

## TP 3

### Exercice 1

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

```
1 class vecteur3d
2 {
3     float x, y, z ; // pour les 3 composantes (cartésiennes)
4     .....
5 };
```

On souhaite pouvoir déclarer un vecteur, soit en fournissant explicitement ses trois composantes, soit en en fournissant aucune, auquel cas le vecteur créé aura trois composantes nulles. Écrire le ou les constructeurs correspondants :

- en utilisant des fonctions membre surchargées ;
- en utilisant une seule fonction membre ;
- en utilisant une seule fonction en ligne.

### Exercice 2

Soit une classe `vecteur3d` définie comme suit :

```
1 class vecteur3d
2 { float x, y, z;
3 public :
4     vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0)
5     { x = c1 ; y = c2 ; z = c3 ;
6     } .....
7 };
```

Introduire une fonction membre nommée `coincide` permettant de savoir si deux vecteurs ont les mêmes composantes :

- en utilisant une transmission par valeur ;
- en utilisant une transmission par adresse ;
- en utilisant une transmission par référence.

Si `v1` et `v2` sont deux vecteurs de type `vecteur3d`, comment écrire le test de coïncidence de ces 2 vecteurs, dans chacun des 3 cas considérés ?

### Exercice 3

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

```
1 main()
2 { chose x ;
3   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) :

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

## Exercise 4

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) :

```
1 #include <iostream>
2 using namespace std ;
3 class demo
4 { int x,y;
5 public :
6 demo (int abs=1, int ord=0)
7 {x=abs; y=ord; cout << "constructeur I : " << x << " " << y << "\n" ;
8 }
9 demo (demo &) ;
10 ~demo () ;
11 };
12
13 demo::demo (demo & d)
14 { cout << "constructeur II (recopie) : " << d.x << " " << d.y << "\n"
15 ;
16 x=d.x; y=d.y; }
17 demo::~~demo ()
18 { cout<<"destruction : "<<x<<" "<<y<<"\n";
19 }
20
21 main () {
22 void fct (demo, demo *) ;
23 cout << "debut main\n" ;
24 demo a ;
25 demo b = 2 ;
26 demo c = a ;
27 demo * adr = new demo (3,3) ;
28
29 // proto fonction independante fct
30 fct (a, adr) ;
31 demo d = demo (4,4) ;
32 c = demo (5,5) ;
33 cout << "fin main\n" ;
34 }
35 void fct (demo d, demo * add) {
36 cout << "entree fct\n" ;
37 delete add ;
38 cout << "sortie fct\n" ;
39 }
```

## Exercise 5

Créer une classe point ne contenant qu'un constructeur sans arguments, un destructeur et un membre donné privé représentant un numéro de point (le premier créera le point 1, le suivant le point 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.

## Exercise 6

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 éléments de l'ensemble dans un tableau alloué dynamiquement par le constructeur. Un argument (auquel on pourra voir 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 terminer 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 ?

## Exercise 7

Soit la classe point suivante :

```
1 class point
2 { int x,y;
3 public :
4 point (int abs=0, int ord=0)
5 { x = abs ; y = ord ;
6 } };
```

Écrire une fonction `affiche`, amie de la classe `point`, permettant d'afficher les coordonnées d'un point. On fournira par conséquent un fichier source contenant la nouvelle déclaration (définition) de `point` et un fichier source contenant la définition de la fonction `affiche`. Écrire un petit programme (main) qui crée un point de classe automatique et un point de classe dynamique et qui en affiche les coordonnées.

## Exercise 8

Créer deux classes (dont les membres sont privés) :

- l'une, nommée `vect`, permettant de représenter des vecteurs à 3 composantes de type `double` ; elle comportera un constructeur et une fonction membre d'affichage ;

- l'autre, nommé `matrice`, permettant de représenter des matrices carrées de dimension  $3 \times 3$  ; elle comportera un constructeur avec un argument (adresse d'un tableau de  $3 \times 3$  valeurs) qui initialisera la matrice avec les valeurs correspondantes.

Réaliser une fonction `ind` pendant le `prod` permettant de fournir le vecteur correspondant au produit d'une matrice par un vecteur. Écrire un petit programme de test. On fournira par la suite les deux déclarations de chacune des classes, leurs deux définitions, la définition de `prod` et le programme de test.

## Exercice 9

Soit une classe `vecteur3d` finie comme suit :

```
1 class vecteur3d
2 { float x, y, z;
3 public :
4 vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0)
5 { x = c1 ; y = c2 ; z = c3 ;
6 } };
```

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 10

Soit la classe `vecteur3d` ainsi définie :

```
1 class vecteur3d
2 { float x, y, z;
3 public :
4 vecteur3d (float c1=0.0, float c2=0.0, float c3=0.0)
5 { x = c1 ; y = c2 ; z = c3 ;
6 } };
```

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 11

Soit la classe `point` suivante :

```
1 class point
2 { int x, y;
3 public :
4 point (int abs=0, int ord=0)
5 { x = abs ; y = ord ;
6 }
7 // .....
8 };
```

a. La munir d'un opérateur de cast permettant de convertir un point en un entier (correspondant à son abscisse).

b. Soient alors ces déclarations :

```

1 point p ;
2 int n ;
3 void fct (int) ;

```

Que font ces instructions : `n = p;` // instruction 1 `fct (p);` // instruction 2

## Exercise 12

Quels résultats fournira le programme suivant :

```

1 #include <iostream>
2
3 using namespace std;
4
5 class point
6 { int x,y;
7 public :
8 point (int abs, int ord)
9 { x = abs ; y = ord ; }
10 operator int () // "cast" point --> int
11 {cout<<"***appelint()pourlepoint"<<x<<" "<<y<<"\n";
12 return x ; }
13 };
14 main()
15 { point a(1,5), b(2,8) ;
16 int n1, n2, n3 ;
17 n1 = a + 3 ; // instruction 1
18 cout << "n1 = " << n1 << "\n" ;
19 n2 = a + b ; // instruction 2
20
21 cout << "n2 = " << n2 << "\n" ;
22 double z1, z2 ;
23 z1 = a + 3 ; // instruction 3
24 cout << "z1 = " << z1 << "\n" ;
25 z2 = a + b ; // instruction 4
26 cout << "z2 = " << z2 << "\n" ;
27 }

```

## Exercise 13

Soient les deux classes suivantes :

```

1 class B
2 { //...
3 public :
4     B () ;// constructeur sans argument
5     B (int) ; // constructeur à 1 argument
6     // ...
7 };
8 class A
9 { //...
10 friend operator + (A, A) ;
11 public :
12 A () ; // constructeur sans argument

```

```

13 A (int) ; // constructeur à 1 argument entier
14 A (B) ; // constructeur à 1 argument entier de type B
15 // ... };

```

a. Dans un programme contenant les déclarations :

```

1 A a1, a2, a3 ;
2 B b1, b2, b3 ;

```

les instructions suivantes seront-elles correctes et, si oui, que feront-elles ?

```

1 a1 = b1 ; // instruction 1
2 b1 = a1 ; // instruction 2
3 a3=a1+a2; //instruction3
4 a3=b1+b2; //instruction4
5 b3=a1+a2; //instruction5

```

b. Comment obtenir le même résultat sans déclarer, dans A, le constructeur A(B) ?

## Exercice 14

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

```

1 class point
2 { float x,y;
3 public :
4 void initialise (float abs=0.0, float ord=0.0)
5 { x = abs ; y = ord ;
6 }
7 void affiche ()
8 { cout << "Point de coordonnées : " << x << " " << y << "\n" ;
9 }
10 float abs () { return x ; } float ord () { return y ; }
11 };

```

- Créer une classe pointb, dérivée de point comportant simplement une nouvelle fonction membre nommée rho, fournissant la valeur du rayon vecteur (première coordonnée polaire) d'un point.
- Méthode question, en supposant que les membres x et y ont des déclarateurs protégés (protected) dans point, et non plus privés.
- Introduire un constructeur dans la classe pointb.
- Quelles sont les fonctions membre utilisables pour un objet de type pointb ?

## Exercice 15

Quels seront les résultats fournis par ce programme :

```

1 #include <iostream>
2 using namespace std ;
3 class A
4 { int n;
5 float x ;
6 public :
7 A (int p = 2) {n=p;x=1;
8 cout << "** construction objet A : " << n << " " << x << "\n" ; }
9 };

```

```

10 class B
11 { int n;
12 float y ;
13 public :
14 B (float v = 0.0) {n=1; y=v;
15 cout << "** construction objet B : " << n << " " << y << "\n" ; }
16 };
17 class C : public B, public A {
18 int n;
19 int p ;
20 public :
21 C (int n1=1, int n2=2, int n3=3, float v=0.0) : A (n1), B(v) { n = n3
    ; p = n1+n2 ;
22 cout << "** construction objet C : " << n << " " << p << "\n" ; }
23 };
24 main()
25 { C c1 ;
26 C c2 (10, 11, 12, 5.0) ;
27 }

```

## Exercise 16

M[0000] me question que pr[0000] c[0000] demment, en remplaçant simplement l'en-t[0000] te du constructeur de C par :

```

1 C (int n1=1, int n2=2, int n3=3, float v=0.0) : B(v)

```

Bon courage!!!