POO sous C++ - TP2

Objectifs
Construction, destruction et initialisation des objets, Les fonctions amies, Surdéfinition des opérateurs

Exercice 1

On souhaite réaliser une classe *vecteur3d* permettant de manipuler des vecteurs à trois composantes. On prévoit que sa déclaration se présente ainsi :

```
class vecteur3d{
float x, y, z ; // pour les 3 composantes (cartésiennes)
.....
} ;
```

On souhaite pouvoir déclarer un vecteur, soit en fournissant explicitement ses trois composantes, soit en en fournissant aucune, auquel cas le vecteur créé possédera trois composantes nulles. Écrire le ou les constructeurs correspondants :
a. en utilisant des fonctions membre surdéfinies ;
b. en utilisant une seule fonction membre ;
c. en utilisant une seule fonction en ligne.

Exercice 2

Soit une classe *vecteur3d* définie comme suit :

```
class vecteur3d{
float x, y, z ;
public :
vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0) {
x = c1 ; y = c2 ; z = c3 ;
}
.....
} ;
```

Introduire une fonction membre nommée *coincide* permettant de savoir si deux vecteurs ont les mêmes composantes :
a. en utilisant une transmission par valeur ;
b. en utilisant une transmission par adresse ;
c. en utilisant une transmission par référence. Si *v1* et *v2* désignent 2 vecteurs de type *vecteur3d*, comment s'écrit le test de coïncidence de ces 2 vecteurs, dans chacun des 3 cas considérés ?

Exercice 3

Soit une classe *vecteur3d* définie comme suit :

```
class vecteur3d{
float x, y, z ;
public :
vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0){
x = c1 ; y = c2 ; z = c3 ;
}
.....
} ;
```

Introduire, dans cette classe, une fonction membre nommée normax permettant d'obtenir, parmi deux vecteurs, celui qui a la plus grande norme. On prévoira trois situations :
a. le résultat est renvoyé par valeur ;
b. le résultat est renvoyé par référence, l'argument (explicite) étant également transmis par référence ;
c. le résultat est renvoyé par adresse, l'argument (explicite) étant également transmis par adresse.

Exercice 4
Réaliser une classe *vecteur3d* permettant de manipuler des vecteurs à 3 composantes (de type *float*). On y prévoira :
• un constructeur, avec des valeurs par défaut (*0*),
• une fonction d'affichage des *3* composantes du vecteur, sous la forme :
`< composante1, composante2, composante3 >`
• une fonction permettant d'obtenir la somme de 2 vecteurs ;
• une fonction permettant d'obtenir le produit scalaire de 2 vecteurs.
On choisira les modes de transmission les mieux appropriés. On écrira un petit programme utilisant la classe ainsi réalisée.

Exercice 5
Comment concevoir le type classe *chose* de façon que ce petit programme :

```
main(){
chose x ;
cout << "bonjour\n" ;
}
```

fournisse les résultats suivants :

```
création objet de type chose
bonjour
destruction objet de type chose
```

Que fournira alors l'exécution de ce programme (utilisant le même type *chose*) :

```
main(){
chose * adc = new chose
}
```

Exercice 6
Quels seront les résultats fournis par l'exécution du programme suivant (ici, la déclaration de la classe *demo*, sa définition et le programme d'utilisation ont été regroupés en un seul fichier) :

```
#include <iostream>
using namespace std ;
class demo{
int x, y ;
public :
demo (int abs=1, int ord=0){ // constructeur I (0, 1 ou 2 arguments)
x = abs ; y = ord ;
cout << "constructeur I : " << x << " " << y << "\n" ;
}
demo (demo &) ; // constructeur II (par recopie)
~demo () ; // destructeur
} ;

demo::demo (demo & d){ // ou demo::demo (const demo \& d)
cout << "constructeur II (recopie) : " << d.x << " " << d.y << "\n" ;
x = d.x ; y = d.y ;
}
```

```
demo::~demo (){
cout << "destruction : " << x << " " << y << "\n" ;
}
main (){
void fct (demo, demo *) ; // proto fonction indépendante fct
cout << "début main\n" ;
demo a ;
demo b = 2 ;
demo c = a ;
demo * adr = new demo (3,3) ;
fct (a, adr) ;
demo d = demo (4,4) ;
c = demo (5,5) ;
cout << "fin main\n" ;
}
void fct (demo d, demo * add){
cout << "entrée fct\n" ;
delete add ;
cout << "sortie fct\n" ;
}
```

Exercice 7

Créer une classe *point* ne contenant qu'un constructeur sans arguments, un destructeur et un membre donnée privé représentant un numéro de point (le premier créé portera le *numéro 1*, le suivant le *numéro 2*...). Le constructeur affichera le numéro du point créé et le destructeur affichera le numéro du point détruit. Écrire un petit programme d'utilisation créant dynamiquement un tableau de 4 points et le détruisant.

Exercice 8

1. Réaliser une classe nommée *set_int* permettant de manipuler des ensembles de nombres entiers. On devra pouvoir réaliser sur un tel ensemble les opérations classiques suivantes : lui ajouter un nouvel élément, connaître son cardinal (nombre d'éléments), savoir si un entier donné lui appartient.

Ici, on conservera les différents éléments de l'ensemble dans un tableau alloué dynamiquement par le constructeur. Un argument (auquel on pourra prévoir une valeur par défaut) lui précisera le nombre maximal d'éléments de l'ensemble.

2. Écrire, en outre, un programme (*main*) utilisant la classe *set_int* pour déterminer le nombre d'entiers différents contenus dans 20 entiers lus en données.

3. Que faudrait-il faire pour qu'un objet du type *set_int* puisse être transmis par valeur, soit comme argument d'appel, soit comme valeur de retour d'une fonction ?

Exercice 9

Modifier l'implémentation de la classe précédente (avec son constructeur par recopie) de façon que l'ensemble d'entiers soit maintenant représenté par une liste chaînée (chaque entier est rangé dans une structure comportant un champ destiné à contenir un nombre et un champ destiné à contenir un pointeur sur la structure suivante). L'interface de la classe (la partie publique de sa déclaration) devra rester inchangée, ce qui signifie qu'un client de la classe continuera à l'employer de la même façon.

Exercice 10

Soit une classe vecteur3d définie comme suit :

```
class vecteur3d{
float x, y, z ;
public :
vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0){
x = c1 ; y = c2 ; z = c3 ;
}
```

```
} ;
```

Définir les opérateurs == et **!=** de manière qu'ils permettent de tester la coïncidence ou la non-coïncidence de deux points :
a. en utilisant des fonctions membre;
b. en utilisant des fonctions amies.

Exercice 11
Soit la classe *vecteur3d* ainsi définie :

```
class vecteur3d{
float x, y, z ;
public :
vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0){
x = c1 ; y = c2 ; z = c3 ;
}
} ;
```

Définir l'opérateur binaire + pour qu'il fournisse la somme de deux vecteurs, et l'opérateur binaire * pour qu'il fournisse le produit scalaire de deux vecteurs. On choisira ici des fonctions amies.

Exercice 12
Soit la classe vecteur3d ainsi définie :

```
class vecteur3d{
float v[3] ;
public :
vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0){
// à compléter
}
} ;
```

Compléter la définition du constructeur (en ligne), puis définir l'opérateur [] pour qu'il permette d'accéder à l'une des trois composantes d'un vecteur, et cela aussi bien au sein d'une expression (**... = v1[i]**) qu'à gauche d'un opérateur d'affectation (**v1[i] = ...**) ; de plus, on cherchera à se protéger contre d'éventuels risques de débordement d'indice.

Exercice 13
Soit la classe point créée dans l'exercice 10, dont la déclaration était la suivante :

```
class point
{
float x, y ;
public :
point (float, float) ;
void deplace (float, float) ;
void affiche () ;
}
```

Adapter cette classe, de manière que la fonction membre *affiche* fournisse, en plus des coordonnées du point, le nombre d'objets de type *point*.