COURS et TP DE LANGAGE C++

Chapitre 11

Propriétés des fonctions membres

Joëlle MAILLEFERT

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

IUT de CACHAN

Département GEII 2

CHAPITRE 11

PROPRIETES DES FONCTIONS MEMBRES

I- SURDEFINITION DES FONCTIONS MEMBRES

En utilisant la propriété de surcharge des fonctions du C++, on peut définir plusieurs constructeurs, ou bien plusieurs fonctions membres, différentes, mais portant le même nom.

Exemple (à tester) et exercice XI-1: Définition de plusieurs constructeurs:
Indiquer quel constructeur est utilisé pour chacun des objets a, b, c.

```
#include <iostream.h> // Surcharge de fonctions
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      point();    // constructeur 1
      point(int abs);  // constructeur 2
      point(int abs,int ord);  // constructeur 3
      void affiche();
   };

    point::point()  // constructeur 1
   {
      x=0; y=0;
   }

    point::point(int abs)  // constructeur 2
   {
      x = abs; y = abs;
   }

    point::point(int abs,int ord)  // constructeur 3
   {
      x = abs; y = ord;
   }

    void point::affiche()
   {
      gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }
```

```
void main()
{
   point a, b(5), c(3,12);
   clrscr(); a.affiche(); b.affiche(); c.affiche(); getch();
}
```

Exercice XI-2: Surcharge d'une fonction membre
Écrire une deuxième fonction **affiche** de prototype **void point::affiche(char *message)**
Cette fonction donne la possibilité à l'utilisateur d'ajouter, à la position du point, un message sur l'écran.

II- FONCTIONS MEMBRES " EN LIGNE "

Le langage C++ autorise la description des fonctions membres dès leur déclaration dans la classe. On dit que l'on écrit une fonction " inline ".
Il s'agit alors d'une " macrofonction ": A chaque appel, le compilateur génère le code de la fonction et ne fait pas appel à un sous-programme.
Les durées d'exécution sont donc plus rapides mais cette méthode génère davantage de code.

Exemple (à tester) et exercice XI-3:

Comparer la taille des fichiers exXI_1.obj et exXI_3.obj

```
#include <iostream.h> // Fonctions en ligne
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
       point()
      {
         x=0; y=0; // constructeur 1
      }
       point(int abs)
      {
         x=abs; y=abs; // constructeur 2
      }
       point(int abs,int ord)
      {
         x=abs; y=ord; // constructeur 3
      }
      void affiche();
   };
```

```
 void point::affiche()
{
   gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
}

 void main()
{
   point a,b(5),c(3,12);
   a.affiche();
   b.affiche();
   c.affiche();
   getch() ;
}
```

III- INITIALISATION DE PARAMETRES PAR DEFAUT

Exemple (à tester) et exercice XI-4:

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

    class point
   {
   private:
      int x,y;
   // Fonctions membres " en ligne "
   public: // arguments par défaut
       point(int abs=0,int ord=2)
      {
         x=abs; y=ord; // constructeur
      }
      void affiche(char *message = "Position du point");
   };

    void point::affiche(char *message)
   {
      gotoxy(x,y-1); cout<<message;
      gotoxy(x,y);   cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a, b(40), c(3,12);
      char texte[10]="Bonjour";
      a.affiche();
      b.affiche("Point b");
      c.affiche(texte);
      getch();
   }
```

IV- OBJETS TRANSMIS EN ARGUMENT D'UNE FONCTION MEMBRE

Quand on passe comme paramètre à une fonction membre un objet de la classe à laquelle appartient cette fonction:

Exemple (à tester) et exercice XI-5:

```
#include <iostream.h>
#include <conio.h>
// objets transmis en argument d'une fonction membre
    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 2)
      {
         x=abs; y=ord;// constructeur
      }
      int coincide(point pt);
   };

    int point::coincide(point pt)
   {  // noter la dissymétrie des notations pt.x et x
      if ((pt.x == x) && (pt.y == y))
         return(1);
      else
         return(0);
   }

    void main()
   {
      int test1,test2;
      point a,b(1),c(0,2);
      test1 = a.coincide(b);
      test2 = b.coincide(a);
      cout<<"a et b:"<<test1<<" ou "<<test2<<"\n";
      test1 = a.coincide(c);
      test2 = c.coincide(a);
      cout<<"a et c:"<<test1<<" ou "<<test2<<"\n";
      getch() ;
   }
```

Noter que l'on rencontre la notation " pt.x " ou " pt.y " pour la première fois. Elle n'est autorisée qu'à l'intérieur d'une fonction membre (x et y membres privés de la classe).

On verra plus tard que le passage d'un objet par valeur pose problème si certains membres de la classe sont des pointeurs. Il faudra alors prévoir une allocation dynamique de mémoire via un constructeur.

Exercice XI-6: On définit la classe **vecteur** comme ci-dessous:

```
 class vecteur
{
private:
   float x,y;
public:
   vecteur(float abs,float ord);
   void homothetie(float val);
   void affiche();
};

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

 void vecteur::homothetie(float val)
{
   x = x*val;  y = y*val;
}

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

La mettre en oeuvre dans **void main(),** en ajoutant une fonction membre **float det(vecteur)** qui retourne le déterminant des deux vecteurs (celui passé en paramètre et celui de l'objet).

V- OBJET RETOURNE PAR UNE FONCTION MEMBRE

Que se passe-t-il lorsqu'une fonction membre retourne elle-même un objet ?

1- Retour par valeur

Exemple (à tester) et exercice XI-7: (la fonction concernée est la fonction **symetrique**)

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

// La valeur de retour d'une fonction est un objet
// Retour par valeur

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      point symetrique();
      void affiche();
   };

    point point::symetrique()
   {
      point res;
      res.x = -x;  res.y = -y;
      return res;
   }

    void point::affiche()
   {
      cout<<"Je suis en "<<x<<" "<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(1,6);
      a=b.symetrique();
      a.affiche();b.affiche();
      getch();
   }
```

2- Retour par adresse (***)

Quand on a besoin de retourner une adresse.

Exemple (à tester) et exercice XI-8:

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

// La valeur de retour d'une fonction est l'adresse d'un objet

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      point *symetrique();
      void affiche();
   };

    point *point::symetrique()
   {
      point *res;
      res = new point;
      res->x = -x;  res->y = -y;
      return res;
   }

    void point::affiche()
   {
      cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(1,6);
      a = *b.symetrique();
      a.affiche();b.affiche();
      getch();
   }
```

3- Retour par référence (***)

La valeur retournée l'est par référence. On en verra l'usage dans un prochain chapitre.

Exemple (à tester) et exercice XI-9:

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

// La valeur de retour d'une fonction est la référence d'un objet

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      point &symetrique();
      void affiche();
   };

    point &point::symetrique()
   {  // La variable res est obligatoirement static
      // pour retourner par référence
      static point res;
      res.x = -x;  res.y = -y;
      return res;
   }

    void point::affiche()
   {
      cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(1,6);
      a=b.symetrique();
      a.affiche();b.affiche();
      getch();
   }
```

Remarque: " res " et " b.symetrique "occupent le même emplacement mémoire (car " res " est une référence à " b.symetrique ". On déclare donc " res " comme variable static, sinon, cet objet n'existerait plus après être sorti de la fonction.

Exercice XI-10: Reprendre la classe **vecteur.** Modifier la fonction **homotéthie**, qui retourne le vecteur modifié. (prototype: **vecteur vecteur::homotethie(float val)**).

Exercice XI-11 (***): Même exercice, le retour se fait par adresse.

Exercice XI-12 (***): Même exercice, le retour se fait par référence.

VI- LE MOT CLE " THIS "

Ce mot désigne l'adresse de l'objet invoqué. Il est *utilisable uniquement* au sein d'une fonction membre.

Exemple (à tester) et exercice XI-13:

```
#include <conio.h>
#include <iostream.h>
// le mot clé this : pointeur l'objet sur l'instance de point
// utilisable uniquement dans une fonction membre
    class point
   {
   private:
      int x,y;
   public:
       point(int abs=0, int ord=0) // constructeur en ligne
      {
         x=abs; y=ord;
      }
      void affiche();
   };

    void point::affiche()
   {
      cout<<"Adresse: "<<this<<" - Coordonnées: "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a(5),b(3,15);
      a.affiche();b.affiche();
      getch();
   }
```

Exercice XI-14: Remplacer, dans l'exercice XI-6, la fonction **coïncide** par la fonction suivante:

```
    int point::coincide(point *adpt)
   {
      if ((this->x == adpt->x) && (this->y == adpt->y))
         return(1);
      else
         return(0);
   }
```

Exercice XI-15: Reprendre la classe **vecteur**, munie du constructeur et de la fonction d'affichage. Ajouter
- Une fonction membre **float vecteur::prod_scal(vecteur)** qui retourne le produit scalaire des 2 vecteurs.
- Une fonction membre **vecteur vecteur::somme(vecteur)** qui retourne la somme des 2 vecteurs.

VII- CORRIGE DES EXERCICES

Exercice XI-2:

```
#include <iostream.h> // Surcharge de fonctions
#include <conio.h>

    class point
   {
   private:
      int x,y;
   public:
      point();  // constructeur 1
      point(int abs);  // constructeur 2
      point(int abs,int ord);  // constructeur 3
      void affiche();
      void affiche(char *message); // argument de type chaîne
   };

    point::point()  // constructeur 1
   {
      x=0; y=0;
   }

    point::point(int abs)  // constructeur 2
   {
      x=y=abs;
   }

    point::point(int abs,int ord)  // constructeur 3
   {
      x = abs; y = ord;
   }

    void point::affiche() // affiche 1
   {
      gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void point::affiche(char *message) // affiche 2
   {
      gotoxy(x,y-1);cout<<message;
      gotoxy(x,y);cout<<"Je suis en "<<x<<" "<<y<<"\n";
   }

    void main()
   {
      point a,b(5),c(3,12); char texte[10] = "Bonjour";
      a.affiche(); b.affiche("Point b:"); c.affiche(texte); getch();
   }
```

Exercice XI-6:

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

// Classe vecteur - Fonction membre déterminant

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord);
      void homothetie(float val);
      void affiche();
      float det(vecteur v);
   };

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

    void vecteur::homothetie(float val)
   {
      x = x*val;  y = y*val;
   }

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

    float vecteur::det(vecteur w)
   {
      float res;
      res = x * w.y - y * w.x;
      return res;
   }

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

Exercice XI-10:

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

// Classe vecteur - Fonction homothétie - Retour par valeur

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord); // Constructeur
      vecteur homothetie(float val);
      void affiche();
   };

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

    vecteur vecteur::homothetie(float val)
   {
      vecteur res;
      res.x = x*val;  res.y = y*val;
      return res;
   }

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

    void main()
   {
      vecteur v(2,6),u;
      v.affiche();
      u.affiche();
      u = v.homothetie(4);
      u.affiche();
      getch();
   }
```

Exercice XI-11:

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

// Classe vecteur - Fonction homothétie - Retour par adresse

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord); // Constructeur
      vecteur  *homothetie(float val);
      void affiche();
   };

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

    vecteur *vecteur::homothetie(float val)
   {
      vecteur *res;
      res = new vecteur;
      res->x = x*val;  res->y = y*val;
      return res;
   }

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

    void main()
   {
      vecteur v(2,6),u;
      v.affiche();
      u.affiche();
      u = *v.homothetie(4);
      u.affiche();
      getch();
   }
```

Exercice XI-12:

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

// Classe vecteur -Fonction homothétie - Retour par référence

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float abs,float ord); // Constructeur
      vecteur &homothetie(float val);
      void affiche();
   };

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

    vecteur &vecteur::homothetie(float val)
   {
      static vecteur res;
      res.x = x*val;
      res.y = y*val;
      return res;
   }

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

    void main()
   {
      vecteur v(2,6),u;
      v.affiche();
      u.affiche();
      u = v.homothetie(4);
      u.affiche();
      getch();
   }
```

Exercice XI-14:

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

// Objets transmis en argument d'une fonction membre
// Utilisation du mot clé this

    class point
   {
   private:
      int x,y;
   public:
       point(int abs = 0,int ord = 0)
      {
         x=abs; y=ord; // constructeur
      }
      int coincide(point *adpt);
   };

    int point::coincide(point *adpt)
   {
      if ((this->x == adpt->x) && (this->y == adpt->y))
         return(1);
      return(0);
   }

    void main()
   {
      point a,b(1),c(1,0);
      cout<<"a et b:"<<a.coincide(&b)<<" ou "<<b.coincide(&a)<<"\n";
      cout<<"b et c:"<<b.coincide(&c)<<" ou "<<c.coincide(&b)<<"\n";
      getch();
   }
```

Exercice XI-15:

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

// Création d'une classe vecteur, avec constructeur, affichage
// Produit scalaire de 2 vecteurs

    class vecteur
   {
   private:
      float x,y;
   public:
      vecteur(float xpos,float ypos);
      vecteur somme(vecteur v);
      float prod_scal(vecteur v);
      void affiche();
   };

    vecteur::vecteur(float xpos=0,float ypos=0)
   {
      x = xpos; y = ypos;
   }

    float vecteur::prod_scal(vecteur v)
   {  // tester le passage par référence &v
      float res = (x * v.x) + (y * v.y);
      return (res);
   }

    vecteur vecteur::somme(vecteur v)
   {  // tester aussi le passage par référence &v
      vecteur res;
      res.x = x + v.x; res.y = y + v.y;
      return res;
   }

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

    void main()
   {
      vecteur a(3);a.affiche(); vecteur b(1,2);b.affiche();
      vecteur c(4,5),d;c.affiche();
      cout<<"b.c = "<<b.prod_scal(c)<<"\n";
      cout<<"c.b = "<<c.prod_scal(b)<<"\n";
      c = a.somme(b); d = b.somme(a);
      cout<<"Coordonnées de a+b:";c.affiche();cout<<"\n";
      cout<<"Coordonnées de b+a:";d.affiche();cout<<"\n"; getch();
   }
```