Lab4/Devoir4 – CSI2772A
Mardi/Vendredi 10/13 Octobre 2023
SITE - Université d'Ottawa
Dû EN LIGNE le vendredi 27 Octobre à 22:00
En groupes d'au plus deux étudiant(e)s.
/10

I. Objectifs

- Notions de classe, Encapsulation de données et Fonctions membres
- Utilisation des objets, passage des objects à une fonction
- Constructeur et destructeur
- Constructeur de recopie

II. Rappels

II.1. Notions de classe

Une **classe** est une **structure** dans laquelle seulement certains membres et/ou fonctions membres seront « publics », c'est-à-dire accessibles « de l'extérieur », les autres membres étant dits « privés ». Les fonctions membres (méthodes) permettent de manipuler les données membres.

```
class Nom
{
    données membres et fonctions membres
};
```

Quelques règles d'écritures générales
- Le nom d'une classe commence généralement par une majuscule.
- Le nom des données membres est précédé d'un caractère souligné "_" (underscore).
- Les noms de méthode sont en minuscules.
- La définition des méthodes se fait en dehors de la déclaration de la classe. Seul le prototype des fonctions est indiqué.

II.1.a. Encapsulation de données: sections "private", "protected", et "public"

La définition d'une classe peut contenir plusieurs sections qui fixent pour l'application une permission d'accès aux données et fonctions membres. Pour cela, les mots clefs "**private**", "**protected**" et "**public**" sont utilisés. Le tableau suivant résume leur signification:

Mot clef	Signification
private	Les données et fonctions membres ne sont accessibles que par les fonctions membres de la classe
protected	Les données et fonctions membres ne sont accessibles que par les fonctions membres de la classe et de toutes classes dérivées (voir héritage)
public	Les données et fonctions membres sont accessibles dans toute l'application

Par défaut, la définition d'une classe commence comme une section "**private**".

II.2. Définitions des méthodes

II.2.a. Les différents types de fonctions membres

Il existe quatre types de fonctions membres:

- les **constructeurs** qui permettent d'initialiser automatiquement les objets à leur création.
- les **destructeurs** qui permettent de détruire automatiquement les objets à la fin de leur vie.
- les **accesseurs** qui permettent d'accéder à certaines informations contenues dans l'objet.
- les **manipulateurs** qui permettent de modifier les informations contenues dans l'objet.

II.2.b. Définition des méthodes (l'opérateur de portée "::")

Comme on définit les fonctions membres en dehors de la déclaration d'une classe, l'utilisation de l'opérateur de portée "**::**" est nécessaire afin de bien rattacher les méthodes à leur classe. La définition d'une fonction membre possède la syntaxe générale suivante:

Type_retour Nom_classe::nom_fonction(arguments)

III. Devoir 4

Exercice 1: (4 POINTS : 1.5 pts pour a) et 2.5 pts pour b)

Définir les classes suivantes dans les fichiers ci-joint (myFile1a.h et myFile1b.h respectivement):

a) ***Classe `Course`*** : représente un cours pris par un étudiant et qui contient le numéro du cours (*code*), et le nombre d'heures par cours (*hours*) comme données membres. Définir donc:
- *Course():* constructeur
- `int getNum()` *:* accesseur du code
- `int getHours()` : accesseur du nombre d'heures

b) Classe ***`Student`*** :
représente un étudiant et contient comme données membres le numéro d'identification ID de l'étudiant, le nombre maximum de cours qu'un étudiant peut suivre, le nombre de cours suivis par l'étudiant ainsi que la liste de ces cours (pointeur sur un tableau de pointeurs vers des objets `Course`), la liste des notes obtenues pour chacun de ces cours (pointeur sur un tableau d'entiers). Définissez donc ses méthodes :
- `Student(int, int)` : constructeur
- *~ Student () :* destructeur
- `double average()`: fonction qui renvoie la note moyenne de l'étudiant,
- `int totalHours()`*:* fonction qui renvoie le nombre total d'heures de cours que l'étudiant a suivi.
- `bool addCourse(Course*, int)` *:* ajoute un cours à *List_courses*. Renvoie vrai si on ne dépasse pas le nombre maximal de cours possible et faux sinon. Son 1er paramètre est le cours à ajouter et son 2ème paramètre est la note pour ce cours.

Un programme principal **`main`** vous est fourni ci-joint pour tester vos méthodes (myFile1.cpp).

/***SORTIE****/***
The total hours of Yan is 180
The average of Yan is 13.5

The total hours of Jane is 180
The average of Jane is 13.5
Enter a number to exit...

Exercice 2 : (6 POINTS : 0.75 pts pour chaque méthode)

La classe `SetInt` donnée ci-joint (myFile2.h) représente un ensemble d'entiers. Cette classe contient un constructeur sans paramètres qui crée un ensemble vide et un constructeur qui reçoit en paramètres un tableau d'entiers ainsi que sa taille et qui crée un ensemble d'entiers contenant les éléments de ce tableau. Un programme d'essai est donné ci-joint. La classe contient également un destructeur et le constructeur de copie, ainsi que des méthodes qui ont la fonction suivante :

- `add(int n);` permet d'ajouter un entier (n) à l'ensemble (s'il n'y appartient pas déjà) ;
- `remove(int n);` permet de supprimer un entier (n) de l'ensemble ;
- `bool contains(int n) ;` permet de tester l'appartenance d'un entier (n) à l'ensemble ; renvoie 1 si c'est le cas et 0 sinon.
- `int nbElem();` qui renvoie le nombre d'éléments de l'ensemble ;
- `int* tabElem();` qui renvoie un tableau d'entiers alloués dynamiquement contenant exactement les éléments de l'ensemble (si l'ensemble est vide cette fonction doit renvoyer NULL).
- `bool containsIn(int n, int& pos)` ; permet de tester si un entier (premier argument) appartient à l'ensemble à une position donnée (deuxième argument).

On impose ici l'utilisation d'un tableau d'entiers comme donnée membre de la classe qui contiendrait les éléments de l'ensemble. À tout moment, la taille du tableau sera égale au nombre d'éléments de l'ensemble, lorsque l'ensemble est vide, aucun tableau ne doit être alloué.

Un programme principal **`main`** vous est fourni ci-joint pour tester vos méthodes (myFile2.cpp).

Questions :
Définir les méthodes de la classe `SetInt`, incluant les constructeurs manquants et destructeur.

Note:

Vous n'êtes pas autorisés à modifier les déclarations des classes ni les programmes principaux fournis (`main`).

`/****Exemple de Sortie****/`
add an int element
10
add an int element
45
add an int element
21
add an int element
0
add an int element
568
a contains 10 :1
remove 10
a contains 10 :0
a contains :4 elements
The elements of a are :

45
21
0
568

Créer et soumettre un seul fichier zip

Directives

- Créez un répertoire que vous nommerez *Devoir43_ID*, où vous remplacerez ID par votre numéro d'étudiant (celui qui soumet le devoir).
  Mettez tous les fichiers suivants dans votre répertoire compressé *Devoir4_ID.zip* pour soumission dans le campus virtuel Brightspace.

  Fichiers :
  - ✓ *README.txt*
  - ✓ ***myFile1**.cpp*
  - ✓ ***myFile1a**.h*
  - ✓ ***myFile1b**.h*
  - ✓ ***myFile2**.cpp*
  - ✓ ***myFile2**.h*

- N'oubliez pas d'ajouter des commentaires dans chaque programme pour expliquer le but du programme, la fonctionnalité de chaque méthode et le type de ses paramètres ainsi que le résultat.
- Dans le répertoire *Devoir4_ID*, créez un fichier texte nommé *README.txt*, qui devra contenir **les noms des deux étudiant(e)s**, ainsi qu'une brève description du contenu :

Nom étudiant :
Numéro d'étudiant :
Code du cours : CSI2772A

Fraude scolaire :

Cette partie du devoir a pour but de sensibiliser les étudiants face au problème de fraude scolaire (plagiat). Consulter les liens suivants et bien lire les deux documents:
https://www.uottawa.ca/administration-et-gouvernance/reglement-scolaire-14-autres-informations-importantes

https://www.uottawa.ca/administration-et-gouvernance/sites/www.uottawa.ca.administration-et-gouvernance/files/processus_de_traitement_des_cas_de_fraude_academique_-_nov_2019.pdf

Les règlements de l'université seront appliqués pour tout cas de plagiat.
En soumettant ce devoir :
1. vous témoignez avoir lu les documents ci-haut ;
2. vous comprenez les conséquences de la fraude scolaire.