TP4 (4ème séance)
Programmation orientée objet (POO) en C++
Surcharge d'opérateurs et fonctions amies

Ce TP aborde les notions suivantes :

- Manipulation plus élaborée des classes et des objets
- **Fonctions amies**
- **Surdéfinition / surcharge d'opérateurs**

Chapitres 3,4,5 du polycopié : classes (fonctions amies), surcharge d'opérateurs

Consignes

- Ce sujet est disponible dans le dépôt `Partage` de votre promotion
- Réutilisez les classes définies dans les TP précédents

Exercice 1 : Définition d'une classe `Complexe` et surcharge d'opérateurs

Dans cet exercice vous devez définir une classe `Complexe` dans laquelle on définira deux membres privés : la partie réelle `_re` et la partie imaginaire `_im`

Vous devez prévoir :

- les différents constructeurs (défaut, paramétré, de copie)
- un destructeur
- une méthode membre d'affichage
- des accesseurs
- des fonctions permettant de faire l'addition, la soustraction et le produit de deux nombres complexes

Question 1

1. Définir la classe `Complexe`, ainsi que ses méthodes et fonctions membres. Rappelez-vous d'utiliser le principe de compilation séparée (`Complexe.h`, `Complexe.cpp`, `main.cpp`)

2. Testez votre classe à l'aide d'une fonction `main()`

Maintenant vous devez surcharger plusieurs opérateurs vous permettant de manipuler des nombres complexes. La surcharge des opérateurs est l'un des mécanismes qui nous permettent d'étendre les capacités des langages de programmation orientés objet. En C++, la déclaration et la définition d'une surcharge d'opérateur sont très similaires à la déclaration et à la définition de n'importe quelle fonction (voir le polycopié de cours).

Question 2

1. Définir les opérateurs « == » et « != » permettant de tester si deux nombres complexes sont égaux ou différentes. Utilisez des fonctions membres.

2. Définir l'opérateur d'affectation « = ». Utilisez une fonction membre.

3. Définir les opérateurs de calcul « + », « − » et « * » pour l'addition, la soustraction et le produit de deux nombres complexes. Utilisez des fonctions membres.

4. Surcharger les opérateurs >> et << pour permettre la saisie et l'affichage de nombres complexes. Utilisez des fonctions amies

   Par exemple : `Complexe c1; cin >> c1; cout << c1;`

5. Testez tous vos opérateurs dans votre fonction `main()`

Exercice 2 : Surcharge d'opérateurs de la classe `Vecteur3d`

En utilisant la classe `Vecteur3d` que vous avez définie et testée dans le TP2

Question 3

1. Définir les opérateurs « == » et « != » de manière à ce qu'ils permettent de tester la coïncidence ou la non- coïncidence entre deux vecteurs (à trois coordonnées). Utilisez des fonctions amies.

2. Définir l'opérateur « + » pour qu'ils fournisse la somme de deux vecteurs et l'opérateur « * » pour qu'il fournisse le produit scalaire de deux vecteurs (il est préférable d'utiliser les fonctions amies)

3. Surcharger les opérateurs >> et << pour permettre la saisie et l'affichage d'un vecteur
   Par exemple : `Vecteur3d v1; cin >> v1; cout << v1;`

4. Testez tous vos opérateurs dans votre fonction `main()`

Exercice 3 : Définition de la classe `String`

On désire concevoir une classe `String` permettant de manipuler des chaînes de caractères d'une longueur quelconque.

Question 4

1. Écrire la déclaration de la classe :
   - deux membres privés
     - `char* _ch;  /* chaine a allouer dynamiquement */`
     - `int _taille; /* taille réelle de la chaîne */`
   - constructeurs (à déduire du programme d'exemple) + destructeur

2. Ajouter les méthodes/fonctions décrites ci-dessous :
   - `int longueur()` : retourne la longueur réelle de la chaîne
   - `char& nieme(int i)` : retourne une référence au n-ième caractère
   - `void affiche()` : affiche la chaîne de caractères
   - `void saisie()` : saisie d'une chaîne d'au plus 1024 caractères
   - `void concatene(const char* ch)` : ajoute une chaîne `ch` ou un caractère à la fin de la chaîne
   - `String& minuscule()` : retourne une nouvelle chaîne formée des éléments de la chaîne d'origine mis en minuscule.
   - `friend int egal(const String ch1, const char* ch2)` : retourne 1 si les chaînes sont égales

3. Testez votre classe à l'aide du programme suivant :

```
int main()
{
   String ch1("essai");
   String ch2 = ch1;
   String ch3('=' , 80); // (1er caractere, taille max)

   cout << ch3.longueur() << endl;
   const String ch4("chaîne constante");

   ch1.nieme(1) = 'S'; // le caractère 1 de la chaîne
   ch1.affiche();

   ch2.saisie();
   ch2.concatene(" de la classe String");
   ch2.affiche();

   ch2.minuscule().affiche();

   if (egal(ch1,"eSsai"))
       cout << "c'est la meme chaine !" << endl;

   return 0;
}
```

Déposez dans Chamilo <u>un seul fichier .zip</u>
contenant uniquement 1 dossier par exercice