TP-5 POO en C++ (5ème Séance)

Ce TP aborde les points suivants :

- **Héritage simple**
- **Hiérarchie d'héritage**
- **Héritage multiple**
- Polymorphisme
- Fonction virtuelle pure et classe abstraite

Chapitre 6 du Polycopie : héritage et polymorphisme

Exercice 1 : L'héritage simple

On dispose d'un fichier ***point.h*** contenant la déclaration suivante de la classe **Point**:

```
#include <iostream.h>
class Point {
     public :
          Point(double abs=0.0, double ord=0.0){_x=abs ; _y=ord ;}
          Void affiche(){cout<<"coordonnees"<<_x <<" "<<_y<<"\n" ;}
          Void deplace (double dx, double dy){_x=_x+dx ;_y=_y+dy ;}
     private :
          double _x, _y ;

};
```

Q1.1 Créer une classe **Pointcol**, dérivée de **Point**, comportant :
- Un membre donnée supplémentaire ***cl*** de type ***int*** destiné à contenir la couleur d'un point
- Les fonctions membres suivantes : « ***affiche*** »(redéfinie), qui affiche les coordonnées et la couleur d'un objet de type « **Pointcol »** ; « ***colore*** (***int*** couleur) », qui permet de définir la couleur d'un objet de type « **Pointcol »** ; un **constructeur** définissant la couleur et les coordonnées (non *inline*).

Q1.2 Que fera alors précisément cette instruction : Pointcol P1(2.5, 3.25, 5);

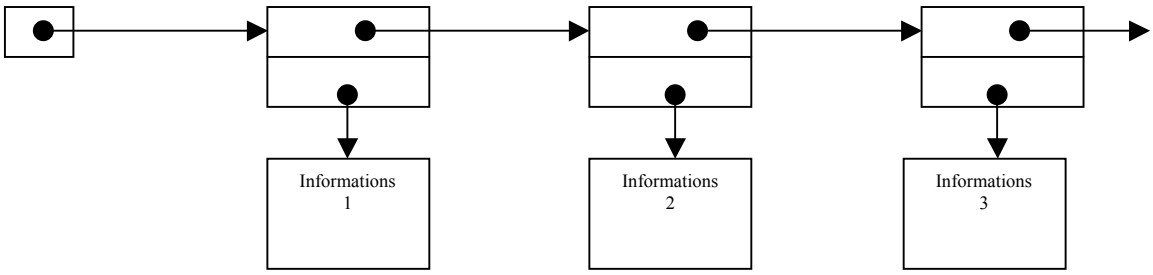
Q1.3 Que faudrait-il faire pour créer une classe **Pointcol** sans héritage mais possédant les mêmes caractéristiques. Quelles différences apparaîtront au niveau des possibilités d'utilisation de ces deux versions.

Exercice 2 : Hiérarchie d'héritage

Q2.1 Recopier l'exemple du cours (Gestion salariés).
Q2.2 Compléter le programme en ajoutant un fichier principal qui contient le «main». Corriger les erreurs.
Q2.3 Ajouter une ou des données et une ou des fonctions permettant de retourner: le nombre de salarié de l'entreprise, nombre de directeurs, employers, commerciaux. Exécuter et Valider.

Exercice 3 : Héritage multiple

On souhaite créer une liste permettant de manipuler des listes chaînées dans lesquelles la nature de l'information associée à chaque nœud de la liste n'est pas connue (par la classe). Une telle liste correspondra au schéma suivant :



La déclaration de la classe liste se présentera ainsi :

```
struct element {
     element *suivant ; void *contenu ;
} ;
class Liste {
     public :   Liste() ;
                ~Liste() ;
                void ajoute(void *) ;
                void *premier() ;
                void *prochain() ;
                int fini() ;
     private :
          element *_debut ;
} ;
```

La fonction « ***ajoute*** » permet d'ajouter un élément pointant sur une information en début de liste dont l'adresse est fournie en argument.
Pour explorer la liste, on a prévu trois fonctions :
- « ***premier*** », qui fournira l'adresse de l'information associée au premier nœud de la liste et qui, en même temps, préparera le processus de parcours de la liste,
- « ***prochain*** », qui fournira l'adresse de l'information associée au « prochain nœud » ; des appels successifs de prochain devront permettre de parcourir la liste (sans qu'il soit nécessaire d'appeler une autre fonction),
- « ***fini*** », qui permettra de savoir si la fin de la liste est atteinte ou non.

Q3.1 Compléter la déclaration précédente de la classe liste et en fournir la définition de manière à ce qu'elle fonctionne comme demandé.

Soit la classe **Point** suivante :

```
class Point {
     public :
        Point (double abs=0.0, double ord=0.0) {_x=abs ; _y=ord ; }
        void affiche() {cout<<"coordonnees" <<_x <<" "<<_y<<"\n" ;}
     private :
        double  _x, _y ;
     };
```

Q3.2 Créer une classe **ListePoints**, dérivée à la fois de Liste et de Point , pour qu'elle puisse permettre de manipuler des listes chaînées de points, c'est-à-dire des listes comparables à celles présentées ci-dessus, et dans lesquelles l'information associée est de type point. On devra pouvoir notamment :

- Ajouter un point en debut d'une telle liste,
- Disposer d'une fonction membre affiche affichant les informations associées à chacun des points de la liste de points

Q3.3 Ecrire un programme pour tester ces classes.