Université Paris Diderot **Langages à objets avancés**
M1 **Année 2015-2016**

TP n° 2

Tableaux, pointeurs (première partie)

Rappels : bonne pratique

— Mettez les `déclarations` d'une classe (ici, la classe `MaClasse`) dans un fichier `MaClasse.hpp`, (i.e. juste les déclarations des attributs et les prototypes des méthodes) ; la `définition` sera mise dans un fichier `MaClasse.cpp` (i.e. le code des méthodes). Donc, pour chaque classe, on a deux fichiers qui par commodité porteront le nom de la classe.

— Par ailleurs, pour éviter des problèmes de compilation, commencez le fichier `MaClasse.hpp` par

```
#ifndef MACLASSE
#define MACLASSE
```

Et finissez par

```
#endif
```

Mettez le nom après le `#define` en majuscule pour qu'il n'y ait pas d'ambiguïté avec le nom de la classe.

Note : Dans ce TP, on va utiliser des tableaux dont la taille est fixée dans la classe, ceci dans des situations où il serait normal de pouvoir la faire fixer par l'utilisateur. Nous apprendrons très vite à faire cela.

Polygones

Exercice 1 On reprend la classe `Point` du TP1. On y ajoute la méthode `translate`, qui prend deux doubles.

Si j'écris dans un main la suite d'instructions suivante que va-t-il se passer pour les points `p1`, `p2` et `p3` ?

```
Point p1(2,4);
Point p2 = p1;
Point *p3 = &p1;
p1.translate(1,1);
```

Après avoir cherché la réponse, testez en affichant ces points.

Exercice 2 On définit maintenant une classe `Polygone`, qui contient un tableau de 10 points (pas de pointeur ici !) et le nombre de points effectivement utilisés par le polygone.

Un polygone sera donc la donnée de 0 à 10 points.

— Écrivez un constructeur qui crée un polygone, sans aucun point. Puis vous écrirez deux méthodes permettant d'ajouter un point si c'est possible et répondant alors `true` et répondant `false` sinon. La première prendra en argument deux doubles, la deuxième prendra un point en argument.

— Testez en créant un quadrilatère, on testera d'abord en ne mettant pas (ou en le commentant) de constructeur sans argument à Point. Que se passe-t-il et pourquoi ?

— Ajoutez maintenant un constructeur sans argument à `Point`. Combien de points sont-ils créés en appelant le constructeur de `Polygone` ? Vérifier votre intuition en ajoutant un affichage dans les deux constructeurs de `Point`.

Exercice 3 On définit maintenant une classe `Polygone2` qui, elle, contiendra un tableau de 10 pointeurs sur `Point`, et toujours le nombre de points effectivement utilisés par le polygone.

Répondez aux mêmes questions que dans l'exercice précédent avec les modifications suivantes :

— On ne fera pas de méthode d'ajout de point prenant deux doubles en argument.

— Le prototype de la méthode prenant un point en argument devra être modifié.

Exercice 4 pour finir,

— Ajoutez une méthode qui translate un polygone dans les deux classes `Polygone` et `Polygone2`.

— Dans la classe `Polygone` seulement, ajoutez un constructeur qui prend un tableau de points en arguments.

Gestion de tâches : Modélisation

On considère les tâches que doivent exécuter un jour donné des employés d'une petite entreprise. Chaque employé a au plus 10 tâches à faire. Pour chaque tâche, on déclarera un certain nombre de dépendances : c'est-à-dire des tâches qui doivent être exécutées avant celle-ci. Par exemple, Jean ne peut mettre sous enveloppe la facture pour la société "Truc" que si Armand l'a rédigée et qu'elle a été validée par Béatrice. Et Judith ne peut aller à la poste que lorsque tous les courriers prévus ce jour-là ont été mis sous enveloppe.

Le nombre de dépendances par tâche est au maximum 10. Les dépendances induites ne seront pas forcément stockées.

— Décrivez des classes `Tache` et `Employe` et `Entreprise` correspondantes. Une tâche aura un numéro donné par l'entreprise qui se chargera de les prendre tous différents. C'est ce numéro qui permettra de comparer deux tâches entre elles.

— Écrivez ces classes.

— Écrivez ensuite une méthode permettant de trouver (s'il en existe une) une des tâches que l'on peut faire tout de suite, i.e. qui ne dépend d'aucune autre.

— Déduisez-en un algorithme permettant de donner un ordonnancement possible de ses tâches. (Cet algorithme n'est pas optimal, l'ordonnancement trouvé non plus a priori.)