



COURS et TP DE LANGAGE C++

Chapitre 14

Fonctions amies

Joëlle MAILLEFERT

joelle.maillefert@iut-cachan.u-psud.fr

IUT de CACHAN

Département GEII 2

CHAPITRE 14

FONCTIONS AMIES

Grâce aux fonctions amies, on pourra accéder aux membres privés d'une classe, autrement que par le biais de ses fonctions membres publiques.

Il existe plusieurs situations d'amitié:

- Une fonction indépendante est amie d'une ou de plusieurs classes.
- Une ou plusieurs fonctions membres d'une classe sont amie d'une autre classe.

I- FONCTION INDEPENDANTE AMIE D'UNE CLASSE

Exemple (à tester) et exercice XIV-1:

Dans l'exemple ci-dessous, la fonction *coincide* est AMIE de la classe *point*. C'est une fonction ordinaire qui peut manipuler les membres privés de la classe *point*.

```
#include <iostream.h> // fonction indépendante, amie d'une classe
#include <conio.h>

class point
{
private :
    int x,y;
public:
    point(int abs=0,int ord=0){x=abs;y=ord;}
    //déclaration de la fonction amie
    friend int coincide(point,point);
};

int coincide(point p,point q)
{
    if((p.x==q.x)&&(p.y==q.y))
        return 1;
    return 0;
}

void main()
{point a(4,0),b(4),c;
    if(coincide(a,b))cout<<"a coincide avec b\n";
    else cout<<"a est différent de b\n";
    if(coincide(a,c))cout<<"a coincide avec c\n";
    else cout<<"a est différent de c\n";
    getch();
}
```

Exercice XIV-2:

Reprendre l'exercice III-8 dans lequel une fonction membre de la classe **vecteur** permettait de calculer le déterminant de deux vecteurs:

Définir cette fois-ci une fonction indépendante AMIE de la classe vecteur.

II- LES AUTRES SITUATIONS D'AMITIE

1- Dans la situation ci-dessous, la fonction **fm_de_titi**, fonction membre de la classe TITI, a accès aux membres privés de la classe TOTO:

```
class TOTO
{
// partie privée
.....
.....
// partie publique
friend int TITI::fm_de_titi(char, TOTO);
};

class TITI
{
.....
.....
int fm_de_titi(char, TOTO);
};

int TITI::fm_de_titi(char c, TOTO t)
{
...
} // on pourra trouver ici une invocation des membres privés de l'objet t
```

Si toutes les fonctions membres de la classe TITI étaient amies de la classe TOTO, on déclarerait directement dans la partie publique de la classe TOTO: **friend class TITI;**

2- Dans la situation ci-dessous, la fonction **f_anonyme** a accès aux membres privés des classes TOTO et TITI:

```
class TOTO
{
// partie privée
.....
.....
// partie publique
friend void f_anonyme(TOTO, TITI);
};
```

```
class TITI
{
// partie privée
.....
.....
// partie publique
friend void f_anonyme(TOTO, TITI);
};

void f_anonyme(TOTO to, TITI ti)
{
...
} // on pourra trouver ici une invocation des membres privés des objets to et ti.
```

III- APPLICATION A LA SURCHARGE DES OPERATEURS

Exemple (à tester) et exercice XIV-3:

On reprend l'exemple V-1 permettant de surcharger l'opérateur + pour l'addition de 2 vecteurs.

On crée, cette fois-ci, une fonction AMIE de la classe **vecteur**.

```
#include <iostream.h>
#include <conio.h>

// Classe vecteur
// Surcharge de l'opérateur + par une fonction AMIE

class vecteur
{
private:
    float x,y;
public: vecteur(float, float);
    void affiche();
    friend vecteur operator+(vecteur, vecteur);
};

vecteur::vecteur(float abs =0, float ord = 0)
{x=abs; y=ord;}

void vecteur::affiche()
{cout<<"x = "<< x << " y = " << y << "\n";}

vecteur operator+(vecteur v, vecteur w)
{
    vecteur res;
    res.x = v.x + w.x;
    res.y = v.y + w.y;
    return res;
}

void main()
{vecteur a(2,6),b(4,8),c,d;
    c = a + b; c.affiche();
    d = a + b + c; d.affiche();
    getch();
}
```

Exercice XIV-4:

Reprendre l'exercice XIV-1: redéfinir l'opérateur == correspondant à la fonction **coïncide**.

Exercice XIV-5:

Reprendre les exercices V-2, V-3 et V-4: En utilisant la propriété de surcharge des fonctions du C++, créer

- une fonction membre de la classe **vecteur** de prototype

float vecteur::operator*(vecteur); qui retourne le produit scalaire de 2 vecteurs

- une fonction membre de la classe **vecteur** de prototype

vecteur vecteur::operator*(float); qui retourne le vecteur produit d'un vecteur et d'un réel (donne une signification à $v2 = v1 * h$;))

- une fonction AMIE de la classe **vecteur** de prototype

vecteur operator*(float, vecteur); qui retourne le vecteur produit d'un réel et d'un vecteur (donne une signification à $v2 = h * v1$;))

On doit donc pouvoir écrire dans le programme:

```
vecteur v1, v2, v3, v4;
```

```
float h, p;
```

```
p = v1 * v2;
```

```
v3 = h * v1;
```

```
v4 = v1 * h;
```

Remarque:

On aurait pu remplacer la fonction membre de prototype **vecteur vecteur::operator*(float);** par une fonction AMIE de prototype **vecteur operator*(vecteur, float);**

Exercice XIV-6:

Etudier le listing du fichier d'en-tête **complex.h** fourni au chapitre V et justifier tous les prototypes des fonctions.

IV- CORRIGE DES EXERCICES

Exercice XIV-2:

```
#include <iostream.h>
#include <conio.h>

// Classe vecteur
// Fonction AMIE calculant le déterminant de 2 vecteurs

class vecteur
{
private :
    float x,y;
public: vecteur(float, float);
    void affiche();
    friend float det(vecteur, vecteur);
};

vecteur::vecteur(float abs =0., float ord = 0.)
{x=abs; y=ord;}

void vecteur::affiche()
{cout<<"x = "<< x << " y = "<< y << "\n";}

float det(vecteur a, vecteur b)
{ // la fonction AMIE peut manipuler
  // les quantités b.x, b.y, a.x, a.y
  float res;
  res = a.x * b.y - a.y * b.x;
  return res;
}

void main()
{
    vecteur u(2,6),v(4,8);
    u.affiche(); v.affiche();
    cout <<"Déterminant de (u,v) = "<<det(u,v)<<"\n";
    cout <<"Déterminant de (v,u) = "<<det(v,u)<<"\n";
    getch() ;
}
```

Exercice XIV-4:

```
#include <iostream.h> // Surchage de l'opérateur ==

class point
{
private:
    int x,y;
public:
    point(int abs=0,int ord=0)
    {
        x=abs; y=ord;
    }
    // déclaration de la fonction amie
    friend int operator==(point,point);
};

int operator ==(point p, point q)
{
    if((p.x==q.x)&&(p.y==q.y))
        return 1;
    return 0;
}

void main()
{
    point a(4,0),b(4),c;
    if(a==b) cout<<"a coïncide avec b\n";
    else cout<<"a est différent de b\n";
    if(a==c) cout<<"a coïncide avec c\n";
    else cout<<"a est différent de c\n";
    getch();
}
```

Exercice XIV-5:

```
#include <iostream.h>
#include <conio.h>

class vecteur
{
private:
    float x,y;
public:
    vecteur(float,float);
    void affiche(); // surcharger +
    vecteur operator+(vecteur); // surcharger * : produit scalaire
    float operator*(vecteur);
    // surcharger * : vecteur (passé en paramètre)*scalaire retourné
    vecteur operator*(float);
    // surcharger * : vecteur (objet)*scalaire retourné
    friend vecteur operator*(float,vecteur);
};

vecteur::vecteur(float abs =0,float ord = 0)
{x=abs;y=ord;}
void vecteur::affiche()
{cout<<"x = "<<x<<" y = "<<y<<"\n";}
vecteur vecteur::operator+(vecteur v)
{
    vecteur res; res.x = v.x + x; res.y = v.y + y;
    return res;
}
float vecteur::operator*(vecteur v)
{
    float res = v.x * x + v.y * y;
    return res;
}
vecteur vecteur::operator*(float f)
{
    vecteur res; res.x = f*x; res.y = f*y;
    return res;
}
vecteur operator*(float f, vecteur v)
{
    vecteur res; res.x = f*v.x; res.y = f*v.y;
    return res;
}

void main()
{
    vecteur a(2,6),b(4,8),c,d; float p,h=2.0;
    p = a * b; cout<< p <<"\n";
    c = h * a; c.affiche(); d = a * h; d.affiche(); getch();
}
```