée-Objet*

### Travail pratique #1

### Allocation dynamique, composition et agrégation

| | |
|---|---|
| **Objectifs :** | Permettre à l'étudiant de se familiariser avec les notions de base de la programmation orientée objet, l'allocation dynamique de la mémoire, le passage de paramètres, les méthodes constantes et les principes de relation de composition et d'agrégation |
| **Remise du travail :** | Lundi 2 février 2015, 8h |
| **Références :** | Notes de cours sur Moodle & Chapitre 2-7 du livre Big C++ 2e éd. |
| **Documents à remettre :** | Les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip. |
| **Directives :** | Directives de remise des Travaux pratiques sur Moodle<br>Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires.<br>Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.<br>Veuillez suivre le guide de codage |

## Informations préalables :

***La directive de précompilation « #ifndef »***

La directive de précompilation « #ifndef » signifie « if not defined » (si non défini). Comme le type de directive le laisse deviner, cette directive est évaluée avant la phase de compilation du code source. Dans les travaux pratiques, vous l'utiliserez dans les fichiers d'en-têtes (.h), pour éviter la double inclusion. Par exemple, un fichier peut inclure deux fichiers d'entête, soit a.h et b.h. Cependant, il se peut que b.h inclure lui-même le fichier a.h. On se retrouve donc à inclure deux fois le fichier a.h, ce qui entrainerait une erreur de compilation, car on ne peut définir deux fois la même classe. La directive « #ifndef » nous évite donc cette double inclusion. Pour utiliser la directive « #ifndef », il faut respecter la syntaxe suivante :

```
#ifndef NOMCLASSE_H
#define NOMCLASSE_H
// Définir la classe NomClasse ici
#endif
```

***La directive de précompilation « #include»***

La directive de précompilation pour l'inclusion de fichiers « #include »

1. #include <nom_fichier>
2. #include "nom_fichier"

Ce qui différencie ces deux expressions est l'emplacement où le fichier spécifié est recherché. Pour la seconde forme, le précompilateur commence tout d'abord par rechercher dans le même répertoire que le fichier compilé. Par la suite, il procède de la même manière que la première forme, c'est-à-dire normalement dans des répertoires prédéfinit par l'environnement de développement intégré.

En résumé, généralement, lorsqu'on inclut un fichier source qui se trouve dans le projet, on utilise la seconde forme. Lorsqu'au contraire on inclut un fichier qui provient d'une bibliothèque externe au projet, on utilise la première forme.

# Travail à réaliser

Votre travail consiste à réaliser un programme qui représente une forme simplifiée des sections de cours d'établissement universitaire.

Vous devez réaliser la définition et l'implémentation des classes *Etudiant, Professeur, Section* et *Ecole*. Pour vous aider, le fichier d'en-tête (.h) de la classe *Ecole* vous est fourni. Vous devrez implémenter les méthodes du fichier source (.cpp) correspondant, ainsi que les classes complètes manquantes (*Etudiant, Professeur* et *Section*).

# Classe *Etudiant*

Créer une classe « Etudiant » qui représente un étudiant dans une université. Cette classe contient les attributs suivants :

- Matricule (string)
- Prénom (string)
- Nom (string)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les trois attributs à la chaîne de caractères vide.
- Un constructeur par paramètres qui initialise les trois attributs selon les paramètres. Trois paramètres - un pour chaque attribut.

- Un destructeur.
- Les méthodes d'accès et de modification des attributs.
- Une méthode d'affiche des informations qui concernent un étudiant.

## Classe *Professeur*

Créer une classe « Professeur » qui représente un professeur dans une université. Cette classe contient les attributs suivants :

- Numéro d'employé (string)
- Prenom (string)
- Nom (string)
- Departement (string)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les quatre attributs à la chaîne de caractères vide.
- Un constructeur par paramètres qui initialise les quatre attributs selon les paramètres.
- Un destructeur.
- Les méthodes d'accès et de modification des attributs.
- Une méthode d'affichage des informations qui concernent un professeur.

## Classe *Section*

La classe « Section » contient les informations et méthodes qui concernent les sections de cours dans une université. Cette classe contient les attributs suivants :

- Sigle du cours donné par la section (string)
- Local de la section (string)
- Titre du cours donné par la section (string)
- Professeur (pointeur sur la classe Professeur) : 1 seul professeur.
- Tableau d'étudiants (pointeur) : maximum de 75 étudiants par section.
- Nombre d'étudiants (unsigned int).

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs sigle, local et titre à la chaîne de caractères vide. L'attribut du nombre d'étudiants est initialisé à 0. De plus, le tableau d'étudiants doit être initialisé avec une taille de 75.
- Un constructeur par paramètres avec les paramètres sigle (string), local (string), titre (string) et professeur (pointeur).
- Un destructeur qui libère la mémoire pour le pointeur sur professeur et sur le tableau d'étudiants.
- Les méthodes d'accès et de modification des variables membres (sauf pour le tableau d'étudiants).
- L'attribut *nombreEtudiants* est le nombre d'étudiants.

- Une méthode qui ajoute un étudiant dans la section. Cette méthode retourne un booléen qui indique si l'étudiant a été correctement ajouté ou pas. Si l'étudiant est déjà inscrit dans la section ou si elle est déjà remplie, alors l'étudiant ne doit pas être ajouté dans la section. La capacité du tableau sera définie par une constante dans la section *Section* d'une valeur de 75.
- Une méthode d'affichage des informations qui concernent une section.

## Classe *Ecole*

La classe « Ecole » contient les informations et méthodes qui concernent les universités. Cette classe contient les attributs suivants :

- Nom de l'école (string)
- Adresse de l'école (string)
- Tableau de sections ( tableau de pointeur sur *Section* ==> Section* sections_[50]) : maximum de 50 sections par école.
- Nombre de sections (unsigned int)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les paramètres nom et adresse à la chaîne de caractères vide et l'attribut nombre de sections à 0.
- Un constructeur par paramètres avec les paramètres nom (string) et adresse (string). L'attribut nombre de sections à 0.
- Un destructeur qui libère la mémoire pour le tableau de sections.
- Les méthodes d'accès et de modification des variables membres (sauf pour le tableau de pointeurs de sections).
- La variable *nombreSections* correspond au nombre de sections.
- Une méthode qui ajoute une section dans l'école. Cette méthode retourne un booléen qui indique si la section a été correctement ajoutée ou pas. Si la section prend un local déjà occupé ou si l'école ne possède plus de places disponibles, alors la section ne doit pas être ajoutée dans l'école. La capacité du tableau sera définie par une constante dans la classe *Ecole* d'une valeur de 50.
- Une méthode qui supprime une section de l'école selon son local. Cette méthode retourne un booléen qui indique si la section a été correctement supprimée de l'école. Cette méthode prend en paramètre le local (string). Il faut s'assurer de décaler le tableau de sections selon la section qui a été supprimée pour ne pas avoir certaines cases vides du tableau ou placer la dernière section à la position de la section supprimée.
- Une méthode d'affichage des informations qui concernent une école.

## Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées.

# Correction

La correction du TP1 se fera sur 20 points. Voici les détails de la correction:

- (10 points) Compilation et exécution exactes des différentes méthodes;
- (02 points) Documentation du code et bonne norme de codage;
- (04 points) Utilisation correcte du mot-clé *const* et dans les endroits appropriés;
- (02 points) Utilisation adéquate des directives de précompilation;
- (02 points) Allocation et désallocation appropriée de la mémoire;